Desarrollo de aplicaciones web

Jeisson Hidalgo-Céspedes

Un enfoque de resolución de problemas y de apego a los estándares web

Desarrollo de aplicaciones web

Fragmento de muestra

Primera edición
Versión 1.0.4-2020-08-23
Publicado en San José, Costa Rica
http://jeisson.ecci.ucr.ac.cr/appweb/material/

ISBN (Pendiente) versión digital
ISBN (Pendiente) versión impresa

Diseño de las ilustraciones: Jeisson Hidalgo-Céspedes.
Fotografía de carátula adaptada de Piqsels con licencia *Creative Commons Zero - CC0*.

Este material terminó de ser preparado en enero de 2020. El contenido de este matarial hace referencia a la realidad de esta fecha, y naturalmente puede haber variado al momento en que se haga su lectura debido a la volátil esencia de la web.

Tabla de contenidos

Prefacio

Encuestas a grandes poblaciones de desarrolladores, como *2020 HackerRank Developer Skills Report* (116,648 informantes), *Stack Overflow Annual Developer Survey 2019* (~90,000 informantes), y *State of the Developer Nation, 17th edition* (~17,000 informantes), coinciden en que la web es el área que más provee empleo a los desarrolladores de software. En el último quinquenio se reporta a JavaScript continuamente como el lenguaje de programación más usado en la industria del software a nivel mundial, y la programación en este lenguaje es la habilidad que los empleadores más necesitan sobre cualquier otra al reclutar (Ravisankar, 2018). Sin embargo, menos de la mitad de los estudiantes recién graduados de carreras universitarias en computación dominan este lenguaje (Ravisankar, 2019), así como el desarrollo de aplicaciones web en general. Por consiguiente, la mayoría de profesionales han de adquirir esta habilidad dentro del contexto laboral tras su formación universitaria.

Para ayudar a los profesionales frente a esta necesidad, afortunadamente ha surgido una oferta asombrosa de materiales de aprendizaje vinculados al desarrollo web, como vídeos, libros, y cursos virtuales. Estos materiales buscan primordialmente satisfacer las necesidades de la industria, que impone presión al empleado para aprender de manera intensiva, en el menor tiempo posible, las tecnologías que la empresa requiere en los procesos de desarrollo para entregar puntualmente sus productos o servicios.

Un ejemplo de este movimiento son los recientes **campamentos de codificación** {*code bootcamp*}, donde los aprendices comen y hasta duermen en el mismo lugar de estudio, con el fin de aprovechar al máximo el tiempo de exposición a la nueva tecnología que han de aprender en unos pocos días. La oferta de estos campamentos tiene una orientación muy marcada hacia las tecnologías de moda demandadas por la industria, como Node.js y React. Al proveer capacitación en poco tiempo y usualmente con menos costo que carreras universitarias, algunos empleadores ven los campamentos como una alternativa a la educación formal (Ravisankar, 2020). Sin embargo, los desarrolladores formados a través de campamentos o materiales de autoaprendizaje tienden a ser considerados como *codificadores*, usualmente con funciones más distantes a la toma de decisiones y con menor retribución salarial que profesionales con educación universitaria (Ravisankar, 2020).

Por su parte, las carreras universitarias en computación buscan formar ingenieros, con énfasis en la resolución de problemas y el dominio de los fundamentos teóricos de la disciplina, lo cual requiere períodos más extensos de aprendizaje. El ideal ingenieril busca el *diseño* de soluciones que se mantienen correctas en el tiempo, y por tanto agnósticas a tecnologías de moda que evolucionan rápidamente o son descontinuadas de acuerdo a la cambiante dinámica del mercado.

Conscientes de la importancia de la web en el mercado laboral, algunos centros universitarios incorporan cursos de desarrollo web como parte de la malla curricular, al menos con carácter electivo. Al impartir uno de estos cursos, el docente universitario busca satisfacer los ideales de la academia de formar ingenieros capaces de resolver problemas con tecnología web. Sin embargo, las presiones de la industria hacen difícil encontrar materiales pedagógicos alineados con los ideales académicos.

Este material surge de esta necesidad vivencial del autor como docente del curso homónimo "Desarrollo de aplicaciones web" en la Universidad de Costa Rica. El propósito de este libro es ayudar a satisfacer ambos ideales, el de la academia de formar ingenieros capaces de resolver problemas

usando principios computacionales, y el de la industria de producir soluciones de calidad minimizando los costos. Para ayudar a llenar el vacío descrito, este material incorpora los siguientes aspectos distintivos:

- Teoría cognitiva de *resolución de problemas* y su aplicación en el desarrollo web.
- Diseño de soluciones siguiendo *paradigmas de programación*.
- Estudio de las *especificaciones oficiales y estándares* que documentan la tecnología web, que son menos cambiantes en el tiempo y con ellos se crean los *frameworks* para acelerar el desarrollo de aplicaciones.
- *Ejercicios y proyectos* para poner en práctica los aspectos anteriores.

Este material abarca algunos *frameworks* populares al momento de escribir, con el fin de acelerar la construcción de soluciones y proveer un acercamiento realista al contexto industrial. No se busca una cobertura exhaustiva de estas tecnologías por su naturaleza cambiante, sino promover en el lector el hábito de auto-aprendizaje a través de fuentes como documentaciones oficiales. Este hábito es sumamente valorado en el contexto laboral de la disciplina.

Audiencia

Este material está dirigido a estudiantes universitarios, autodidactas, o profesionales que deseen o necesiten aprender más sobre resolución de problemas con tecnología web. Es deseable, pero no obligatorio conocimiento sobre los siguientes temas afines.

1. *Redes de computadoras* para comprender los protocolos que sustentan la arquitectura web. Es deseable conocimiento sobre el modelo de referencia TCP/IP, sobre todo de la capa de aplicación, y protocolos como TCP, UDP, DNS, y FTP. Un resumen se puede consultar en el anexo A.
2. *Bases de datos* para proveer persistencia a los datos a comunicar en el sitio web. Es deseable conocimiento de bases de datos relacionales y no relacionales. El anexo B provee un resumen de bases de datos orientadas a documentos con MongoDB como motor.
3. *Control de versiones* para realizar los ejercicios y proyectos de este material. Debe tenerse en cuenta que el dominio de control de versiones es indispensable para el trabajo con la web en la industria, en especial, si es realizado en equipo. La persona lectora interesada puede consultar el libro libre Pro Git de Scott Chacón y Ben Straub.
4. *Expresiones regulares* para procesamiento de texto, tanto al usar un editor de texto como al escribir programas en JavaScript. Existen abundantes recursos en línea que permiten un aprendizaje interactivo, por ejemplo, RegexOne.
5. *Interfaz de comandos de Unix* para quienes deseen publicar sitios web en servidores basados en este sistema operativo. Este es el caso del sistema operativo más popular que sustenta la web en el mundo: Linux. Se puede consultar el libro libre *The Linux Command Line* por William Shotts.
6. *Virtualización* para poder probar un sitio web en diferentes arquitecturas, alojar un sitio web en un servidor virtual privado, o en la nube.
7. *Ilustración y edición de fotografía* para elaborar gráficos web, en especial en formato SVG, PNG, y JPEG. Esta necesidad es satisfecha trabaja como parte de un equipo web que cuente con diseñador(a) gráfico(a). Se recomienda conocimiento de ilustración, edición fotográfica. Se puede usar software libre como Inkscape y Gimp.

8. *Edición de audio, producción musical y efectos de sonido* para proveer sonido a sitios web multimediales, como videojuegos. Esta necesidad es satisfecha por músicos que colaboren en parte de un equipo web. También existen recursos reutilizables en la web que pueden ayudar para proyectos personales.

Convenciones

Se trató de proveer traducción de la totalidad de los términos inglés ampliamente usados en el contexto web. Algunos de ellos son transliteraciones del de este idioma, debido a la ausencia de correspondientes oficiales en español, por ejemplo, "renderizado" como traducción de *rendering*. Algunos términos se mantuvieron sin traducir por su amplio uso en el contexto ingenieril, como es el caso de *framework*. Cuando se considere apropiado, los textos en inglés se incluyen para mantener la precisión terminológica y se resaltan en itálicas.

El código fuente en los ejemplos de este material están escritos en inglés, y se recomienda al estudiante seguir esta práctica difundida en la industria. Se agradece enormemente al lector el reporte de errores que encuentre en este material.

El libro está distribuido en cuatro partes, cada una contiene capítulos, divididos en secciones, y éstas en apartados.

Ejercicios

Para maximizar el aprendizaje se recomienda a la persona lectora seguir una metodología que combine la asimilación teórica y su aplicación práctica. Para apoyar este método se entrelazan en el libro las *nociones* sobre la web con *ejercicios* prácticos con el fin de que el o la aprendiz aplique (o como su nombre lo indica, "ejercite") la teoría recién presentada. Si este material se usa como parte de un curso presencial, indague con su tutor si recibirá además crédito al resolverlos.

Se sugiere resolver los ejercicios en un repositorio de control de versiones personal. Cada ejercicio tiene un nombre en inglés que lo identifica de forma única dentro del material. Este identificador se incluye entre corchetes y es acompañado con una sugerencia de puntaje numérico que podría ser usado para calcular el crédito que recibirá en un curso presencial, o como un indicio de esfuerzo en un contexto autodidacta.

Ejercicio 1 [repository, 5 pts]

Para responder a los ejercicios de este material, cree un repositorio de control de versiones personal, y por tanto *privado*, en algún medio compartido, por ejemplo, GitHub, GitLab, o BitBucket.

Si su servicio de repositorios ofrece crear un archivo `.gitignore` escoja uno para Node o Node.js. Sino, cree un archivo `.gitignore` en la raíz del repositorio con al menos el siguiente contenido:

```
.env
.DS_Store
build*
node_modules
npm-debug.log
ehthumbs.db
Thumbs.db
```

Su repositorio eventualmente evolucionará a un sitio web personal. Clone el repositorio en su máquina local. Cuando se le solicite, cree una carpeta para las partes o capítulos de este material. Al finalizar el curso tendrá una lista de carpetas como la siguiente:

```
intro/
xml/
html/
css/
js/
jsc/
jss/
extra/
```

Cada vez que resuelva un ejercicio, cree una subcarpeta dentro de la carpeta del capítulo o parte al que pertenece. Utilice como nombre de la carpeta el identificador del ejercicio. Por ejemplo, use `intro/app_types` y no `intro/Ejercicio2`, ya que los números de ejercicios pueden cambiar porque son autogenerados. Por cada ejercicio que resuelva, recuerde hacer un *commit* en su repositorio de control de versiones.

Proyectos

Para formar la habilidad de resolución de problemas, indispensable en la disciplina, se recomienda a la persona lectora identificar alguna necesidad o deseo, sea propia o de terceros, que una aplicación web no trivial pueda resolver. Se recomienda enfáticamente aplicar las nociones conforme avanza en este material a la construcción de su aplicación web, al menos cuando alcance los recuadros titulados "Avance de proyecto".

Si participa en un curso presencial, indague sobre el requisito de resolver un problema de mayor complejidad. En caso afirmativo, recabe detalles como el porcentaje de crédito que recibirá, si el

problema a resolver es libre o acotado, si lo resolverá en forma individual, en pareja, o en equipos más numerosos, si es un desarrollo nuevo o debe ampliar código existente, la cantidad de avances o fechas de entrega, si usarán metodologías ágiles o en cascada, entre otros.

Avance de proyecto 1 [prj_repository, 5 pts]

Si va a realizar un proyecto en equipo o como parte de un curso presencial, indague los requisitos de control de versiones que administrará el código. Es probable que tenga que crear o unirse a un repositorio compartido. Si sigue el material de forma autodidacta o el proyecto es individual, puede crear una subcarpeta `project/` en su repositorio de control de versiones personal. O si lo prefiere, puede usar como nombre de la carpeta un identificador alusivo al proyecto.

En su repositorio o carpeta para el proyecto cree un archivo `README.md` y describa el problema a resolver como resultado de una lluvia de ideas. Conviene documentar los resultados de las indagaciones que hizo sobre el proyecto (solicitadas en el párrafo antes de este avance de proyecto).

Este material presenta ejemplos de problemas más extensos y realistas a los que se llaman **proyectos de ejemplo**, con el fin de ilustrar el proceso de resolución de problemas con tecnología web. Algunos recuadros presentan *avances de ejemplo*, que consisten en aplicar las nociones expuestas a los proyectos de ejemplo. Se recomienda al lector aprovechar los *avances de ejemplo* como apoyo para progresar en su propio proyecto. Si desea maximizar el aprovechamiento trate de resolver los *proyectos de ejemplo* primero, luego revise el *avance de ejemplo* para comparar los resultados. Los proyectos de ejemplo se listan a continuación.

1. **Bingo**. Es una aplicación web ideada para promover la interacción social del juego de mesa bingo. La aplicación facilita a un grupo de amigos o familiares jugar cuando no se tienen los implementos, o cuando se necesita apoyo para animar el juego. Este proyecto aprovecha la amplia difusión del juego para aplicar los conceptos web a la resolución de problemas.
2. **iyuök** (del bribri, "aprender a hacer") es un juez de programación en línea ideado para ayudar en el proceso de aprendizaje de la programación, en especial, del paradigma de programación concurrente. El proyecto tiene un fuerte énfasis en la experiencia de usuario y el uso de servicios web. Disponible en la edición completa de este material.

Agradecimientos

Este material ha sido preparado con software libre que merece el agradecimiento a las personas que han trabajado en hacerlo disponible al mundo, en específico:

Recurso	Descripción
AsciiDoc	Es un formato para especificar documentos en forma legible para humanos, con el que se escribió el contenido de este material.
Figma	Aplicación web para diseño de interfaces gráficas de usuario {*wireframes*} y prototipos interactivos {*prototyping*} que pueden ser probados con usuarios.
Geany	Editor de texto con el que se escribió el contenido, ejemplos de páginas web, hojas de estilo, y programas.

Recurso	Descripción
Gimp	Editor de imágenes escalares con el que se prepararon las fotografías, como por ejemplo, la portada del material.
Git	Este material se elaboró naturalmente bajo control de versiones.
Inkscape	Editor de imágenes vectoriales con el que se elaboraron diagramas y el diseño de la portada.
Libxml2	Herramientas de análisis de XML del proyecto Gnome usadas para validar los documentos en esta notación.
Railroad Diagram Generator	Herramienta en línea con la que se elaboraron los diagramas de sintaxis.
ThisPersonDoesNot Exist	Generador con inteligencia artificial de imágenes de personas inexistentes usadas en el método de análisis de usuarios *personas*.

Parte I. Introducción

Esta parte ayuda a la persona lectora a iniciar su aprendizaje en la solución de problemas mediante el uso de la tecnología web. El primer capítulo pretende que el lector pueda identificar una necesidad que quiera resolver mediante un sitio web que involucre una aplicación web, y elabore un diseño de ese sitio. El resto del libro está ideado para apoyar la construcción de esta solución.

El segundo capítulo introduce la arquitectura web, con el fin de ayudar a la persona desarrolladora a comprender los principios en los que se fundamenta la web, y al hábito de la lectura o consulta de los estándares. Cualquier tecnología que acelere o facilite el desarrollo web, dependerá ineludiblemente de estos principios. Aunque estas tecnologías de moda resultan atractivas, son muy volátiles, mientras que los estándares tienden a ser más estables en el tiempo. La persona que asimile los estándares no sólo podrá comprender mejor las tecnologías del momento, sino que podrá crear nuevas, e incluso, más adecuadas a una necesidad particular.

Capítulo 1. Resolución de problemas web

El objetivo principal de la ingeniería es la *resolución de problemas* con *artefactos*. Cada ingeniería difiere en los tipos de artefactos empleados para resolver problemas. A modo de ejemplo y en forma simplificada, la ingeniería civil emplea estructuras, la ingeniería mecánica emplea máquinas, y la computación emplea software.

El software típicamente se clasifica en tres grandes grupos. El software que controla hardware o alguna arquitectura se llama **software del sistema** {*system software*}, como son los sistemas operativos. Se le contrapone el software que usa al anterior para satisfacer necesidades de los usuarios, conocido como **software de aplicación** {*application software*} o simplemente *aplicaciones*. Son ejemplos de aplicaciones los paquetes de ofimática, editores de texto, y videojuegos que trabajan sobre los sistemas operativos. En un punto intermedio se encuentra el tercer grupo de **herramientas de desarrollo** {*development tools*}, que pueden tener acceso a bajo nivel del software del sistema y están ideadas para desarrolladores en lugar de usuarios finales.

El hardware no es la única arquitectura computacional disponible. Una de las más populares en la actualidad es la *arquitectura web* (Capítulo 2), cuyo "software del sistema" incluye los servidores web y navegadores web. Sobre este software del sistema web se puede construir *aplicaciones web* para resolver problemas de los usuarios. El propósito de este material es ayudar a la persona lectora a desarrollar la habilidad de resolver problemas mediante la construcción de sitios y aplicaciones web.

Aunque sea su propósito, la resolución de problemas no aparece con la disciplina de la ingeniería, sino que es una habilidad cognitiva humana y por tanto, una rama de estudio de la psicología. La **resolución de problemas** es el proceso que los seres vivos realizan para transformar una situación vivencial en una situación deseable, pero las acciones que la transformación requiere no son inmediatamente obvias (adaptado de (Robertson, 2016)).

Al tratar de resolver un *problema* es normal que se produzca un "bloqueo mental", una sensación de "no saber cómo iniciar". Esta circunstancia es distinta para un **ejercicio**, donde la persona conoce un modelo solución que puede aplicar de inmediato (y por tanto "ejercitar" la solución conocida). Por ejemplo, trate de calcular:

1. ¿Cuánto es el *mínimo* de varillas metálicas de 6 metros que se deben comprar para poder colgar las cortinas de un edificio que tiene 10 ventanas todas de 2.2 metros de ancho? Después de cortar las varillas y colgar las cortinas ¿cuántos metros de varilla se desperdician, dado que no se pueden soldar?
2. ¿Cuánto es el *mínimo* de varillas metálicas de 6 metros que se deben comprar para poder colgar las cortinas de un edificio que tiene 10 ventanas con anchos 2.1, 1.85, 0.8, 2.6, 1.92, 1.75, 1.2, 1.5, 2.33, y 3 metros? Después de cortar las varillas y colgar las cortinas ¿cuántos metros de varilla se desperdician, dado que no se pueden soldar?

La categorización en ejercicio y problema es subjetiva. Si para alguna de las dos situaciones anteriores encontró un modelo matemático que lo resuelve, habrá sido un *ejercicio* para usted, sino habrá identificado un *problema*.

El **problema** se caracteriza porque la persona no conoce de previo un modelo solución que pueda aplicar de inmediato, de ahí la sensación de bloqueo. Para lograr resolver el problema, la persona debe salir de este estado de bloqueo, en especial si se es una o un ingeniero ante un cliente. Para superar el estado de bloqueo y poder resolver problemas se ha de seguir un proceso de resolución de problemas.

1.1. Proceso de resolución de problemas

Un **proceso de resolución de problemas** es un conjunto ordenado de pasos que al seguirlos ayuda a las personas a resolver problemas. En 1945 el matemático húngaro George Pólya publicó un libro corto con problemas matemáticos, y en las primeras páginas describió un proceso compuesto de los siguientes cuatro pasos para resolver los problemas que seguían en el libro (Pólya, 1957).

1. Comprender el problema.
2. Idear un plan.
3. Ejecutar el plan.
4. Evaluar el plan.

Una cantidad considerable de adaptaciones se han realizado de los cuatro pasos de Pólya, como el proceso diagramado en la Figura 1 que este material propone para la resolución de problemas web, aunque es apto también para otros artefactos de software. Los pasos o fases del proceso se representan en la figura con aristas (flechas), y sus productos con rectángulos.

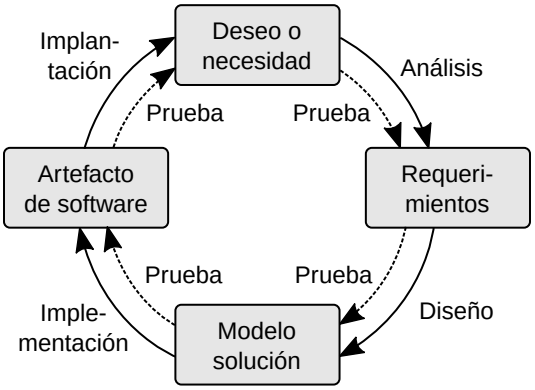


Figura 1. Proceso propuesto de resolución de problemas

El proceso de resolución de problemas de la Figura 1 parte de un deseo o necesidad de un cliente que puede ser satisfecha mediante el uso de una aplicación web. El equipo de desarrollo desconoce esta situación y realiza un **análisis** {*analysis*} para comprender claramente *qué* es lo que el cliente quiere o necesita. El equipo documenta los resultados en una especificación de *requerimientos*, normalmente en forma de documentos. Los requerimientos son probados con intervención del cliente, hasta obtener su visto bueno de que representan lo que realmente quiere o necesita, y tratando de

minimizar la ambigüedad.

Si no existe una solución reutilizable para los requerimientos del cliente, el equipo de desarrollo deberá idear una. En la fase de **diseño** {*design*} el equipo construye un *modelo solución* usando artefactos de un nivel alto de abstracción y fácil modificación, de forma análoga a los planos que un ingeniero civil elabora al diseñar estructuras. Dependiendo de las herramientas de diseño que se empleen, el modelo solución puede ser probado automáticamente, o en su defecto, por el equipo mismo. La computadora o el equipo de desarrollo sigue el modelo al pie de la letra aplicando entradas elaboradas previamente, y contrasta los comportamientos o salidas del modelo contra los esperados. Estas parejas de entradas y comportamientos esperados se llaman *casos de prueba* y son recopilados con participación del cliente.

Cuando el modelo logra resolver el problema para los casos de prueba, pasa a la siguiente fase. La **implementación** {*implementation*} es la traducción del modelo y los casos de prueba a códigos que pueden ser procesados o ejecutados por una computadora. El resultado es el artefacto ingenieril pretendido, que en este caso corresponde a software. Si éste es ejecutado por la computadora y pasa los casos de prueba, se implanta o integra con los sistemas en producción para uso del cliente.

Es recomendado que el proceso de desarrollo se realice en forma cíclica. En cada iteración se escoge y desarrolla un subconjunto de los requerimientos. Las metodologías ágiles de moda en el desarrollo web asignan períodos cortos, por ejemplo dos semanas, para realizar una iteración. El cliente interviene en las fases de implantación y análisis, lo que permite junto al equipo realizar ajustes en la administración del proyecto. Si el cliente participa además en las pruebas del diseño e implementación, se conoce como **diseño centrado en el usuario** {UCD, *user-centered design*}.

Rosenfeld, Morville, y Arango recomiendan realizar una primera iteración en cascada del ciclo de la Figura 1 para definir la *arquitectura de la información* del sitio web (Rosenfeld et al., 2015). Esta adaptación se ilustra en la Figura 2. El producto final de esta iteración son documentos de análisis, un mapa de sitio, bocetos de los tipos páginas, y opcionalmente un prototipo del sitio web. Esta primera iteración es la que se realizará en el resto de este capítulo con el *sitio web* global como objeto de estudio. Las iteraciones subsecuentes se realizarán en capítulos posteriores para crear las *páginas web* y *aplicaciones web* que conforman el sitio web.

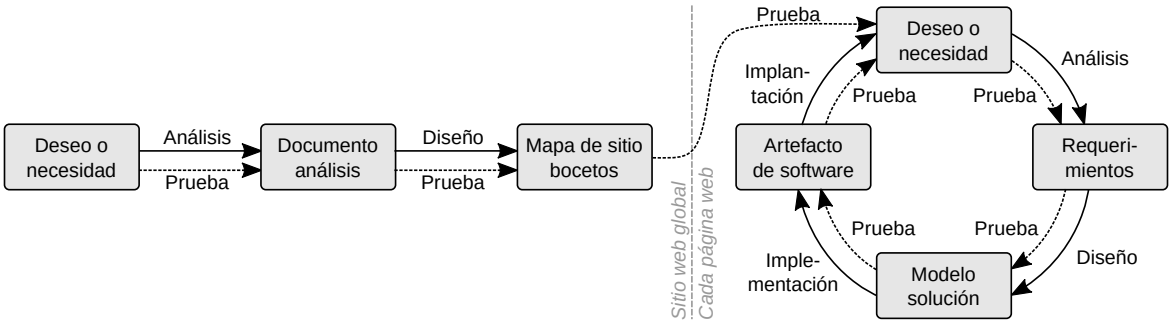


Figura 2. Proceso propuesto de creación del sitio web

1.2. El sitio web

La Figura 3 muestra los conceptos involucrados en un sitio web, y sus relaciones usando el *lenguaje unificado de modelado* {UML, *unified modeling language*}. UML es el lenguaje estándar para diseño de soluciones orientadas a objetos que se estudia con más detalle en el Capítulo 8. La Figura 3 incluye tres tipos de relaciones. La relación de agregación (◇⟶) indica que el objeto al lado del rombo "tiene" o "contiene" objetos del lado de la flecha. Se pueden usar números o rangos para indicar la cantidad de objetos que participan al lado de la flecha. La relación de generalización (⟶▷) se lee "es un(a)", dado que el objeto es un caso particular del objeto al lado de la fecha. Otros tipos de asociaciones se representan en el diagrama simplemente con una flecha etiquetada (⟶). Se usan números entre círculos para ligar el texto a continuación con su correspondiente representación en el diagrama.

Un **sitio web** {*website*} es una colección de páginas web (representado por la relación de agregación ① en la Figura 3). Una **página web** {*web page*} es un documento de *hipertexto* o *hipermedios* por su capacidad de contener recursos de diferentes medios de comunicación ② y de enlazar a otras páginas web ③. Un **recurso web** {*web resource*} es una pieza de información como un trozo de texto, una imagen, código de programas, un archivo de sonido o vídeo, entre otros ④. Como ilustra el diagrama de la Figura 3, una página web es también un recurso web (relación de generalización ⑤). Los recursos web son reutilizables, de tal forma que varias páginas de un sitio web pueden compartir los mismos recursos ②, y esto se considera una buena práctica.

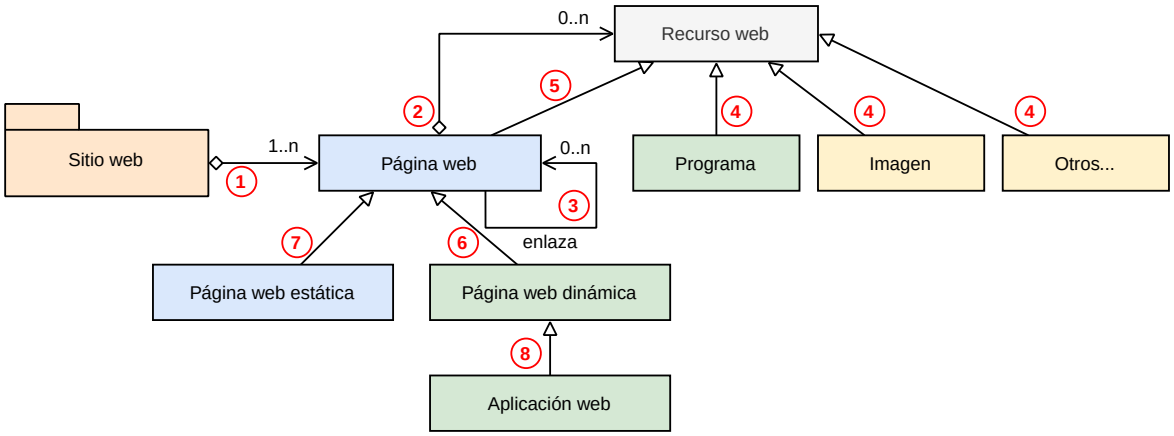


Figura 3. Relación conceptual entre un sitio web y los recursos que lo componen

Los enlaces constituyen el principal mecanismo que caracteriza al hipertexto (relación ③ en la Figura 3). El popular concepto de **la web** puede definirse como un sistema de documentos de hipertexto interrelacionados entre sí y accesibles a través de internet. Si se ilustran las páginas web como nodos y los enlaces como aristas, forman un grafo con apariencia de "tejido", a lo que en inglés se llama *web*, y a veces representado metafóricamente con una telaraña (*spider's web*).

De todas las páginas que conforman un sitio web, una tendrá el rol de **página de inicio** {*start page*}, **página principal** {*home page*}, o **página frontal** {*front page*}. Esta página será servida en caso de que un usuario acceda al sitio web sin especificar un recurso particular. Su propósito es ayudar al visitante a encontrar los recursos que se encuentran el sitio web y que son de su interés.

No siempre es claro determinar los límites donde termina un sitio web e inicia otro. Las páginas web

pueden interrelacionarse, incluso aunque pertenezcan a sitios web distintos. Una heurística es considerar los recursos identificados con el mismo *nombre de dominio* como parte del mismo sitio web, como `www.ejemplo.com`, aunque debe tenerse en cuenta que un mismo recurso puede servirse a través de distintos dominios.

Las páginas que componen un sitio web pueden ser estáticas o dinámicas (⑥ y ⑦ en la Figura 3). Una **página web dinámica** {*dynamic web page*} es aquella que puede variar parcial o totalmente entre un acceso y otro gracias a la ejecución de programas de computadora. Una **página web estática** {*static web page*} es aquella que nunca puede variar entre un acceso y otro, dado que carece de alguna forma de programación. Por analogía, un **sitio web estático** {*static website*} es aquel cuyas páginas son todas estáticas, mientras que un **sitio web dinámico** {*dynamic website*} es aquel que tiene al menos una página web dinámica. Realmente estos términos son extremos de un continuo, donde el sitio es más dinámico entre más páginas web dinámicas lo conformen.

De acuerdo a la relación ⑧ de la Figura 3, una *página web dinámica* que contiene programas que permiten a un usuario interaccionar de forma análoga a una aplicación de escritorio con interfaz gráfica de usuario {GUI, *graphical user interface*} se llama **aplicación web** {*web application*}. Al ser una página web más (relaciones ⑧ y ⑥ en la Figura 3), una aplicación web no está aislada del sitio web, sino que está acompañada de otras páginas estáticas, dinámicas, o incluso de otras aplicaciones web, asociadas a través de enlaces. Esta versatilidad permite distribuciones interesantes como las siguientes:

1. Un sitio web donde una página, normalmente la página principal, es una aplicación web, y las demás páginas son estáticas o menos dinámicas. Por ejemplo, aplicaciones como Google Maps, Windy, y draw.io, constan de una aplicación como su página principal, y páginas secundarias con ayuda para el usuario, documentación para desarrolladores, o *blogs* de noticias, entre otros.
2. Un sitio web donde todas sus páginas son generadas automáticamente por el mismo programa, normalmente a través de administradores de contenido {CMS, *content management system*}. Son ejemplos la Wikipedia y la NASA.
3. Un sitio web donde las páginas son aplicaciones web distintas, pero intercomunicadas de manera uniforme. Ejemplos podrían ser sitios de videojuegos como y8 y Steam.

Ejercicio 2 [app_types, 5 pts]

Cree la carpeta `intro/app_types` para este ejercicio, y dentro un documento de Markdown (`app_types.md`) o AsciiDoc (`app_types.adoc`) para responderlo. Suponga que como analista usted tiene clientes con necesidades de sistemas distintos:

Nombre	Descripción
Leaf	Los biólogos necesitan tomar fotografías de hojas de plantas en el campo y que el software encuentre cuál es la planta más similar en la base de datos.
Vet	La veterinaria quiere que los clientes puedan registrar las citas y consultar el expediente de su mascota.
BrokenLink	Un departamento de TI quiere que un buscador de enlaces rotos en el sitio web se pueda ejecutar todas las madrugadas o cada vez que el equipo hace un *commit* en el repositorio de control de versiones del sitio.
Enroll	Una nueva aplicación necesita acceder a la base de datos de matrícula en la Oficina de Registro en una universidad, pero ésta última es un sistema heredado {*legacy system*} y no se quiere modificar.
Vinyl	Un coleccionista quiere una aplicación que ayude a extraer el audio de sus discos de vinilo, escanee las portadas de los discos, mantenga los archivos juntos, agregue las letras de las canciones, y los muestre mientras el audio se reproduce.

En el siguiente cuadro indique para cada tipo de aplicación cuál de las necesidades anteriores satisface mejor en la columna "Ejemplo". Finalmente, en la columna "Apropiado cuando..." escriba al menos un contexto (situación o restricción) donde usted considera que este tipo de aplicaciones es el más adecuado a desarrollar. Un ejemplo de una situación es "[Apropiado cuando...] se ingresan grandes cantidades de datos".

Tipo	Ejemplo	Apropiado cuando...
Aplicación en línea de comandos		
Aplicación para dispositivo móvil		
Aplicación gráfica de usuario (GUI) en escritorio		
Aplicación web		
Servicio web		

1.3. El equipo web

Entre más complejo y extenso es un sitio web, mayor será la necesidad de aporte multidisciplinario para construirlo y mantenerlo. Como desarrollador(a) web, es importante que usted conozca sobre otras disciplinas que usualmente intervienen en un *equipo web*, sea que tenga que colaborar con profesionales de dichas disciplinas en equipos grandes, o que deba asumir otros roles en equipos de

menor tamaño. En este material se llamará **equipo web** {*web team*} al conjunto de personas que trabajan colaborativamente para crear o mantener un sitio web y sus aplicaciones. A continuación sólo se cita algunas de las profesiones que intervienen en un equipo web, escogidas entre una creciente cantidad de especializaciones, y ambientadas a un contexto hipotético. Aprender más sobre las disciplinas de estos roles le permitirá un mejor desempeño e interacción en el equipo web.

Supóngase que usted decide emprender en el desarrollo web. Su primer potencial *cliente* necesita un sitio web de mediana complejidad. Usted acaba de terminar la primera reunión y su potencial cliente quiere que le estime en tres días hábiles cuánto tiempo usted tomará en desarrollar el sitio y cuánto presupuesto necesitará, con el fin de determinar si firma el contrato con su recién creada empresa o con alguna de la competencia. Si usted aún no dispone de estas habilidades, necesitará la experiencia de un(a) **administrador(a) de proyectos** {*project manager*} en estimar presupuestos y cronogramas.

Naturalmente antes de poder ofrecer sus servicios debe primero constituir una empresa. Usted puede estudiar la normativa de su país para cumplir con los requisitos iniciales y permanentes, como declaraciones de renta y pago de derechos, o bien adquirir asesoría legal. También necesita estudiar administración del recurso humano para reclutar empleados, pagarles un salario acorde a sus méritos, realizar los aportes y deducciones de ley, y mantener un buen clima organizacional... o bien delegar en un administrador de recursos humanos. Todo esto requiere dinero, y usted necesita conseguir una fuente de apoyo inicial, y hacer una sana administración monetaria para disponer de instalaciones aptas donde poder trabajar, pagar a sus colaboradores, y evitar una bancarrota... o contratar a un administrador financiero. Estos y otros procedimientos pueden simplificarse si cuenta con apoyo de una incubadora de empresas, o al menos de un grupo de amistades formadas en estas profesiones que decidan emprender en conjunto.

Ejercicio 3 [entrep_organigram, 5 pts]

Diagrame la estructura organizacional de su empresa, la que hipotéticamente emprende en esta sección. Suponga que todos los roles mencionados intervienen en su empresa, como los administradores citados. Dibuje los roles como actores de UML {*unified modeling language*} y sus relaciones. Agrupe los roles por departamentos y resáltelos, por ejemplo, usando rectángulos. Sugerencia: use una aplicación web de diagramación, como draw.io o Lucidchart.

Volviendo a su equipo web. Supóngase que su empresa logró el contrato con el cliente que requiere el sitio web de mediana complejidad. Ahora debe cumplir con el compromiso adquirido. El cliente tiene ya un sitio web y sus empleados lo consideran "un desorden", la información está desactualizada, y hay mucha no publicada ni documentada. El cliente espera que usted pueda recabar esta información y estructurarla en el renovado sitio web para que sea fácil de encontrar y comprender para los visitantes. Usted puede aprender y desarrollar más sobre estas habilidades de bibliotecología o agregar a su equipo web un(a) **arquitecto(a) de la información** {*information architect*}. Este profesional realizará una investigación documental del sitio web existente y del nuevo material, y sugerirá una estructura para el nuevo sitio web, que sea natural de comprender para los visitantes.

Su arquitecto(a) de la información realizará las actividades en la parte izquierda de la Figura 2, y le entregará un mapa del sitio web más bocetos de los tipos de páginas que lo conforman. Estos productos carecen de una apariencia atractiva y no puede entregarlos aún al cliente. Usted necesita habilidades de artes para lograr que el sitio web se comunique visualmente con los visitantes, que el

mensaje les resulte claro, eficaz, eficiente, y atractivo. Si aún no ha formado estas habilidades, enriquezca su equipo web con una(a) **diseñador(a) gráfico(a)** {*graphic designer*}, quien elaborará la identidad del sitio web, las fotografías e ilustraciones, y las hojas de estilos que dan composición y apariencia a los bocetos del(a) arquitecto(a) de la información.

El sitio web aún no está completo, y por tanto no se puede entregar al cliente. Hasta el momento se tiene un listado de páginas web que deben conformar el sitio, y la apariencia de los tipos de páginas. Falta el contenido y los programas de computadora que permiten capturarlo y presentarlo de acuerdo a la arquitectura de la información e identidad visual definidas previamente. Aquí es donde entra su trabajo de **desarrollador(a) web** {*web developer*}.

Su arquitecto(a) de la información y su diseñador(a) gráfico(a) no le entregarán la arquitectura y apariencia del sitio y se irán a casa. Ellos(as) le estarán esperando porque como *equipo web* deberán recorrer el ciclo a la derecha en la Figura 2 una y otra vez hasta construir productos funcionales y usables que se irán entregando al cliente. En cada iteración se seleccionará un subconjunto del mapa del sitio a construir. El(la) arquitecto(a) de la información estará ahí para proveer contenidos y detalles necesarios que no se pueden presentar en un mapa del sitio o sus bocetos, y para asegurar que la información no pierda su facilidad de ser encontrada y comprendida. El o la diseñador(a) gráfico(a) estará ahí para elaborar los recursos visuales del nuevo contenido y evitar que se distorsione la identidad visual. Durante las discusiones del equipo web en cada iteración, es normal que la arquitectura de la información y la apariencia del sitio sean reajustadas para asegurar la calidad y la buena experiencia del usuario.

Su equipo web aborda la primera iteración y se ha definido como prioritario iniciar con la aplicación web que ocupará la página principal del sitio. Usted tiene dos meses para entregar la funcionalidad más elemental al cliente, sin errores, y que genere una buena experiencia de usuario. Son varias habilidades técnicas combinadas que requiere formar o delegar para lograr su meta en dos meses, como las siguientes:

1. **Tecnólogo de la información** {*information technologyst*} para instalar los servidores, brindarles conectividad, mantenimiento, seguridad, alimentación eléctrica ininterrumpida, y tolerancia a fallos.
2. **Experto en bases de datos** {*database expert*} para comprender las fuentes de datos existentes, diseñar las bases de datos para el nuevo sitio (relacionales o no), optimizar las consultas, asegurar la integridad de los datos, y la concurrencia, entre otros.
3. **Desarrollador del lado del servidor** {*server-side developer*}, que pueda conectar el sitio web con la infraestructura tecnológica de las organizaciones, en tecnologías como Node.js, Java, ASP.NET, Ruby on Rails, o Dyango.
4. **Desarrollador del lado del cliente** {*client-side developer*}, que implemente las aplicaciones que corren en los navegadores web, con tecnologías como HTML, CSS, JavaScript, y WebAssembly.
5. **Experto en interacción humano-computador** {HCI, *human computer interaction*}, que diseñe aplicaciones usables, y realice pruebas hasta que generen una buena experiencia de usuario.
6. **Optimizador de consultas** {SEO, *search engine opmitization*}, que estructure el sitio web para mejorar su visibilidad en los resultados de buscadores web como Google o Bing.
7. **Asegurador de la calidad** {QA, *quality assurance expert*}, que supervise el proceso de desarrollo con el fin de prevenir errores que introduzcan defectos o fallos en las aplicaciones.
8. **Experto en seguridad** {*security expert*}, que evite vulnerabilidades en las aplicaciones web que

pongan en peligro los datos de la organización, la privacidad e integridad de las personas.

9. **Analista de software** {*software analyst*}, con grandes habilidades de comunicación para comprender las necesidades de los usuarios y traducirlas a requerimientos de software.
10. **Soportista técnico** {*Technical support*}, que capacite y ayude a los usuarios a lograr sus metas cuando las aplicaciones web están en producción.

Entre más complejo sea el sitio web, o más proyectos tenga su empresa, o ambos, mayor será la probabilidad de delegar en profesionales como los anteriores. Para proyectos de menor escala, es común que un informático abarque varias categorías. En inglés se suele llamar ***backend developer*** a quien abarca las categorías 1 a 3, y ***front-end developer*** a quien se especialice en las categorías 4 a 5. Si se especializa en ambas, se le suele llamar ***full-stack developer*** y es el rol que este material pretende ayudarle a adquirir, con algunas influencias del diseño gráfico y de la arquitectura de la información.

Ejercicio 4 [entrep_budget, 5 pts]

En una hoja de cálculo, preferiblemente de formato abierto {ODF, *Open Document Format*}, elabore un presupuesto hipotético de su empresa. Para cada profesional que participa, estime la cantidad de personas y un salario bruto mensual que recibiría. Inclúyase usted y el salario que desea percibir. Calcule aproximadamente el incremento de los salarios aplicando las aportes de ley del patrono. Agregue otros gastos que podría tener su empresa, como personal de limpieza, servicios básicos, equipo de cómputo, y alquiler de bienes inmuebles.

Calcule cuánto sería el monto mínimo que tendría que cobrar a su cliente del sitio web para poder sufragar los gastos de su empresa. Agregue el monto de las utilidades que espera su empresa percibir.

Si su empresa logra varios contratos de nuevos desarrollos, algunos gastos comunes se distribuyen pero incrementan la cantidad de profesionales en algunos roles, calcule cuánto sería el monto promedio por proyecto. Aunque sus cálculos son ficticios, trate de averiguar aproximaciones realistas para obtener una mayor percepción de esta dinámica que ocurre en la sociedad laboral.

1.4. Análisis del sitio web

Su objetivo en la fase de análisis (a la izquierda en la Figura 2) es comprender el problema a resolver, y por tanto determinar las necesidades y deseos del cliente que pueden ser satisfechos con un sitio web y sus aplicaciones. **Análisis** {*analysis*} es la descomposición de un todo en partes más fáciles de comprender. Durante la fase de análisis el equipo web indaga sobre el problema, lo descompone en sub-problemas, y documenta los hallazgos de tal forma que al leer los documentos es claro el problema atendido y qué se espera que el sitio web haga para ayudar a solucionarlo.

Rosenfeld, Morville, y Arango recomiendan un modelo de descomposición del problema en tres entidades presentes en los sitios web: el contexto, el contenido, y los usuarios (Rosenfeld et al., 2015). Durante la fase de análisis usted realizará entrelazadamente investigación social e investigación documental con el fin de comprender estas tres entidades del sitio web, lo cual también se hará en los apartados de esta sección. En la investigación social las fuentes de información son personas, como el cliente y los usuarios del sitio web. En la investigación documental las fuentes de información son

documentos o bases de datos existentes en la organización, su sitio web actual en caso de existir, o los sitios web de la competencia.

1.4.1. Contexto

Los sitios web se crean en el **contexto** {*context*} de una organización, o al menos de una persona, que tiene un objetivo y recursos para crearlo. Durante la fase de análisis es probable que usted celebre una cantidad copiosa de reuniones con el cliente y los usuarios. En ellas realizará preguntas para comprender el contexto del problema a resolver. Conviene preparar una lista inicial de preguntas, como los ejemplos de la Tabla 1 (Rosenfeld et al., 2015).

Tabla 1. Potenciales preguntas y documentos para comprender el contexto del sitio web

Ejemplos de preguntas a realizar	Ejemplos de documentos
¿Cuáles son los objetivos de la organización con este sitio web? ¿Por qué vendría gente a visitarlo? ¿Por qué vendrían de nuevo? ¿Qué tipos de tareas realizarán los visitantes en este sitio web? ¿Quién creará el contenido y cómo? ¿Quién le da mantenimiento? ¿Cuál es la infraestructura técnica donde correrá el sitio web? Si existe, ¿qué funciona bien en el sitio web actual y qué no? ¿Cuáles obstáculos usted anticipa en el desarrollo del sitio web?	Misión, visión, y objetivos. Estructura organizacional. Documentos de diseño del sitio actual. Minutas de reuniones del sitio actual. Sitios web de la competencia.

Si la organización es muy grande, indague primero quién es la persona idónea para responder cada pregunta que usted tenga. Trate de mantener reuniones informales con pocas personas (5 o menos). Anote las respuestas o grabe la sesión con consentimiento previo. La reunión informal se recomienda con el fin de establecer confianza {*rapport*} con las personas de la organización, y así obtener respuestas sinceras, sobre todo a las preguntas difíciles pero necesarias para el nuevo sitio web. Haga las preguntas que necesite sobre la marcha y puede permitir a los interlocutores vagar un poco, lo cual le permitirá conocer más sobre el contexto (Rosenfeld et al., 2015).

Durante las reuniones descubrirá documentos afines como los listados en la segunda columna de la Tabla 1. Si la organización ya cuenta con un sitio web, es valioso averiguar en qué medida éste ayuda o no a la organización a alcanzar su misión, visión, y objetivos. El Ejemplo 1 muestra el análisis de contexto del proyecto de ejemplo Bingo, el cual es un trozo de su archivo `README.md`.

Ejemplo 1. Avance 1 de Bingo: análisis de contexto

Contexto del proyecto Bingo

Bingo es un popular juego social de azar. Conceptualmente cada jugador supervisa uno o más cartones que adquiere, regularmente tras pagar algún valor monetario por ellos. Cada cartón tiene una matriz de símbolos, usualmente 25 celdas distribuidas en cinco filas por cinco columnas. Los símbolos son pre-asignados en cada cartón, escogidos al azar entre un alfabeto de símbolos, típicamente los números de 1 a 75. Un moderador saca al azar, uno tras otro, los símbolos del alfabeto de un bombo (tómbola). El moderador divulga cada símbolo extraído con tiempo suficiente para que los participantes puedan buscarlo en sus cartones. Los jugadores marcan las coincidencias en sus cartones con algún mecanismo. El primer jugador que forme un patrón con símbolos marcados en su cartón (por ejemplo, cuatro esquinas o una línea vertical), anuncia "bingo" con el fin de alertar al moderador y demás jugadores. El moderador comprueba que el patrón sea válido y en tal caso el jugador obtiene un premio predefinido para el patrón. El premio puede dividirse o rifarse si dos o más jugadores ganan el mismo patrón simultáneamente.

Los símbolos en el bombo y en los cartones usualmente son idénticos. Si son distintos, deben proveer alguna clave que ayude a corresponderlos de forma única, lo que permite variaciones creativas del juego (por ejemplo, los símbolos en el bombo son operaciones matemáticas y en el cartón son resultados). Una partida termina cuando se reclamen los premios acordados para la misma, usualmente tras el primer jugador que marque todas las celdas de un cartón (llamado "cartón lleno"). Una sesión de bingo consta de varias partidas. Entre partidas los jugadores pueden cambiar sus cartones buscando "mejor suerte".

El bingo se juega desde contextos informales a profesionales. En los últimos es frecuente que el sorteo lo realicen computadoras, las cuales determinan automáticamente los cartones ganadores, los reportan como una lista de identificadores tras el sorteo, y los participantes sólo necesitan buscar los identificadores de sus cartones en la lista. Sin embargo, esta forma estructurada de jugar descarta la riqueza social del juego bingo.

En ambientes más informales se espera que el moderador acompañe los símbolos que extrae de la tómbola con algún chiste o frase con rima (por ejemplo, "el loco... 35", "las patas de <nombre de alguien flaco(a)>, el 11"). En algunos sorteos el moderador realiza juegos intermedios entre algunas partidas. Por ejemplo, para premiar al cartón "más malo", el moderador extrae números del bombo y aquellos cartones que tienen alguno de ellos dejan de participar, hasta quedar el "desafortunado" ganador.

En circunstancias aún más informales, un grupo de amigos(as) pueden querer jugar bingo pero no disponen de un bombo completo, cartones, suficientes fichas para marcar, o de un(a) animador experimentado. La tecnología, en especial por la difusión de dispositivos móviles, puede ayudar a que estas personas alcancen su objetivo de disfrutar los beneficios sociales del juego. Sin embargo, la mayoría de las aplicaciones existentes se han orientado al los ideales formales del juego o a los ideales lucrativos de los casinos.

Hay dos objetivos del proyecto. El primer objetivo es del sitio web, que consiste en ayudar a las personas a disfrutar los beneficios sociales del juego bingo. El segundo objetivo es de la organización que lo crea, el autor de este material, que consiste en aprovechar el conocimiento popular del bingo para ilustrar las nociones sobre creación de aplicaciones web.

Los visitantes de este sitio vendrán por su interés de disfrutar el juego con amigos o familiares, por sus servicios de amenización, o con fines didácticos. No es necesario registrarse para poder jugar, y no

tiene costo. Para poder jugar es necesario que un visitante cree una sesión de bingo en el sitio web. Este visitante automáticamente adquirirá el rol de moderador, y el sistema le creará una sesión identificada mediante un código.

Los demás jugadores accederán al sitio web y se unirán a la sesión mediante el código que les proveerá el moderador. El moderador podrá ver la lista de jugadores conectados y eligir en qué momento se inicia la primera partida. Los jugadores podrán solicitar cartones de bingo cuyos símbolos son generados al azar. La cantidad máxima de cartones por jugador la estipula el moderador en la configuración de la partida. Si nuevos jugadores se unen a una sesión mientras una partida está en progreso, esperarán a que inicie la próxima partida.

El moderador podrá escoger si desea que el sistema locute ("cante") los símbolos o si sólo debe presentarlos en pantalla. Los jugadores podrán marcar o desmarcar los símbolos en sus cartones al accionarlos. En caso de que formen un patrón, podrán reclamar al presionar un botón de "Bingo" para alertar a los demás jugadores (útil para sesiones muy concurridas). El sistema no jugará cartones, ni dará pistas sobre celdas que los jugadores inadvertidamente omitieron o marcaron incorrectamente, ya que la atención combinada con la suerte son requisitos para ganar en la versión social del bingo. El sistema mostrará el último símbolo favorecido, y si el moderador lo permite, el estado de todos los símbolos favorecidos en la partida.

Al crear una partida, el moderador podrá escoger los patrones que serán premiados, y los obsequios disponibles. Además podrá escoger el *tipo de bingo* entre una lista de predefinidos, creados por desarrolladores u otros visitantes, adaptarlos, o crear un tipo de bingo propio. El moderador decide si quiere compartir sus tipos de bingo o mantenerlos para uso privado. En caso de compartirlos, el sistema podrá llevar estadísticas de popularidad, como el número de partidas jugadas.

Al editar un tipo de bingo, el sistema le permitirá ingresar el alfabeto del bombo y el alfabeto de los cartones, o indicar que son iguales. Los símbolos de los alfabetos se pueden ingresar directamente en el sitio web si son numéricos o textuales, o pueden cargarse desde recursos como imágenes, sonidos, vídeos, sea al proveer un enlace o arrastrar un archivo. Para cada símbolo del bombo, el moderador podrá agregar recursos complementarios como textos con sugerencias para "cantar" el símbolo, o sonidos grabados. El moderador podrá indicar si los símbolos y sus complementos tienen dependencias culturales, como regionalismos.

Además de textos, sonidos, imágenes, o vídeos, los símbolos podrían ser programas de computadora que corren cuando el símbolo se asigna a una celda de un cartón o cuando el símbolo es favorecido (extraído del bombo). Estos programas son útiles para crear bingos dinámicos, por ejemplo, un *bingo matemático* donde las celdas del cartón podrían tener operaciones aritméticas generadas al azar. Sin embargo, este tipo de bingo estará limitado a usuarios registrados de confianza por razones de seguridad.

Avance de proyecto 2 [prj_context, 10 pts]

Realice el análisis del contexto de su proyecto. En el archivo `README.md` del repositorio de control de versiones para su proyecto cree una sección "Contexto" (o "*Context*" si documenta en inglés). Responda en esta sección a las preguntas que apliquen de la Tabla 1. Recabe los documentos pertinentes, estúdielos, y agregue los hallazgos a la sección de contexto en su archivo `README.md`.

1.4.2. Contenido

Un sitio web es un mecanismo tecnológico que permite a las personas comunicar información entre sí. Esta información es codificada en medios digitales, y al resultado se le llama **contenido** {*content*}. En la fase de análisis de contenido, su objetivo es determinar qué información se intercambiará entre las personas a través del sitio web, y las cualidades que deberá tener.

El análisis de contenido revisará los trozos de información existentes en la organización, con el fin de identificar los tipos de contenidos que tienen sentido formen parte del sitio web (Rosenfeld et al., 2015), como páginas estáticas o aplicaciones. Para cada contenido es apremiante cuestionarse sus cualidades, que además le permitirá una mayor comprensión del sitio web a elaborar. La Tabla 2 muestra ejemplos de algunas cualidades, y preguntas a realizar sobre el origen de los contenidos, y su destino en el sitio web.

Tabla 2. Ejemplos de cualidades del contenido web

Cualidad	Origen del contenido	Destino del contenido
Identificación	¿Con qué nombre conocen a este contenido?	¿Es este nombre natural para los visitantes del sitio web?
Propósito	¿Para qué es este contenido?	¿Qué beneficio provee al visitante consultar o proveer este contenido?
Autoría	¿De dónde proviene este contenido? ¿Es una fuente externa a la organización? Si es interna ¿qué usuarios lo crean?	¿Cuáles usuarios del sitio web pueden leer o editar este contenido?
Medio	¿En qué medio se encuentra o produce este contenido: texto, texto con formato, texto estructurado, correo electrónico, hoja de cálculo, sonido, imagen, vídeo? ¿En qué medio de almacenamiento proviene: archivos, bases de datos, torrente de bytes {*streaming*}?	¿Los receptores lo consumirán en el mismo medio de origen o es necesario hacer una conversión?
Dinamismo	¿Este contenido es estático o dinámico? Si es dinámico ¿qué aplicaciones lo producen?	¿Se pueden reutilizar o ajustar las aplicaciones para el sitio web? ¿Se deben crear aplicaciones nuevas?
Metadatos	¿Este contenido viene etiquetado o descrito por la fuente? ¿Alguien o algún programa en la organización lo hace? ¿Se usa un vocabulario controlado o es abierto?	¿Cómo describiría este contenido a alguien que no lo conoce? ¿Debe el sitio web facilitar el etiquetado del contenido? ¿Qué palabras clave tendría alguien que escribir en un buscador para encontrar este contenido?
Volumen	¿Cuántas unidades de este contenido existen en la organización? ¿Qué se hace con el contenido obsoleto?	¿Cuántas unidades de este contenido se producirán aproximadamente en el sitio web por año? Si nadie consulta este contenido, ¿qué se debe hacer con él?

Cualidad	Origen del contenido	Destino del contenido
Relaciones	¿Tiene este contenido referencias a otros del mismo o diferente tipo? ¿Son estas relaciones jerárquicas? ¿Depende de otros contenidos para ser comprendido?	¿Qué distingue a este tipo de contenido de los ya identificados? ¿Se puede fusionar con otro?

El análisis de contenido de la organización es un trabajo arduo pero no infinito. Dependiendo de los recursos e intereses de la organización, puede ser un muestreo informal, una auditoría detallada, o una mezcla de ambas (Rosenfeld et al., 2015). Es probable que al inicio del análisis de contenidos sienta que hay muchos tipos de contenidos distintos, pero conforme avance, los contenidos se irán clasificando en los tipos que ya haya identificado. Para cada nuevo tipo de contenido documente sus cualidades (como las de la Tabla 2). Mientras lo hace, notará relaciones entre los tipos de contenidos, o si es un tipo ya existente. Usted puede detener el análisis de contenidos cuando ya no encuentre tipos de contenidos nuevos, o los contenidos no le provean conocimiento nuevo sobre los tipos existentes (Rosenfeld et al., 2015).

Si la organización ya cuenta con un sitio web que quiere mejorarse o desecharse, es una oportunidad para reutilizar contenido y obtener requerimientos. Con ayuda de una herramienta automatizada puede crear un **mapa de contenido** {*content map*}, que es una representación visual del contenido existente y sus relaciones (Rosenfeld et al., 2015). Estas representaciones pueden discutirse con miembros de la organización para determinar qué contenido debería o no mantenerse o mejorarse en el nuevo sitio web. El Ejemplo 2 muestra un trozo del archivo `README.md` con los tipos de contenidos identificados en el proyecto Bingo.

Cuando existen sitios web similares al que se quiere construir, sea de la organización o de la competencia, conviene estudiarlos y evaluarlos. Su estudio permite obtener una lista de características que desean mantenerse o mejorarse en el nuevo sitio web. La evaluación permite obtener medidas cuantitativas o cualitativas que forman una línea base con la que se puede medir y comparar nuevo sitio web (Rosenfeld et al., 2015). Ejemplos de estas medidas son la cantidad de tiempo que tarda un visitante en encontrar un trozo de información relevante, o la eficacia con que logra resolver alguna tarea con el sitio web existente.

Una forma de estudiar un sitio web existente es a través de una **evaluación heurística** {*heuristic evaluation*} (Rosenfeld et al., 2015). Esto es, al menos un(a) experto(a), idealmente ajeno(a) a la organización, critica la arquitectura de la información, el diseño gráfico, la experiencia de usuario, o algún otro aspecto del sitio web actual siguiendo un conjunto de principios o prácticas recomendadas llamadas heurísticas. Si se cuenta con varios expertos(as), éstos deliberarán en busca de un consenso. Los expertos harán visible un conjunto de supuestos que el sitio web actual cumple o no, y sugerencias de cómo mejorarlo, las cuales servirán como requerimientos para el nuevo sitio web.

Ejemplo 2. Avance 2 de Bingo: análisis de contenido

Contenido

Dado que es con fines ilustrativos, la organización en el proyecto Bingo es el autor de este material. El autor dispone simplemente de juegos de mesa compuestos de un bombo, 75 bolitas numeradas, un tablero para colocar los símbolos extraídos, cartones impresos con 24 números escogidos al azar, y algunas fichas para marcar en los cartones. El autor tiene experiencia con la creación de bingos textuales y gráficos, y dispone de listas de símbolos para algunos eventos, como tes de canastilla.

El autor no es dueño de un sitio web para el juego de Bingo. Los sitios listados en la Tabla 3 pueden considerarse como competencia. Los sitios que no coinciden con los ideales de este proyecto se excluyen de la lista, que son la mayoría de aplicaciones web de bingo, caracterizados por un ambiente de casino donde los jugadores pagan para adquirir poderes que les incrementen sus probabilidades de ganar.

Tabla 3. Sitios web afines al proyecto Bingo

Sitio web	Descripción
BingoBaker	Genera cartones cuadrados de 3x3, 4x4, o 5x5. Permite exportarlos o imprimirlos. Permite marcar símbolos en un cartón en línea sin un estado de juego. Los símbolos pueden ser palabras o imágenes. Permite reusar alfabetos creados por otros. Permite escoger letras como encabezados de las columnas.
My Free Bingo Cards	Permite escoger el tema (plantillas) de cartones cuadrados de 3x3, 4x4, o 5x5. Sólo permite símbolos textuales. Permite reusar alfabetos creados por otros. Los cartones sólo se pueden imprimir.
Print-Bingo	Permite crear cartones de 5x5. Sólo permite símbolos numéricos o textuales. Permite reusar alfabetos creados por otros. Permite escoger los encabezados de columna. Los cartones sólo se pueden imprimir.
Bingo Card Template	Permite crear cartones de 5x5. Sólo permite símbolos textuales. Permite reusar alfabetos creados por otros. Los cartones sólo se pueden imprimir.
Bingo Card App	Permite crear cartones de 5x5. Sólo permite símbolos textuales. Permite escoger las letras encabezado de columnas y el valor de la celda del centro. Permite reusar alfabetos prestablecidos. Los cartones sólo se pueden imprimir.
Free Bingo Sheet Generator	Crea cartones de dimensiones arbitrarias. Sólo imprime cartones. Sólo permite símbolos textuales. Permite reusar alfabetos creados por administradores. Permite asignar el centro vacío.
An Owomoyela's bingo generator	Crea cartones cuadrados de dimensiones 1x1 a 7x7. Permite reusar alfabetos prestablecidos por el administrador. Permite configurar la celda central. Genera código HTML.
DLTK Cards	Crea cartones cuadrados 3x3, 4x4, y 5x5, con números o imágenes. Permite reusar alfabetos prestablecidos por el administrador. Permite escoger las letras encabezado de columnas. Los cartones sólo se pueden imprimir.

A continuación se muestran resultados del análisis los contenidos en propiedad del autor y en los sitios web existentes de la Tabla 3.

Contenido:	**Sesión {*Session*}**
Propósito	Permitir que varias personas jueguen de acuerdo a sus preferencias sin interferir con otras.
Autoría	Se crea por una persona que adquiere el rol de organizador o moderador de la sesión.
Medio	Una sesión es un concepto abstracto. En el juego tradicional se puede asociar con el afiche del evento, usualmente con un texto como "Gran bingo a beneficio de <organización> a celebrarse el <fecha> en <lugar>..." y otros detalles como el costo del cartón, y los premios principales.
Dinamismo	La sesión es un contenido dinámico a crear por la aplicación web.
Metadatos	Título del evento, organizador, fecha, duración estimada, costo de cartón.
Volumen	No se tienen estadísticas de cuántas sesiones se pueden crear por año. Las sesiones caducadas se pueden eliminar del sistema.
Relaciones	Una sesión está compuesta de cero o más *partidas*.

Contenido:	**Partida {*Round*}**
Propósito	Fragmentar la sesión en unidades que indican a los jugadores cuándo se han ganado ciertos premios, cuando es preciso limpiar los cartones, y cuando disponen de pausas.
Autoría	Se crea por el moderador, el cual define la cantidad de partidas en la sesión, sea de forma previa o dinámicamente, de acuerdo a si los jugadores pretenden continuar con la sesión. A veces se identifica como "bingo" y algún número, por ejemplo, "Limpien sus cartones que vamos con el tercer bingo de la noche..." El moderador anuncia los premios y los patrones con que se ganan antes de iniciar la partida.
Medio	En el juego tradicional se anuncia verbalmente, o quizá con alguna indicación visual si se cuenta con un proyector.
Dinamismo	Se generan de forma dinámica, durante la configuración de la sesión o durante la misma de acuerdo a los intereses de los jugadores.
Metadatos	Número de partida, hora de inicio, tiempo transcurrido.
Volumen	Depende de los intereses del organizador o de los jugadores. Una sesión típicamente tiene menos de una decena de partidas.
Relaciones	Una partida tiene una o más parejas *patrón-premio*. El patrón corresponde a un conjunto de celdas de los cartones. El primer jugador en cubrir el patrón obtiene el premio asociado. Algunos patrones característicos son: una línea horizontal o vertical, cuatro esquinas, una diagonal, o cartón lleno. Una partida tiene un *estado* que indica los símbolos que han sido extraídos del bombo y los patrones-premios que están pendientes de ser reclamados. En algunas sesiones, los símbolos que se han extraído del bombo se colocan sobre un *monitor* visible a los jugadores, que les permite rápidamente saber si un símbolo ha sido llamado o no.

Contenido:	**Cartón {*Card*}**
Propósito	Agrupar una selección aleatoria de símbolos que el jugador vigila durante la partida.
Autoría	En el juego tradicional es creado por el fabricante del juego o por un fabricante de cartones estándar. En los sitios web existentes es creado por el sistema.
Medio	En el juego tradicional es un papel o cartulina impresa, y en línea es generado como una página web o un PDF imprimible.
Dinamismo	En el juego tradicional es estático, un cartón nunca cambia su contenido. En los sitios web existentes los cartones son generados por el sistema a demanda de los usuarios para ser impresos.
Metadatos	Título del cartón, número o código identificador, título de la columnas, dimensiones.
Volumen	En el juego tradicional se requiere aproximadamente dos o tres por jugador. Si se agotan, se pueden regular. En los sitios web se generan a demanda cuando los jugadores lo solicitan.
Relaciones	Contiene columnas y éstas símbolos de cartón. Se relaciona con patrones de juego. Los símbolos del alfabeto normalmente se distribuyen equitativamente entre las columnas, y por tanto, un símbolo aparece siempre en la misma columna en todos los cartones, no necesariamente en la misma fila.

Contenido:	**Alfabeto {*Alphabet*}**
Propósito	Permite a los moderadores definir la lista de símbolos que estarán en el bombo y los cartones.
Autoría	En el juego tradicional lo crea el diseñador del juego. En los sitios web es el usuario quien define los símbolos del alfabeto. Ninguno de los sitios web identificados distingue entre el alfabeto del bombo y el alfabeto de los cartones.
Medio	En el juego tradicional los símbolos se imprimen en las bolas del bombo, en las celdas de los cartones, y en el estado del juego (los símbolos que han sido extraídos del bombo).
Dinamismo	Se crea cuando se define un nuevo tipo de bingo, por ejemplo los números de 1 a 75, o un té de canastilla. Es común reusarlos en sesiones del juego.
Metadatos	Identificador, nombre del alfabeto, tipo (de bombo, de cartón).
Volumen	Uno por tipo de bingo.
Relaciones	Bombo, símbolo, estado de la partida.

Contenido:	**Símbolo {*Symbol*}**
Propósito	Representa los símbolos que estarán en el bombo, los cartones, y en el estado del juego.
Autoría	Creados por el diseñador del juego tradicional, o por los usuarios en sitios web existentes.

Contenido:	**Símbolo {*Symbol*}**
Medio	Números, textos, o gráficos impresos en el juego tradicional y sitios web existentes. Se puede extender a sonidos, y vídeos.
Dinamismo	Se crean cuando se define un nuevo tipo de bingo.
Metadatos	Tipo (de bombo o de cartón)
Volumen	Al menos *T * (R + 1) * C*, donde *T* son los tipos de alfabetos (de bombo, de cartón), *R* el número de filas por cartón, y *C* el número de columnas por cartón.
Relaciones	Emparejamiento (el correspondiente símbolo de alfabeto o de cartón)

Avance de proyecto 3 [prj_content, 10 pts]

Realice el análisis de contenido de su proyecto. En el archivo `README.md` del repositorio de control de versiones para su proyecto cree una sección "Contenido" (o "*Content* si documenta en inglés). Identifique los tipos de contenido que las personas intercambiarán a través de su sitio web. Para cada uno de ellos, documente las cualidades de la Tabla 1.

1.4.3. Usuarios

Si usted construye un sitio web que confunde a sus usuarios, ellos simplemente no lo usarán o acudirán a la competencia (Robertson, 2016). Los daños de un sitio que no satisface las necesidades de sus usuarios van más allá de los altos costos de creación, y encima de re-diseño. El prestigio de la organización está en riesgo, sin listar otros perjuicios a sus clientes y empleados.

Durante el análisis de los usuarios averigüe, potencialmente con su cliente, quiénes usarán su sitio web. Si la tiene una lista de personas físicas, clasifíquelas en roles de usuario. Su objetivo es identificar para cada rol, cuáles son sus necesidades o deseos de información, y cómo piensan usar el sitio web para satisfacerlas. El Ejemplo 3 lista en una tabla los roles de usuarios identificados en el proyecto Bingo.

Si no puede interaccionar directamente con sus usuarios, pero sí comunicarse con ellos, puede pedirles que le respondan una encuesta con preguntas como las listadas en Tabla 4 (adaptadas de (Robertson, 2016)). Si usted puede interaccionar con los usuarios identificados, lo cual es más probable si laboran para una organización, puede observarlos en su contexto para identificar cómo resuelven sus necesidades de información con los recursos que la organización actualmente dispone, en especial si cuenta con un sitio web. Esta oportunidad le permite alcanzar un mayor nivel de comprensión a través de métodos como *personas*.

Tabla 4. Preguntas de ejemplo para entrevistar usuarios del sitio web

Sitio web	Ideas de pregunta
Futuro	¿Qué información requiere para realizar las tareas en <contexto del sitio web>? ¿Cuál de esta información es difícil de obtener con los medios actuales? ¿Qué hace usted cuando no puede encontrar la información que necesita?
Actual	¿Qué es lo más útil del sitio web actual? ¿Que le frustra más del sitio web actual? ¿Qué información que usted necesita no puede encontrar en el sitio web actual? Si crea contenido en el sitio web que otros usan, describa su experiencia ¿Podría decirme tres cambios que harían mejor al sitio web actual?

Personas o personajes (del inglés, *personas*) es un método de análisis usado en interacción humano-computador (HCI) que describe los roles de usuario y sus necesidades representados por al menos un miembro ficticio del rol. Identificar *personas* que usan el sitio web convierte al análisis de usuarios en un proceso menos abstracto, más humanizado y detallado. El equipo web crea al menos un personaje para cada rol de usuario como si fuese una persona física real que va a usar el sitio web. A esta persona se le dan las características como las listadas en la Tabla 5 (Brown, 2011). No se recomienda documentar personas físicas reales, sino crear personajes ficticios a partir de ellas. Recuerde que un personaje representa un rol de usuario, y por tanto, se construye a partir de características comunes de las personas físicas que representa. El Ejemplo 3 muestra *personas* de ejemplo para el proyecto Bingo.

Tabla 5. Cualidades de las *personas*

Cualidad	Descripción
Nombre	Al igual que las personas físicas, los personajes se identifican con un nombre que provee precisión y humanismo en las conversaciones.
Rol y calificativo	Indica el rol de usuario, y un calificativo si dentro del rol pueden distinguirse subtipos como "principante", "impaciente", o "hacker".
Distintivo	Una oración que resume lo que distingue a las personas con este rol.
Conocimiento	El conocimiento y habilidades que la persona tiene o carece que son relevantes en la interacción con el sitio web, así como sus motivaciones y temores. Se pueden usar frases literales expresadas por usuarios reales.
Objetivos	Los objetivos que la persona quiere lograr al usar el sistema, usualmente expresados como listas de acciones en formato verbo-objeto. Por ejemplo "imprimir los cartones de bingo".
Escenarios	Una misma persona puede usar el sitio web en diferentes circunstancias, cada cual provoca una interacción distinta. En un escenario se documenta el evento que causa el escenario, el curso de interacciones entre el usuario y el sitio web, y las expectaciones finales de la persona.

Cualidad	Descripción
Información personal	Puede incluirse una fotografía ficticia que refleje las características de la persona y algunos pocos detalles de su historia de vida que ayuden a hacer al personaje más humano, fácil de recordar y comprender su interacción con el sitio web. No use fotografías de usuarios reales, a menos de que tenga autorización escrita. Conviene emplear imágenes de personas inexistentes generadas por computadora a través de servicios como ThisPersonDoesNotExist que genera una imagen nueva cada vez que se refresca la página, o Generated photos que permite buscar en una base de datos de imágenes generadas.

Una vez que haya completado el análisis de usuarios y contenidos (Sección 1.4.2), conviene crear una matriz de roles de usuario contra contenidos del sitio web. En las celdas se anotan las acciones que los usuarios pueden realizar con dichos contenidos, tales como crear, y consultar, editar o eliminar (los propios o de los demás). El Ejemplo 3 muestra un ejemplo de esta matriz para el proyecto Bingo.

Ejemplo 3. Avance 3 de Bingo: análisis de usuarios

Usuarios

La Tabla 6 lista los roles de usuario del Bingo social, y deseos de información que éstos tienen del sitio web. La Tabla 7 muestra las acciones que los roles de usuario pueden realizar sobre los contenidos del sitio web. Se indican las acciones en las celdas con letras para crear (`C`, *create*), consultar (`R`, *read*), editar (`U`, *update*), y eliminar (`D`, *delete*). Las letras están en minúsculas cuando el usuario sólo puede realizar la acción sobre su propio contenido y en mayúsculas con el contenido de otros usuarios.

Tabla 6. Usuarios del sitio web Bingo

Usuario	Descripción y deseos
Moderador	Es la persona que organiza o ameniza un evento (sesión) de bingo. Crea y configura sesiones, y en ellas las partidas. Para cada partida indica los patrones y sus premios asociados. Inicia o pausa las partidas. Escoge si el sistema extrae los símbolos del bombo o hacerlo manualmente, lo mismo de anunciar ("cantar") los símbolos extraídos. En una versión futura se podría distinguir entre el *organizador* y el *animador*, y tener varios de cada rol para una misma sesión.
Jugador	Es la persona que monitorea con atención cartones en las partidas con el fin de ganar premios. Compra (puede ser sin costo) cartones que el sitio web crea con símbolos escogidos al azar. Marca o desmarca símbolos conforme son anunciados por el moderador o el sistema. Necesita saber cuáles patrones están pendientes de ganar y sus respectivos premios. Advertir al moderador y los demás jugadores cuando ha sido el primero en completar un patrón.
Administrador	Es la persona que da mantenimiento al sistema del juego de Bingo, en especial para resolver disputas o casos de usuarios que violen los términos de uso. El administrador puede realizar las tareas que pueden hacer todos los otros usuarios, y además gestionar la totalidad de contenidos del sitio web.

Tabla 7. Matriz de roles de usuario y tipos de contenidos en el sitio web Bingo

Contenido	Jugador	Moderador	Administrador
Sesión	`-R--`	`cRud`	`cRUD`
Partida	`-R--`	`cRud`	`CRUD`
Cartón	`cr-d`	`cR-d`	`cR-D`
Alfabeto	`-R--`	`CRud`	`CRUD`
Símbolo	`-r--`	`CRud`	`CRUD`

Personas

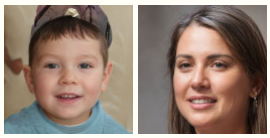
Doña Catalina
Moderadora no tecnóloga
Animadora aficionada al bingo pero no a la tecnología.

A sus 17 años, en un bingo organizado por la iglesia de su barrio, se ausentó el animador por una situación de licor. Para evitar cancelar el evento, Catalina se animó a moderar el juego. A partir de ese momento el bingo se convirtió en uno de sus juegos preferidos. Ahora con 67 años continúa animando eventos sociales y familiares alrededor de este juego. Muchas veces, de improvisto la gente a su alrededor se entera de su presencia o habilidad y quiere oírla amenizar el juego, pero carecen de los implementos.

Doña Catalina porta siempre su teléfono celular para estar en contacto con sus familiares y amigos, incluso en otros países. Su uso se limita a llamadas, mensajes, y redes sociales. Emplear su celular para moderar una sesión de bingo no le parece natural en absoluto, pero podría intentarlo por satisfacer a sus familiares y amigos cuando no hay implementos disponibles. Ella siente mucho temor de descomponer su teléfono, o bloquearse en medio de la sesión sin saber qué hacer con el dispositivo. Ella se sentiría más segura si algún nieto u otra persona la puede asistir en estas situaciones.

El objetivo de doña Catalina es poder animar una sesión de bingo como lo haría con los implementos tradicionales. Ella no ve cómo una reducida pantalla pueda reemplazar la sensación del bombo en sus manos, el sonido de las bolas girando como moléculas, y la emoción que genera la bola rodante que logra escapar del enrejado. Ella desea que el sitio web pudiera replicar la experiencia lo más fiel posible al juego tradicional. Ella no necesita que el sitio web anime, dado que es su especialidad.

Doña Catalina tiene configurado su sistema operativo con fuentes grandes porque su visión lo requiere, pese a ello necesita poder ver el estado de los símbolos extraídos para responder las preguntas de los jugadores y poder verificar los cartones ganadores.



Camilo y Beatriz
Moderador incipiente y Jugadora normal
Madre en espera de su segundo hijo.

Camilo ya no sólo sabe levantar cinco deditos cuando los adultos le preguntan su edad. Está comenzando a comprender el inacabable mundo de los números y está fascinado con el juego de Bingo. Ya no espera que la familia saque el juego de la caja, él toma la iniciativa y no admite que alguien más amenice, aunque a veces necesite ayuda con los números más grandes.

En un par de meses Camilo dejará de ser hijo único. En la agenda del té de bienvenida a su hermanita, poco más de una hora es para el clásico Bingo. Beatriz le ha explicado a Camilo que no es un bingo que se juega con números, sino con elementos alusivos a los bebés. Aún así, Camilo se mantiene determinado a ser el animador. Beatriz quisiera que su hijo realmente pudiera amenizar el bingo de su té de canastilla, pero Camilo aún no sabe leer. Ella piensa que si el juego tuviera dibujos en lugar de palabras, podría reducir la frustración y aburrimiento de Camilo al enfrentarse a los textos y tener que pedir ayuda constantemente.

Aunque no disponen de una impresora en casa, tanto Beatriz como Camilo son aficionados a la tecnología móvil. Beatriz siente que sería una agradable sorpresa para sus amigas poder jugarlo en sus dispositivos. Beatriz quisiera poder escoger las imágenes para asegurarse de que sean reconocibles para Camilo. También quisiera poder agregarle sonidos a las imágenes, de tal forma que si Camilo olvida una de las imágenes, pueda reproducir el sonido a bajo volumen y así ofrecer una animación más independiente.

Beatriz tiene varios premios sorpresa, le gustaría distribuirlos en tres partidas cada cual con cartones de diferentes dimensiones, y cada vez más grandes. Pese a que son dibujos, Beatriz espera que los símbolos aparezcan siempre en la misma columna en los cartones. Ella puede proveer nombres para los símbolos que pueden usarse para ordenarlos. Ellos no quieren proyectar un monitor que las jugadoras puedan vean con el fin de que estén atentas durante el juego.

Avance de proyecto 4 [prj_users, 10 pts]

Realice el análisis de usuarios de su proyecto. En el archivo `README.md` del repositorio de control de versiones para su proyecto cree una sección "Usuarios" (o "*Users* si documenta en inglés). Identifique los tipos de usuarios que interaccionarán con su sitio web. Para cada uno de ellos, documente las necesidades que tienen del sitio web y cree al menos una persona que represente al grupo.

Avance de proyecto 5 [prj_user_contents, 5 pts]

Agregue al final de la sección de usuarios del `README.md` de su proyecto la matriz de roles de usuario (identificados en este apartado) contra los contenidos (identificados en el apartado anterior). Para cada celda, indique las acciones que pueden realizar los usuarios sobre los contenidos.

1.4.4. Prueba del análisis

Durante la fase de *análisis del sitio web* (Figura 1) usted elabora documentos que describen el contexto, los usuarios, y los contenidos que intercambiarán en el sitio web. Es recomendable que usted pruebe estos artefactos conforme los produzca. Sin embargo, es fundamental probarlos para determinar si es propicio avanzar a la siguiente fase del proceso de resolución de problemas.

Para probar los documentos de análisis, distribúyalos entre los clientes, usuarios, o interesados en el sitio web. Solicíteles leerlos y criticar lo que no es claro o correcto. Pregúnteles qué es necesario en el sitio web que haga falta en el documento. Cada deficiencia que se detecte se hará en el momento idóneo, pues su corrección requerirá el mínimo de esfuerzo. De pasar a las fases posteriores, un error será más costoso de corregir y sus efectos no son sólo nocivos para los clientes, sino también para el prestigio del equipo web.

Avance de proyecto 6 [prj_analysis_test, 5 pts]

Pruebe el análisis realizado en su proyecto. Provea una copia de su `README.md` a al menos dos personas interesadas en su sitio web, y pídales que critiquen lo que no es claro, correcto, o completo. No les explique verbalmente sino que el documento debería ser suficiente. Si es ineludible proveer explicaciones para que los interesados puedan avanzar, grábelas e incorpórelas a su documento antes de pasar al siguiente interesado. Haga las correcciones que necesite hasta que el documento sea claro y preciso. Al menos un *commit* debería reflejar los resultados su prueba de análisis y tener el identificador `prj_analysis_test` como parte del mensaje.

1.5. Diseño del sitio web

El propósito de la fase de diseño del sitio web es crear modelos que representan al sitio web y probarlos para determinar si resuelven el problema atendido. Se usan modelos y no la tecnología web final con la que se construye el sitio web por varias razones. Los modelos son simplificaciones o abstracciones de la realidad que ayudan a construir artefactos más rápidamente, y que son más baratos de modificar que la tecnología web final. Además son agnósticas y más estables en el tiempo, lo que permite construir un mismo diseño con diferentes tecnologías. Pueden pensarse los modelos como los planos que un arquitecto elabora, y sin los cuales no se iniciaría la construcción de una estructura como un puente o edificio.

No debe confundirse el diseño del sitio web con el diseño gráfico de sus páginas o el diseño de software de las aplicaciones web. El diseño del sitio web busca representar el sitio completo o al menos sus páginas más importantes. Se presentarán tres tipos de modelos para diseñar un sitio web:

el mapa del sitio, los esquemas de página {*wireframes*}, y una combinación de ellos {*wireflows*}.

1.5.1. Mapa del sitio

El **mapa del sitio web** {*site map*} es una representación visual de la estructura de un sitio web. La **estructura del sitio web** {*website structure*} se refiere a las relaciones que existen entre los contenidos o páginas web que componen al sitio (Brown, 2011). El mapa del sitio es un diagrama que facilita la comprensión del sitio y ayuda enormemente en el proceso de toma de decisiones. Conviene mantenerlo actualizado conforme el sitio cambia su estructura, lo cual es normal a través de su ciclo de vida.

El término "mapa de sitio" también se usa en otros contextos para referirse a una página del sitio web cuya función es listar las páginas más relevantes para los visitantes humanos, o a un archivo en notación XML (`sitemap.xml`) con detalles importantes para motores de búsqueda que no se pueden inferir directamente de la navegación del sitio. En este material el término hará referencia al diagrama que muestra la estructura del sitio web con fines de diseño.

Un mapa de sitio es un grafo que representa con nodos el contenido del sitio y con aristas las relaciones de pertenencia o navegación (hiperenlaces) entre estos contenidos. **Contenido** {*content*} es cualquier trozo de información enlazable, como una página web, un video, o una imagen (Brown, 2011). Para un sitio web documental, los nodos del mapa serán páginas web que contienen documentos. Para una aplicación web los nodos del mapa serán pantallas. Normalmente los sitios web modernos se componen de una combinación de ambos tipos, es decir, de páginas web documentales y páginas web que son pantallas de una aplicación web.

Un *mapa de sitio documental* trata de resaltar jerárquicamente la pertenencia o navegación entre los contenidos, donde un nodo hijo es un contenido que pertenece o forma parte del contenido padre (Brown, 2011). Relajando algunas restricciones, un organigrama cumple con estas características. Quizá esto explique por qué muchos sitios web se han construido reflejando la estructura de su organización. La Figura 4 muestra un extracto del organigrama institucional de la Universidad de Costa Rica.

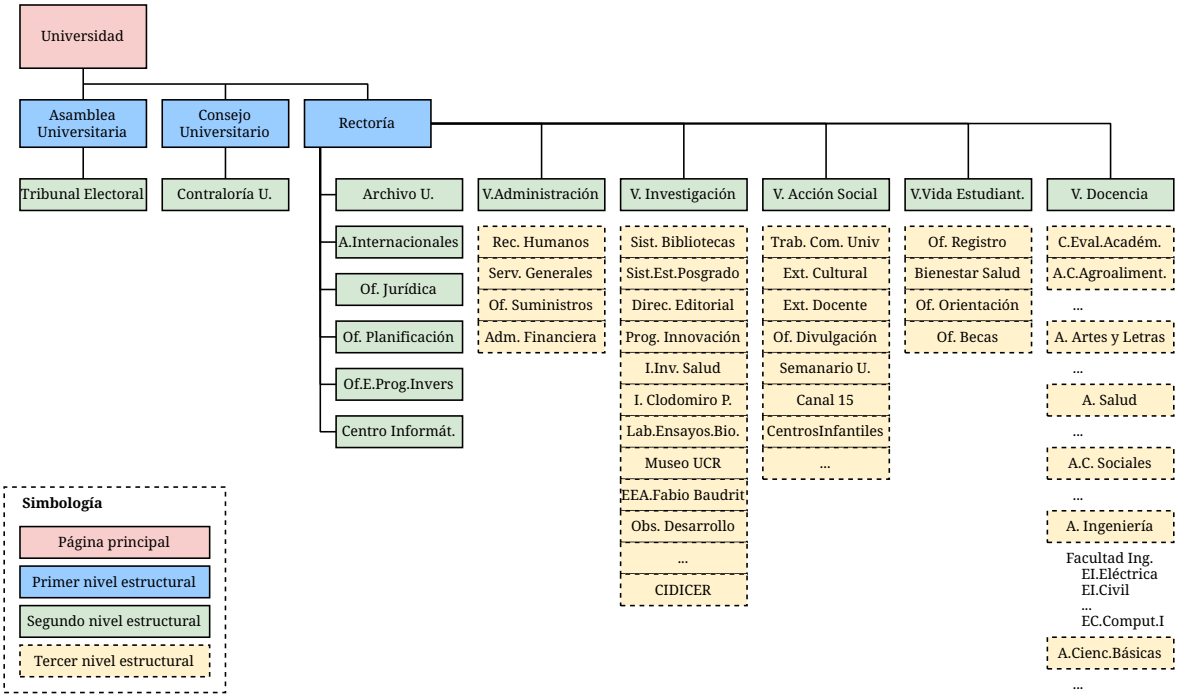


Figura 4. Mapa del sitio para la Universidad de Costa Rica basado en su estructura organizacional

La experiencia indica que la estructura organizacional pocas veces resulta natural para los visitantes (Brown, 2011). Por ejemplo, si se usa el organigrama de la Figura 4 como mapa de sitio, su navegación constaría de tres menúes, los dos primeros con una entrada para el "Tribunal Electoral Universitario" y la "Contraloría Universitaria" respectivamente, y el tercer menú para la "Rectoría" estaría sobrecargado de entradas y sub-menúes. Siguiendo esta navegación, un estudiante de computación tendría que realizar cinco accesos, desde "Rectoría", "Vicerrectoría de Docencia", "Área de ingeniería", "Facultad de Ingeniería", para llegar a la "Escuela de Ciencias de la Computación e Informática".

La Figura 5 es un extracto del mapa de sitio real de la Universidad de Costa Rica en abril de 2020. Aunque la estructura sigue siendo jerárquica, la página inicial se dibujó hacia el centro del diagrama y se enlazó con las secciones principales del sitio web.

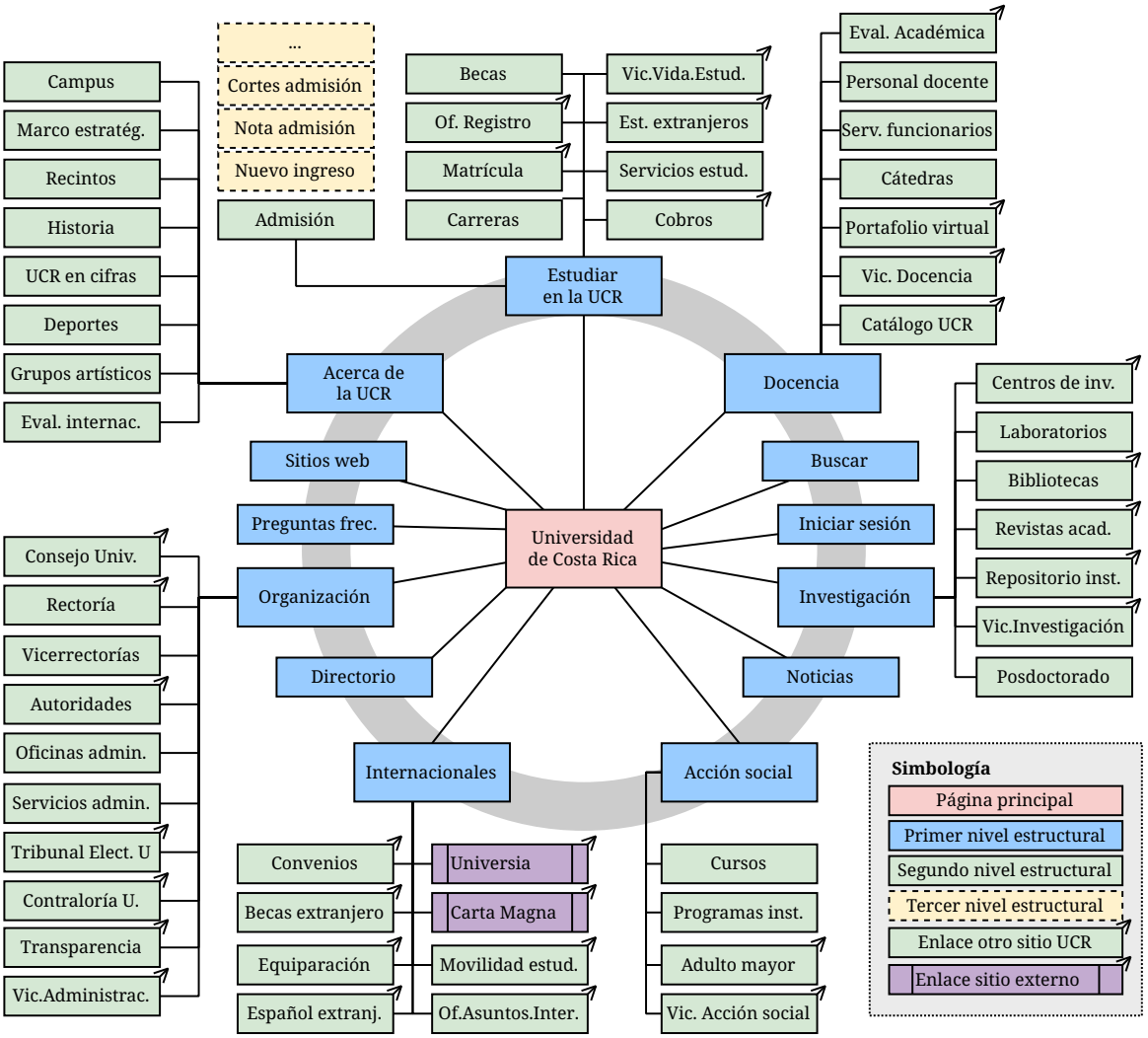


Figura 5. Extracto del mapa del sitio actual de la Universidad de Costa Rica

De las relaciones de pertenencia en un mapa de sitio puede inferirse parcialmente la *navegabilidad* del sitio web, dado que es normal proveer enlaces que permiten al visitante desplazarse desde la página principal hacia las hojas o viceversa. Es parcial porque otros mecanismos como buscadores internos o externos también posibilitan llegar a los contenidos directamente (Brown, 2011). Por ejemplo, en el mapa de la Figura 5, para llegar a la "Escuela de Ciencias de la Computación e Informática" el estudiante podría usar el buscador o el directorio, ambos en el primer nivel estructural, y por tanto, disponibles en todas las páginas del sitio (aunque no en los subdominios).

A la circunferencia alrededor de la página principal se le llama **nivel global de estructura** {*global structural level*}, o *primer nivel de estructura*, porque resalta las secciones globales en que se divide el contenido del sitio. También se le dice **nivel global de navegación** {*global navigation level*} porque las secciones resaltadas en la circunferencia conforman el primer nivel del menú de navegación, el cual

está presente en todas las páginas del sitio web. Los contenidos en la circunferencia de la Figura 5 están presentes en todas las páginas del sitio web de la Universidad de Costa Rica (UCR), sea como enlaces o controles en el encabezado, pie de página, o la barra de menú. La Figura 6 presenta una captura de pantalla del menú compuesto de los nodos cuyos rectángulos tienen más altura en la circunferencia de la Figura 5.

Figura 6. Captura de pantalla de la barra de menú del sitio de la Universidad de Costa Rica

Los nodos externos directamente conectados a los que están ubicados en la circunferencia conforman el segundo nivel de estructura o navegación. Normalmente aparecen como ítems en el menú que les corresponde, como las siete entradas del menú "Docencia" en la Figura 6. Usualmente un mapa de sitio presenta hasta un tercer nivel de estructura o navegación si se dispone de espacio. Si necesita más niveles, considere en subdividir el diagrama. (Brown, 2011)

En un mapa del sitio un contenido se representa con un rectángulo, cuadrado, o círculo, que encierra una etiqueta que lo identifica. En ocasiones es difícil precisar una etiqueta significativa o, por el contrario, se debe escoger entre varias candidatas. En tal caso anótelas como tentativas ya que en la fase de prueba del diseño servirán para discusión. (Brown, 2011)

Es común la presencia de plantillas en los sitios web. Las **plantillas de páginas web** {*web page templates*} son páginas web con campos ideados para que un programa de computadora los llene con contenido extraído de bases de datos u otras fuentes. Cada vez que este "relleno" ocurre, se genera una página web. De esta forma, una plantilla puede producir una cantidad arbitraria de páginas web, lo que se representa en el mapa del sitio como una pila de documentos (Brown, 2011). El Ejemplo 4 ilustra algunas plantillas del proyecto Bingo.

No existe una notación estándar para los mapas de sitio. Aspectos de apariencia de los nodos y sus relaciones, tales como formas, tamaños, bordes, rellenos, tipografías, capturas de pantalla, miniaturas, colores, transparencias, patrones, brillo, y decoraciones, pueden usarse para comunicar otros detalles útiles sobre el sitio web. Por ejemplo, la apariencia de los nodos puede ayudar a distinguir:

- Los contenidos creados o editados por diferentes autores.
- Los contenidos más consumidos por los distintos usuarios (personajes).
- Los tipos de contenidos (ej.: formulario, audio, vídeo).
- Los contenidos nuevos de los ya existes en el sitio web.
- Las páginas dinámicas de las estáticas.
- Las páginas más prioritarias de implementar de las menos visitadas.

- Las páginas internas del sitio de las externas.

En cualquier caso, asegúrese de proveer una leyenda en el diagrama, usar pocos estilos, y mantener la consistencia. Por ejemplo, en la Figura 5 se usaron formas y colores para distinguir las páginas del sitio oficial, de los subdominios, y de sitios web externos. A modo de sugerencia puede crear un "esténcil" {*stencil*} con sus convenciones y reutilizarlo en cada diagrama. Conviene indicar en el título del diagrama si representa a un sitio web existente o a un sitio web futuro. Es común agrupar varios nodos encerrándolos dentro de un rectángulo, porque las relaciones con el rectángulo aplican automáticamente a todos los nodos agrupados. (Brown, 2011)

En lugar de reflejar un organigrama, la información a comunicar en el sitio web ha de estructurarse para satisfacer las necesidades de sus usuarios, y por tanto, que les sea fácil de encontrar y comprender (Rosenfeld et al., 2015). El mapa del sitio resulta de organizar los contenidos recabados en la fase de análisis de contenidos (Sección 1.4.2) para satisfacer las necesidades recabadas en la fase de análisis de usuarios (Sección 1.4.3). El Método sugerido de elaboración de mapas de sitio documentales puede ayudar en esta labor.

Método sugerido de elaboración de mapas de sitio documentales

1. Coloque la página principal como un nodo hacia el centro del diagrama y dibuje una circunferencia alrededor.
2. Para cada contenido recabado en el análisis de contenido, cree un nodo y colóquelo en algún lugar del diagrama fuera de la circunferencia.
3. Trate de agrupar los nodos temáticamente. Este es el paso más subjetivo y conviene realizar pruebas con usuarios finales para encontrar el agrupamiento más común para ellos.
4. Designe un nodo o cree uno nuevo para representar cada grupo. Ubique al nodo representante sobre la circunferencia.
5. Trace las relaciones entre la página principal y el nodo que representa cada grupo.
6. Dentro de cada grupo, identifique si hay subgrupos. Este paso es especialmente necesario si la cantidad de nodos dificulta la lectura del diagrama y conviene subdividirlo.

Si la aplicación web es parte de un sitio documental, puede representarse en el mapa del sitio como un nodo más, y pueda que sea necesario un diagrama aparte para la aplicación web. En un mapa de sitio para una aplicación web, los nodos son pantallas de la aplicación, trozos de estas, u otros contenidos. Las aristas pueden representar la navegación que pueden realizar los usuarios entre las pantallas, o los contenidos que son mostrados en las pantallas. El Ejemplo 4 muestra esta labor para el proyecto Bingo.

Ejemplo 4. Avance 4 de Bingo: mapa del sitio

La Figura 7 muestra el mapa del sitio para el proyecto Bingo. Desde la página principal, y desde cualquier otra a través del menú del sitio, se puede acceder a la pantalla de ayuda (*Help*) y de registro de usuario (*User Account*). A partir de la página principal se pueden seguir dos flujos de eventos comunes, el del moderador y el del jugador.

Desde la página principal el moderador podrá crear una nueva sesión de Bingo, o continuar con alguna que haya creado previamente. La pantalla *Moderate Session* permite al moderador ajustar propiedades de la sesión, como la cantidad máxima de cartones que un jugador puede comprar, si la sesión está disponible al público o es privada. Entre estas propiedades está el alfabeto. Al accionarlo el moderador es presentado con la pantalla *Alphabet*, la cual permite escoger entre alfabetos disponibles o construir el propio.

La pantalla *Moderate Session* permitirá al moderador dividir la sesión en partidas, y para cada una de ellas establecer los premios y los patrones con los que se ganan (por ejemplo, cuatro esquinas). La pantalla de moderación también mostrará en tiempo real la cantidad de jugadores conectados, y permitirá al moderador iniciar con una partida en el momento en que lo considere apropiado.

Al iniciar una partida, el moderador verá la pantalla *Moderate Round*, que le permitirá sacar símbolos del bombo, ver el estado del juego (los símbolos que ya han sido favorecidos), un indicador de si alguien está gritando Bingo!, y los premios que ya han sido reclamados o están pendientes. Una vez terminada una partida el moderador regresa a moderar la sesión, donde puede iniciar la siguiente partida, o terminar la sesión.

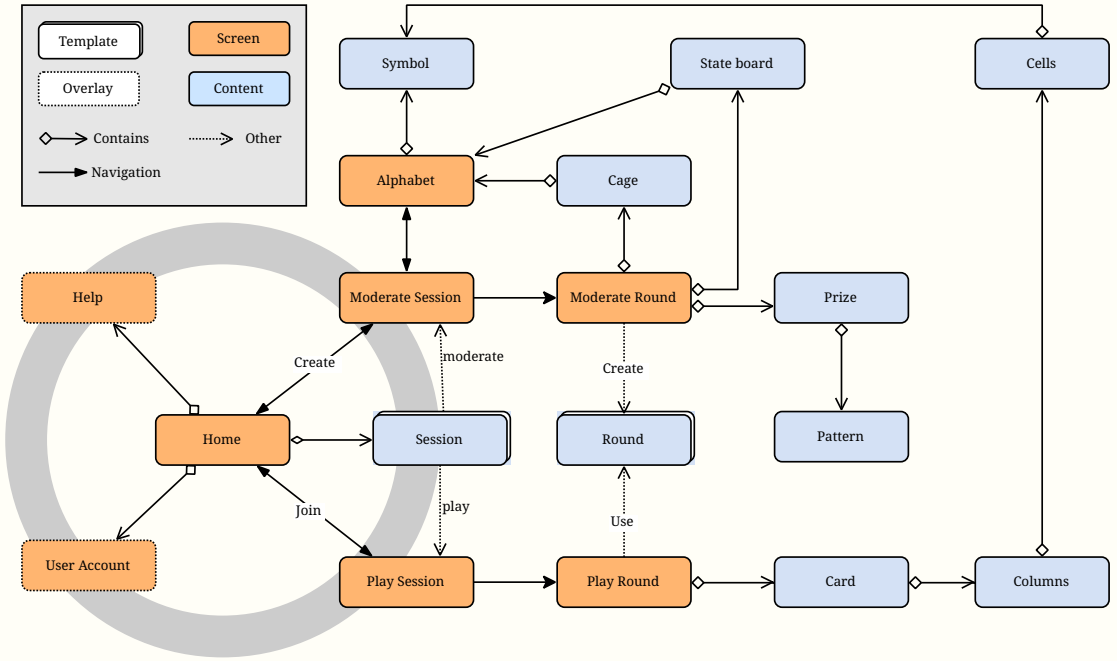


Figura 7. Mapa del sitio web Bingo

Por su parte, un jugador se une desde la página principal a una sesión y es presentado con la pantalla *Play Session*. Esta pantalla le permite consultar propiedades de la sesión, las rondas en que están

divididas, sus premios, y el tiempo que falta para iniciarla. Mientras la partida no haya comenzado, el jugador puede comprar cartones y consultarlos.

Una vez que una partida de bingo inicia, el jugador interacciona con la pantalla *Play Round*. Esta pantalla muestra los premios de la ronda que han sido reclamados o están pendientes, el último símbolo extraído del bombo, y los cartones comprados por el jugador. Al marcar una celda de los cartones, una ficha será colocada (o retirada) sobre la celda. La pantalla permitirá al jugador anunciar "Bingo!" cuando considere que ha sido la primer persona en completar uno de los patrones y con ello reclamar el premio. Una vez finalizada una partida, el jugador será regresado automáticamente a la pantalla de *Play Session* para estar a la espera de la próxima partida.

El mapa del sitio de la Figura 7 se elaboró de la siguiente forma. Para cada tipo de contenido identificado en la fase de análisis de contenidos, se agregó un nodo en el diagrama. En forma simultánea se elaboraron los primeros esquemas de página de las pantallas de la aplicación (Sección 1.5.2). Para cada pantalla se creó un nodo diferenciado en el mapa (con otro color). Las pantallas se conectaron por relaciones de navegación. Los nodos que no son pantallas, sino contenidos, fueron asociados a las pantallas que las contienen con relaciones de agregación de UML (pertenencia). Los contenidos aún no conectados, fueron asociados a otros contenidos mediante relaciones de agregación o herencia de UML.

Avance de proyecto 7 [prj_site_map_design, 15 pts]

Elabore el mapa de sitio para su proyecto. Cree un nodo para cada contenido que recabó en la fase de análisis de contenido. Luego estructure los nodos para satisfacer las necesidades de información que identificó en la fase de análisis de usuarios. Si hay conflictos entre estos, priorice los roles con más usuarios. Provea un título e indique si se trata de una reestructuración o un sitio nuevo. Incluya una leyenda. Almacene su diagrama en una carpeta `design` de su repositorio de control de versiones. Agregue una sección "Diseño del sitio web" en su `README.md` e incruste el diagrama. Sugerencia: use una aplicación web de diagramación, como diagrams.net o Lucidchart.

Para probar mapas de sitio explique la estructura de su sitio web a un colega que conozca los requerimientos. Para probar con clientes espere a tener al menos bocetos, de lo contrario, los mapas serán muy abstractos para ellos. Explique el diagrama por partes, asociándolo con el sitio existente o futuro. Durante la sesión traten de determinar si la estructura es correcta, completa, y satisface las necesidades de los usuarios.

- Para saber si es completa, recorra la lista de tipos de contenidos que recabó durante el análisis y responda dónde iría este contenido en el sitio.
- Para saber si satisface las necesidades los usuarios, recorra la lista de roles y las *personas* que recabó durante la fase de análisis, y para cada necesidad trate de recorrer los pasos que el usuario haría sobre la estructura del sitio para lograrlo.
- Para saber si es correcta, pregunte a su colega si las secciones en que dividió el contenido, las etiquetas que escogió, y las conexiones que resaltó tienen sentido.

Las tres pruebas anteriores pueden realizarse de forma concurrente en un mismo recorrido por el diagrama. Es probable que realice varias pruebas. En cada nueva prueba resalte sólo los cambios nuevos.

Avance de proyecto 8 [prj_site_map_test, 5 pts]

Pruebe el mapa de sitio de su proyecto. Exponga su diagrama a un(a) colega explicando las decisiones que tomó y trate de determinar si hay omisiones. Haga las correcciones que necesite y refléjelas en control de cambios de su repositorio.

1.5.2. Esquemas de página

Un **esquema de página** (*wireframe* o *page schematic*) es un boceto que ilustra cómo ubicar los contenidos en una página web con el fin de hacer clara la comunicación y satisfacer las necesidades de sus usuarios. Los esquemas de página lucen como las páginas web finales (Figura 8) que el usuario experimentará, pero con pocos contenidos y detalles de diseño gráfico, para que puedan producirse de forma muy rápida en la fase de diseño del sitio web (Brown, 2011). En el caso de las aplicaciones web, los esquemas de página son bosquejos {*mockups*} de las pantallas que ayudan a los desarrolladores web a idear la interacción con sus usuarios.

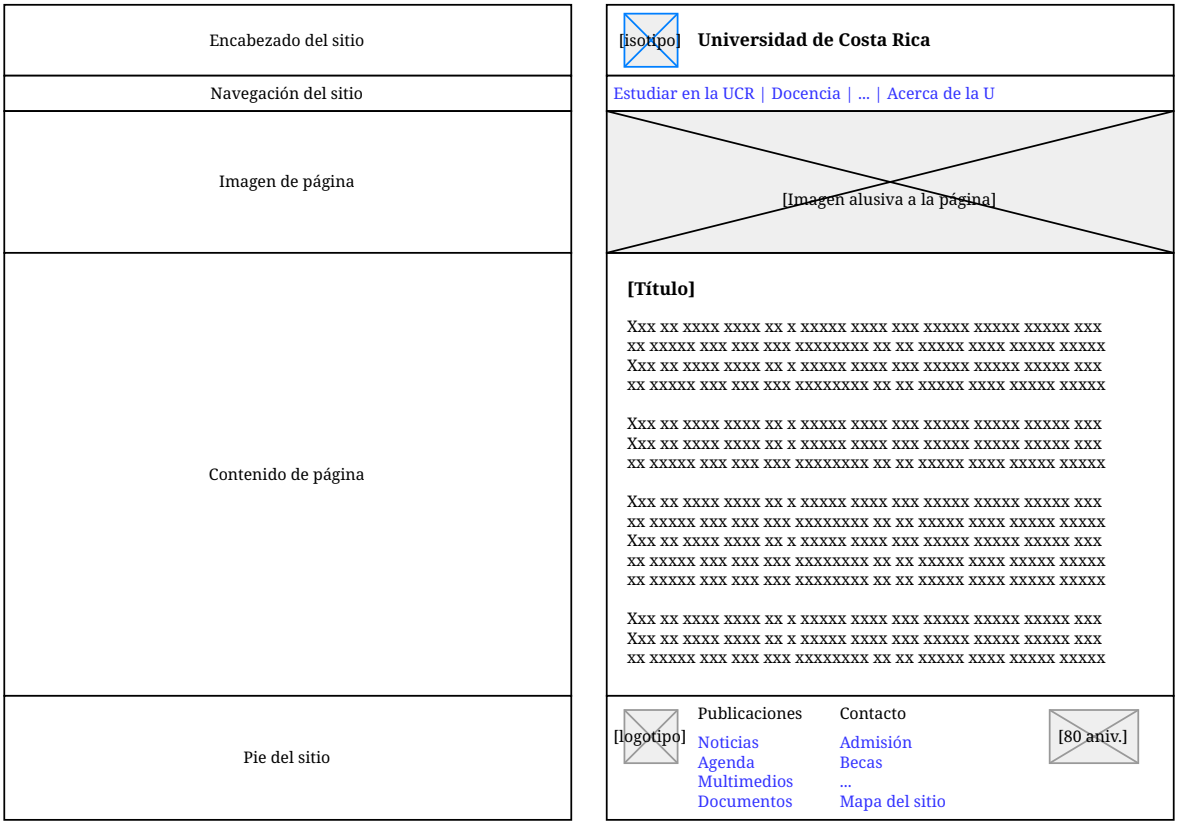


Figura 8. Esquemas de páginas secundarias en el sitio web de la Universidad de Costa Rica de fidelidad (a) baja (b) media

En el discurso técnico en español se emplea más el anglicismo *wireframe* que el término *esquema de página*. Literalmente *wireframe* es una armazón de alambre o estructura metálica. Aunque es una metáfora con origen en el campo de la construcción, el término *wireframe* se usa de forma exclusiva para la web en el léxico anglosajón. El término *wireframe* no ha corrido con la misma suerte que el término *web* en el Diccionario de la Real Academia Española, por lo que en este texto se preferirá el vocablo "esquema de página".

No es necesario elaborar esquemas para todas las páginas del sitio. Son candidatas aquellas que serán más visitadas y las marcadas como plantillas en el mapa del sitio. Normalmente un mismo esquema aplica para varias páginas del sitio, por lo que se consideran *plantillas* que al ser rellenadas con contenido y estilos producen las páginas web finales.

Eventualmente cada diseñador seguirá su propio método para elaborar bocetos. El Método sugerido de elaboración de esquemas puede usarse como punto de partida y que con la práctica puede refinarse y llegar a ser un proceso mecánico (hábito). Los resultados de este método serán aún mejores si es aplicado por un equipo web multidisciplinario. Si el equipo es grande puede aún subdividirse en equipos que realizan el proceso en paralelo y deliberan los últimos tres pasos (Brown, 2011).

Método sugerido de elaboración de esquemas

1. Ubique en el mapa del sitio la página o pantalla que va a bosquejar.
2. Tome una hoja de papel, preferiblemente que forme parte de un cuaderno de bocetos, y transcriba del mapa del sitio la identificación que tiene la página.
3. Identifique cuáles usuarios van a interaccionar con la página, y sus deseos o necesidades sobre esta página. Puede que ya haya recabado esta información en la fase de análisis de contenidos.
4. Pregúntese "¿qué debe tener esta página o pantalla para satisfacer esas necesidades?" Puede que ya haya recabado indirectamente esta información en la fase de análisis de usuarios.
5. Anote las respuestas como oraciones cortas en el papel.
6. Haga un dibujo rápido con la primera idea que le llegue a la mente para reflejar la primera respuesta. La calidad del dibujo no es importante, sino la idea.
7. Piense en otras formas de la página que podrían lograr el mismo objetivo y dibújelas aparte, no borre ni adapte los previos.
8. Si hay más necesidades o respuestas, haga nuevos dibujos para incorporarlas.
9. Al final tendrá una lluvia de ideas de dibujos, escoja el que siente que mejor satisface las necesidades de los usuarios.
10. Si lo necesita cree una versión digital del dibujo escogido, mediante escaneo o con ayuda de un software de ilustración.
11. Sólo en caso justificado, incremente la fidelidad del diseño gráfico en la ilustración.

Para dibujar un esquema se inicia con un rectángulo que representa la página web en diseño (véase la Figura 8(a)). Dentro de éste se dibujan más rectángulos que representan las regiones o áreas donde aparecerá el contenido en las páginas finales. La posición y tamaño de los rectángulos determinan parcialmente la prioridad que se quiere dar a los contenidos (Brown, 2011). Antes de llenar los rectángulos determine el nivel de abstracción y fidelidad (Figura 9) que tendrá el esquema.

- La **abstracción del esquema de página** {*wireframe abstraction*} se refiere a los trozos de contenido incluidos en el esquema. El contenido es *concreto* si se incluyen textos o recursos reales o realistas. Son útiles para usuarios del sitio porque ayudan a comprender un ejemplo realista de la página. El contenido es *abstracto* si se usan textos o símbolos sobre la naturaleza de los contenidos que eventualmente llenarán el rectángulo, tales como nombres de variables. Son útiles para desarrolladores web porque ayudan a comprender la estructura de la información. (Brown, 2011)
- La **fidelidad del esquema de página** {*wireframe fidelity*} se refiere al nivel de detalle de diseño gráfico. Varía desde *baja fidelidad* {*low-fidelity (lo-fi)*} que representa la página con figuras geométricas básicas de rápida producción, hasta *alta fidelidad* {*high-fidelity (hi-fi)*} que incorpora elementos de diseño gráfico realistas que hacen al esquema una representación fiel de la página final. (Brown, 2011)

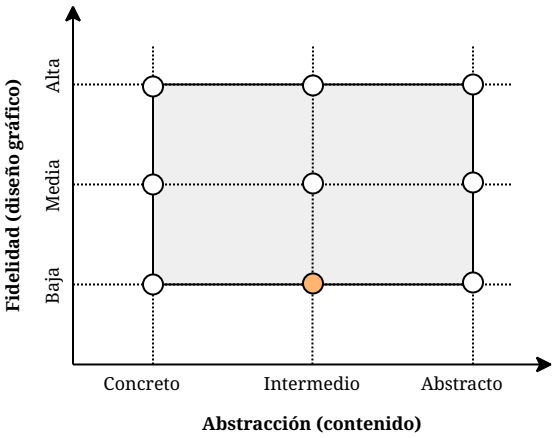


Figura 9. Clasificación de los esquemas de página por abstracción y fidelidad

La decisión de los niveles de abstracción y fidelidad, es decir uno de los nueve puntos de la Figura 9, depende de la audiencia a quien va dirigido el esquema de página. Si los esquemas son parte del diseño para el equipo de desarrollo, éstos preferirán esquemas abstractos que provean información sobre las variables y sus restricciones. Si los diagramas serán probados con usuarios finales, serán mejor comprendidos esquemas concretos y de mayor fidelidad. (Brown, 2011)

Debe considerar que entre mayor fidelidad del diseño gráfico, mayor será el tiempo de elaboración de un esquema, y de acuerdo al Método sugerido de elaboración de esquemas debería ser el último paso y sólo en caso necesario. Por lo tanto, se recomienda iniciar con esquemas a un nivel intermedio de abstracción y con baja fidelidad (punto resaltado en la Figura 9). Para este tipo de esquemas, en las regiones no se escriben contenidos, sino una etiqueta corta que indica su propósito, como "Pie del sitio" en la Figura 8(a). Las regiones en la página se pueden o no reutilizar en otras páginas del sitio, lo que permite clasificarlas en:

1. **Regiones globales** {*global regions*}. Son áreas que están presentes en todas las páginas del sitio web, normalmente el encabezado del sitio web {*website header*} y pie de página del sitio web {*website footer*}, como es el caso de la Figura 8. La consistencia de estas regiones comunican visualmente al visitante que las páginas pertenecen al mismo sitio web.

2. **Regiones de grupo** {*group regions*}. Son áreas compartidas entre varias páginas web, usualmente agrupadas en un mismo nivel del mapa del sitio. Por ejemplo, la región de la pantalla donde se ubica el menú secundario de navegación o un carrusel. La consistencia entre estas regiones en varias páginas comunican al visitante que pertenecen a la misma sección del sitio web.
3. **Regiones individuales** {*individual regions*}. Son las regiones para el contenido propio o exclusivo de la página web, es decir, aquel contenido que no se repite en otras páginas del sitio. Es el caso del "Contenido de página" en la Figura 8.

Se puede usar colores de relleno para distinguir regiones, siempre y cuando se apliquen consistentemente en todos los esquemas de página del mismo sitio web, por ejemplo, naranja para el encabezado del sitio, marrón para el pie de página, y así por el estilo. Esos colores son sólo para ayudar a distinguir las regiones en los diagramas, no quiere decir que sean los colores que tendrán en el diseño gráfico final. En cuanto al color del texto, se recomienda resaltar enlaces en azul, dado que las personas tienden a hacer esta asociación implícitamente, como se hizo en la Figura 8(b). Si los enlaces se encuentran dentro de texto, conviene además subrayarlos. (Brown, 2011)

Como se ilustra en el eje-x de la Figura 9, se puede variar el nivel de abstracción de los esquemas de página para adaptarlos a la audiencia. Si se considera al esquema de la Figura 8(a) en un punto intermedio (resaltado en la Figura 9), se puede incrementar el nivel de abstracción o de concreción. Un ejemplo es el esquema de la Figura 8(b) que tiene las mismas regiones del esquema Figura 8(a) con adaptaciones mixtas como las siguientes.

- Contenido concreto. Corresponde a ejemplos de contenido real que estará en el sitio web final o que se le asemeja. Es el caso de la navegación y el pie del sitio en la Figura 8(b). Tiene la ventaja de que ayuda a las personas a comprender la página de forma concreta, reduciendo interpretaciones subjetivas y malentendidos. Es práctico cuando se conoce bien un contenido de la plantilla, por ejemplo, tras la fase de análisis de contenido (Sección 1.4.2) y cuando es fácil trasladar trozos de contenido hacia el esquema de página.
- Patrones de contenido. Corresponde a patrones de texto con cierto formato reconocible, como fechas, números de identificación, números telefónicos. Es un ejemplo la fecha de actualización (ej.: `DD-MMM-AAAA`) al final del contenido de página en la Figura 8(b). Proveen un nivel intermedio de abstracción.
- Contenido de relleno. Corresponde a texto o contenido ininteligible que se usa para evocar la sensación visual que tendrán los usuarios cuando se reemplace por el contenido real. Proveen un nivel intermedio de abstracción. Son ejemplos textos como lorem ipsum y el texto de contenido de página en la Figura 8(b) hecho con líneas rellenas de equis ("xxxx xx"). Para imágenes, vídeos, u otros medios se pueden agregar áreas de relleno, como los rectángulos con diagonales de la Figura 8(b) que serán reemplazados por imágenes. Pueden tener una etiqueta con detalles sobre el medio que representan y sus restricciones.
- Etiquetas de contenido. Corresponde al nombre de un campo, usualmente encerrado entre corchetes, el cual será reemplazado por su valor cuando la plantilla se llene de contenido. Son ejemplos `[Imagen]` y `[Titulo]` en la Figura 8(b). Proveen un alto nivel de abstracción. Se pueden indicar restricciones sobre el campo, por ejemplo `[Aula:5,5]` podría indicar que el campo está limitado a 5 caracteres mínimo y máximo, y `[Sigla:\w\w\d\d\d\d]` debe cumplir la expresión regular indicada.

Si se elabora un esquema de página concreto para usuarios, es posible incluir información abstracta

para desarrolladores, o viceversa, en la descripción del esquema. Esta estrategia tiene la ventaja de unir las ventajas de ambos mundos y evitar redundancia.

Los esquemas de página para aplicaciones web reflejan las pantallas con las que interacciona el usuario. Normalmente se componen de controles de los formularios web {*widgets*} como campos de texto y listas desplegables, así como otros controles que han ido surgiendo en el campo de la web como carruseles y mapas geográficos. Si usa un programa de diseño, es probable que disponga de figuras pre-diseñadas para representar controles de interfaz de usuario web.

Dependiendo de las necesidades de su sitio web, pueda que deba diseñar *variaciones* de un mismo esquema de página (Brown, 2011). Los siguientes son algunos ejemplos.

- Si la página luce o muestra diferentes contenidos si el usuario está registrado {*logged in*} o no.
- Si la página tiene varios estados, como si el usuario ha ingresado datos correctos o no en un formulario, o si el contenido depende de otros contenidos en una base de datos.
- Si la página necesita adaptarse al medio en que se publica, como celulares inteligentes, tabletas, computadoras de escritorio, impresoras, u otros dispositivos.

Los esquemas de página son ambiguos incluso aunque sean concretos y de alta fidelidad. Uno o más lectores frente a un esquema pueden interpretar diferentes escenarios de interacción. Por lo tanto, se recomienda documentarlos. Puede acompañar a cada esquema con detalles como los siguientes:

Aspectos de un esquema de página

1. Identificación. Puede ser un nombre o un código, el cual debe ser uniforme entre el mapa del sitio y su esquema.
2. Variación. Si un esquema tiene variaciones, conviene que la identificación lo contemple.
3. Propósito de la página o pantalla.
4. Los supuestos de información que los usuarios deben conocer para lograr ese propósito.
5. Supuestos técnicos que deben cumplirse, como que debe existir al menos una instancia de una entidad en la base de datos.
6. A dónde lleva cada enlace, si no es obvio.
7. Ejemplos de los datos que se cargan en cada control de formulario y sus fuentes como bases de datos.
8. Reglas de validación de datos ingresados por los usuarios.
9. El programa en el servidor que recibirá los datos del usuario.

Mientras se lee una descripción puede ser difícil llevar el rastro de qué secciones del esquema hace referencia el texto. Una forma de facilitar el rastreo es *anotar* regiones en el esquema con números, y usarlos como referencia en el texto. El Ejemplo 5 muestra y detalla las pantallas para el proyecto Bingo Social.

Ejemplo 5. Avance 5 de Bingo: esquemas de página

Los esquemas de página para el proyecto Bingo Social se elaboraron simultáneamente con el mapa del sitio. Se siguió el Método sugerido de elaboración de esquemas. Primero se elaboraron esquemas abstractos de bajo nivel de fidelidad, como la Figura 10 para la pantalla principal, y la Figura 11 para la pantalla de sesión. Estos esquemas están ideados para dispositivos de baja resolución (320x480), como celulares inteligentes. En ambos casos, el objetivo de estos esquemas es mostrar como se distribuye el área disponible de la pantalla en regiones para la aplicación. El encabezado del sitio y el pie de página son comunes para todas las pantallas de la aplicación.

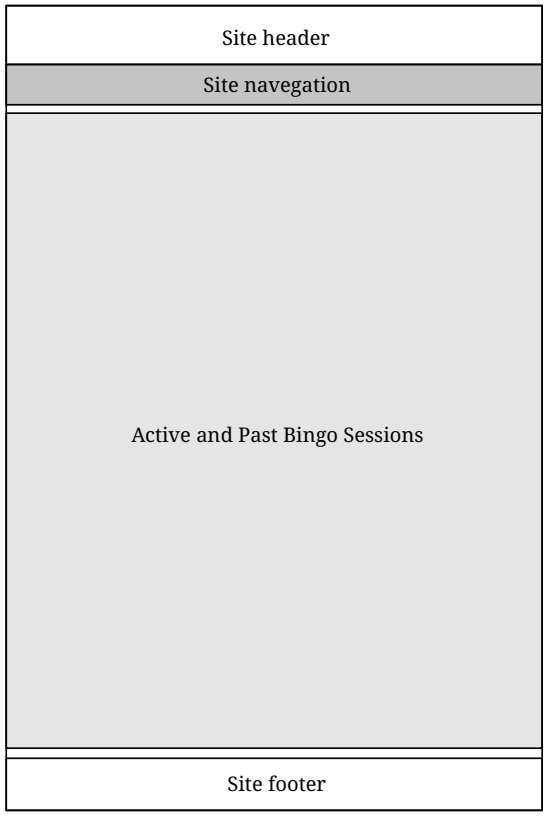


Figura 10. Pantalla principal abstracta

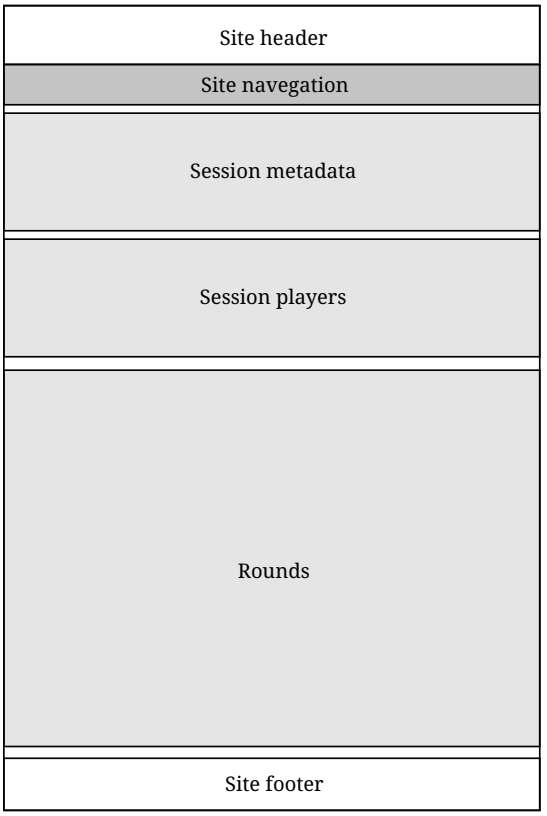


Figura 11. Pantalla de sesión abstracta

Para la página principal (Figura 10), el área central y más prominente se dedicará a mostrar las sesiones de bingo públicas activas o pasadas. Una vez que el moderador cree una sesión en la pantalla principal, se le mostrará la pantalla de sesión de la Figura 11. Esta pantalla es más compleja y está dividida en secciones que le permite ajustar las propiedades (metadados) de la sesión, ver los jugadores conectados, y administrar las partidas del juego.

Los esquemas de página abstractos de bajo nivel de fidelidad mostrados son convenientes para iniciar el diseño de las interfaces, pero no son suficientes para explicar la interacción entre los usuarios y el sistema. Se requieren al menos esquemas de mediana fidelidad. Para las siguientes pantallas se

presentarán sus versiones de fidelidad media y se documentarán sus Aspectos de un esquema de página.

Pantalla principal

El propósito de la pantalla principal (*Homepage*) (Figura 12) es permitir a los moderadores crear o administrar sesiones de bingo, y a los jugadores unirse a ellas. La pantalla muestra todos los visitantes las sesiones públicas ①, con el fin de atraer jugadores. Si el visitante está registrado o ha moderado sesiones de forma anónima en el dispositivo, la pantalla le permite ver la lista de las sesiones que ha creado ②. El visitante puede acceder a sesiones pasadas ③, por ejemplo para consultar los resultados.

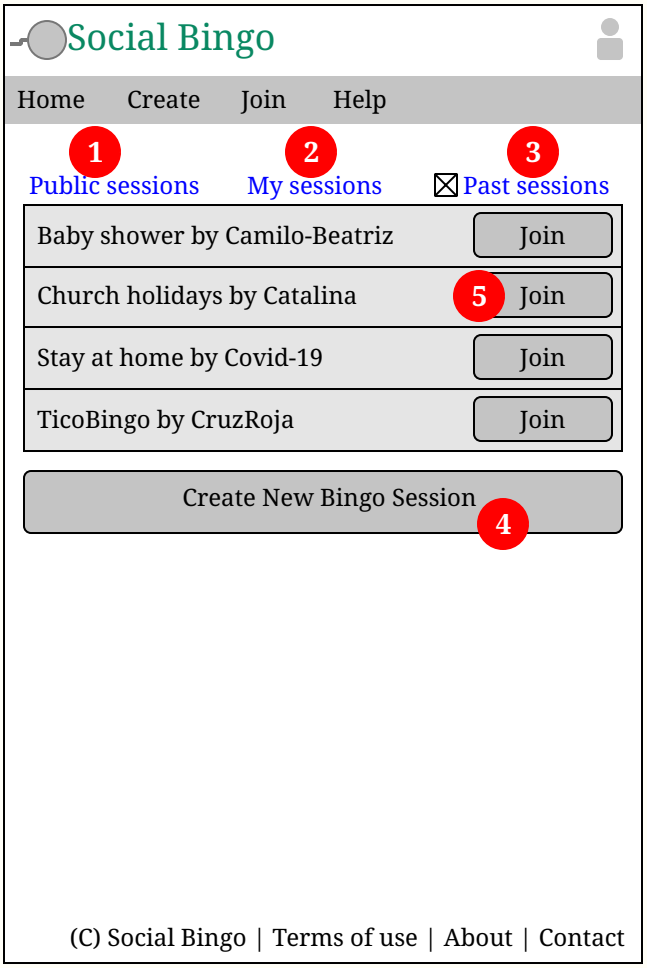


Figura 12. Esquema de la pantalla principal

Si el visitante quiere moderar una sesión, creará una nueva ④ o presionará el botón *Moderate* de una sesión que le pertenece. En cualquier caso será dirigido a la pantalla de sesión en su variante de moderación.

Si el visitante quiere jugar una sesión, presionará el botón *Join* ⑤ en alguna de las sesiones públicas, o en una de las sesiones que haya jugado antes. En cualquier caso será dirigido a la pantalla se sesión en

modalidad jugador.

Pantalla de sesión

La pantalla de sesión tiene dos variantes, una para moderar una sesión de bingo (Figura 13) llamada *Moderate Session* en el mapa del sitio, y otra para jugar una sesión de bingo (Figura 14) llamada *Play Session*. Prácticamente son la misma pantalla para ambos usuarios, con la diferencia que la mayoría de elementos son modificables para el moderador al accionarlos (resaltados en azul), y de sólo-lectura u ocultos para el jugador.

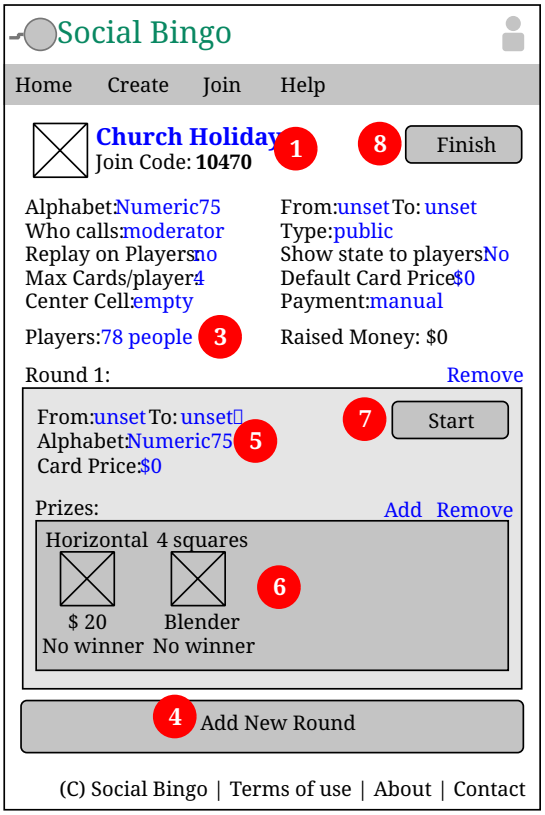


Figura 13. Pantalla para moderar una sesión

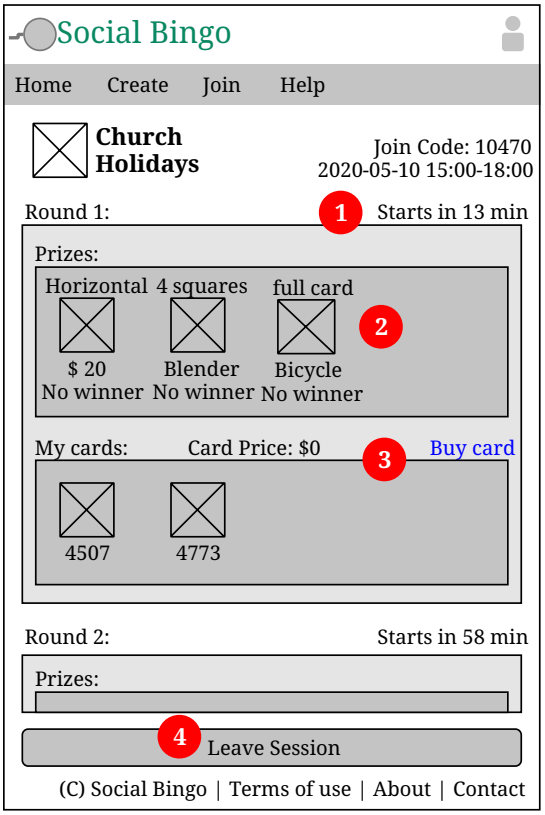


Figura 14. Pantalla para jugar una sesión de bingo

La pantalla de sesión permite al moderador (Figura 13) proveer un nombre e imagen para la sesión de bingo ① que ayude a los jugadores a reconocer el evento social. El sistema genera automáticamente un código (*Join Code*) que el moderador puede distribuir a los participantes en caso de que el bingo sea un evento privado. El moderador puede establecer varias propiedades sobre la sesión de bingo ②, como las listadas a continuación.

1. *Alphabet*. El alfabeto de símbolos del bombo y los cartones, que lleva a la pantalla de alfabeto.
2. *From-To*. La fecha y hora de inicio y fin del evento, para ayudar a los jugadores a estar preparados para el evento, así como poder agendarlo.
3. *Who calls*. Quien "canta" cada símbolo que sale del bombo: el moderador o el sistema. Si es el

sistema, éste reproducirá un sonido en el dispositivo del moderador, para lo cual cada símbolo del alfabeto debe tener un audio o vídeo pregrabado. Si es el moderador el sonido no se reproducirá automáticamente. En ambos casos, si el alfabeto tiene sonidos pregrabados, una opción de reproducción será provista que puede accionarse tantas veces como el usuario necesite.

4. *Replay on Players*. Si se deshabilita, el sonido de cada símbolo que sale del bombo será reproducido únicamente en el dispositivo del moderador. Si se habilita, el sonido será reproducido en los dispositivos de los jugadores, lo cual es útil para un juego de bingo remoto. Si el moderador es quien "canta" los símbolos, podrá usar el micrófono de su dispositivo y el sonido será reproducido en los dispositivos de los jugadores.
5. *Type*. Permite al moderador escoger si su evento de bingo es público o privado. Un evento público será visible en la página principal del sitio, con el fin de anunciarlo a más jugadores potenciales. Un evento privado no será visible en la página principal, sino que los jugadores tendrán que unirse ingresando el *Join Code* autogenerado por el sistema.
6. *Max Cards/Player*. Permite al moderador limitar el número de cartones máximo que puede adquirir un jugador. Si se establece en 0, será ilimitado.
7. *Show State to Players*. Si se habilita, los jugadores verán en sus dispositivos el estado del juego. Éste corresponde a una tabla con todos los símbolos de los cartones, y aquellos que han sido extraídos del bombo se encuentran resaltados. Si se deshabilita, el estado del juego no se muestra en los dispositivos de los jugadores. En cualquier caso, el moderador podría proyectar el estado del juego con los símbolos del bombo que estará disponible en su dispositivo.
8. *Default Card Price*. Precio por defecto de los cartones en dinero real. Si se establece un monto mayor a cero, los jugadores deberán contar con alguna forma de pago para poder adquirir los cartones, incluso dinero en efectivo. Este precio es el por defecto, pero al igual que otras propiedades, podría variar luego de una partida a otra.
9. *Payment Method*. El método de pago. Si se establece como electrónico, los jugadores deberán ingresar un método de pago electrónico como una tarjeta bancaria o un servicio como PayPal para poder adquirir cartones. Si se establece como manual, los jugadores comprarán cartones directamente al moderador con alguna forma de pago como dinero en efectivo. En tal caso, el moderador otorgará los cartones a los jugadores.
10. *Reuse Cards* (no mostrado). Permite al jugador en una ronda usar los cartones que haya adquirido en rondas previas. Puede configurarse para forzar a comprar nuevos cartones, a reusar cartones del mismo tipo y precio.
11. *Sell Back Cards* (no mostrado). Permite a los jugadores regresar cartones que no han jugado y recuperar el dinero que pagaron por ellos.

La sección de jugadores (*Players*) ③ en la pantalla de sesión (Figura 13) permite al moderador administrar la lista de jugadores registrados en la sesión, otorgarles cartones, y ver la cantidad de dinero recaudado en el evento. Al accionar el número de jugadores, lleva a la pantalla de administración de jugadores.

Una sesión consta de partidas. El botón *Add New Round* ④ permite agregar partidas a la sesión. Cada partida está numerada, y podría ser removida si no se va a jugar. Toda partida hereda las propiedades ⑤ de la sesión, y el moderador podría sobrescribirlas, por ejemplo, para hacer que los cartones tengan un alfabeto o precio diferente en cada ronda.

Cada ronda tiene premios ⑥. El moderador puede indicar los premios disponibles y los patrones que deben completarse para ganarlos. Por ejemplo, en la Figura 13 el primer jugador que complete las cuatro esquinas ganará una licuadora.

Las rondas son editables hasta el momento en que se inician. Cuando el moderador considere que los participantes estén listos, iniciará la ronda con el botón *Start* ⑦, que lo llevará a la pantalla de ronda. Una vez que la ronda haya finalizado regresará a la pantalla de sesión donde podrá iniciar la siguiente ronda. Cuando todas las rondas hayan finalizado o cuando el moderador lo considere necesario, podrá finalizar la sesión ⑧, lo cual terminará las transmisiones de audio, marcará la sesión como finalizada y será presentada en la pantalla principal en las listas de sesiones pasadas.

La pantalla de sesión para el jugador (Figura 14) es similar a la del moderador, pero oculta la mayoría de propiedades. El jugador podrá ver el tiempo esperado de inicio y las rondas prestablecidas ①. En cada ronda podrá consultar los premios y los patrones para ganarlos ②. La principal diferencia con la pantalla de moderación está en que el jugador puede adquirir cartones al precio establecido para la ronda ③. Si el moderador lo ha permitido, el jugador verá los cartones que haya adquirido en rondas previas. Al accionar un cartón lo verá una pantalla superpuesta que le permitirá estudiarlo antes de iniciar la ronda. Un jugador puede abandonar la sesión ④, por ejemplo, en caso de una eventualidad, lo que permite al moderador tener realimentación de su audiencia efectiva.

Pantalla de ronda

(Documentación pendiente)

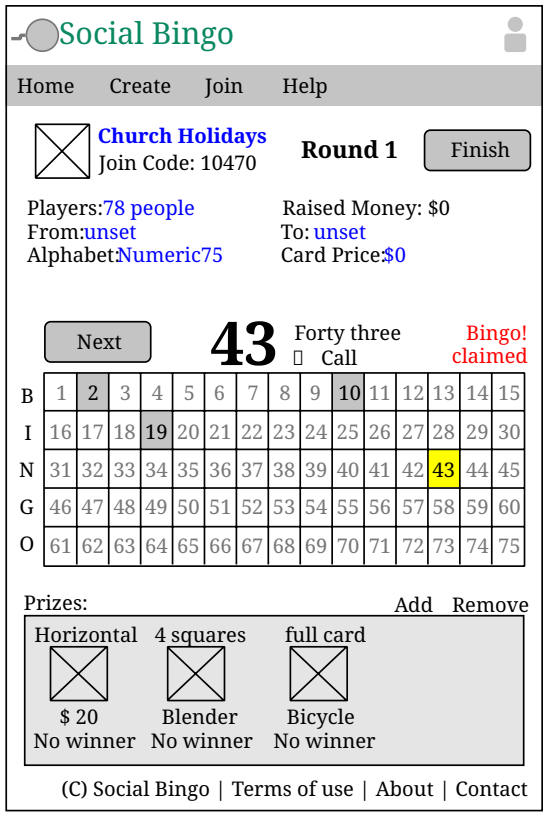


Figura 15. Pantalla para moderar una partida

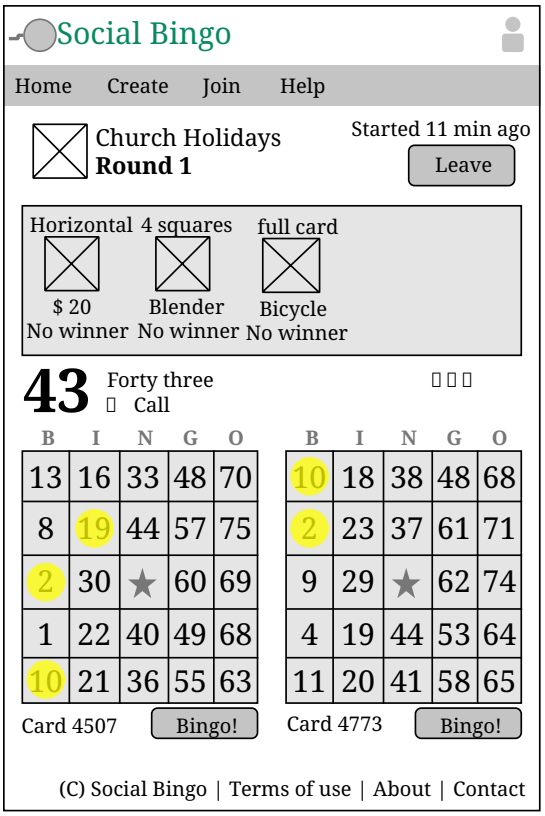


Figura 16. Pantalla para jugar una partida de bingo

Pantalla de alfabeto y administrar jugadores

(Documentación pendiente)

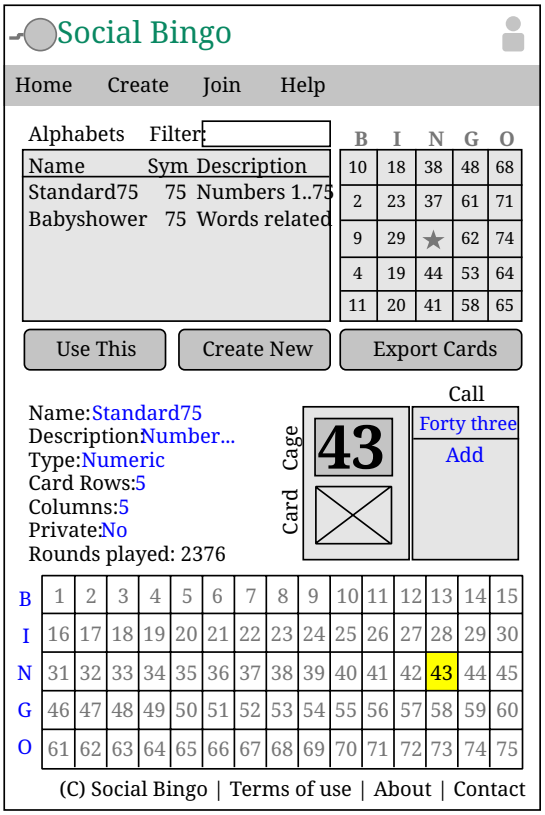


Figura 17. Pantalla para moderar el alfabeto

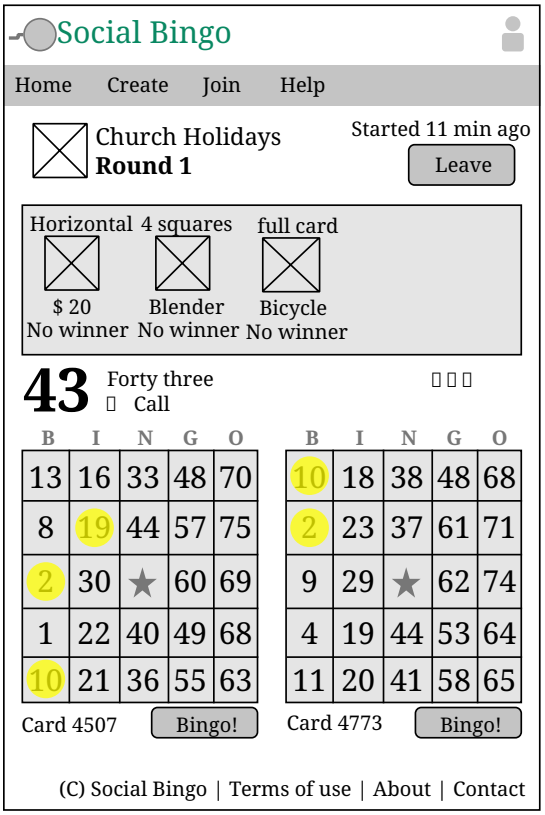


Figura 18. Pantalla para administrar jugadores

Avance de proyecto 9 [prj_wireframe_design, 25 pts]

Elabore los esquemas de página para su proyecto. Para cada pantalla en su mapa de sitio cree un esquema de página siguiendo el Método sugerido de elaboración de esquemas hasta que alcancen un nivel medio de fidelidad.

Elabore varios esquemas para cada pantalla primero en papel, en forma de lluvia de ideas. Para cada pantalla repase las secciones del documento de análisis (contexto, contenidos, y usuarios) relativas a la pantalla. Asegúrese que su esquema permite a los usuarios realizar sus propósitos con la pantalla. No borre esquemas, sino cree nuevos con cada idea que surja. Finalmente marque en el papel el esquema que considere satisface mejor las necesidades y deseos de los usuarios.

Si se trabaja en equipo, cada miembro debe realizar el paso descrito en el párrafo anterior de forma independiente, aislada de la influencia de los demás. Una vez que todos los miembros hayan concluido, compartan y discutan sus esquemas. Es esperable que se realice una dinámica de debate de crítica y defensa de diseños. Mediante votación o discusión adapten el diseño candidato en papel para cada esquema de pantalla.

Transcriba los diseños candidatos a un programa de ilustración de esquemas de páginas, como Figma, Sketch, o Adobe XD. Almacene los esquemas de página resultantes de cada pantalla en una carpeta `design/` de su repositorio de control de versiones, preferiblemente en un formato vectorial (por ejemplo, SVG).

Avance de proyecto 10 [prj_wireframe_doc, 15 pts]

Documente cada esquema de página resultante como se hizo en el Ejemplo 5 para el proyecto de bingo.

En la carpeta `design/` de su repositorio de control de versiones cree un documento `design.md` y en ella una sección para los esquemas de página (título de nivel 2). Para cada pantalla de su aplicación cree una sub-sección (título de nivel 3). Incruste la imagen del esquema de página correspondiente y documente cada componente de ella, de tal forma que quede claro todo lo que los usuarios pueden realizar en la pantalla, o hacia dónde llevan sus enlaces. Use anotaciones para ligar el texto con la imagen. Puede anotar el esquema usando números encerrados en círculos o nubes {*callouts*} y usar los mismos en el texto.

Es normal que necesite realizar cambios en el esquema de pantalla. Si trabaja en equipo, discútalos con sus colegas antes de incorporarlos en el esquema.

1.5.3. Modelos de comportamiento

Los mapas de sitio y los esquemas de página son vistas estructurales de la solución, pero limitan diseñar la riqueza del comportamiento dinámico de algunas aplicaciones web, en especial aquellas que constan de pocas pantallas cuyo contenido cambia dinámicamente de acuerdo a las interacciones con el usuario (Laubheimer, 2016). Esta sección presenta dos tipos de diseños, los *wireflows* y *wirestates*, que enriquecen los esquemas de página con aspectos del comportamiento de las aplicaciones.

Un ***wireflow*** es un formato de diseño que combina esquemas de páginas con elementos de diagramas de flujo para representar interacciones entre los usuarios y las pantallas de la aplicación. Puede pensarse como un diagrama de flujo que cambia sus pasos abstractos por esquemas de página que facilitan a los lectores comprender la interacción con el sistema (Laubheimer, 2016).

El término *wireflow*, literalmente "flujo de alambres", proviene de fusionar el esquema de página (*wireframe*) con el diagrama de flujo (*flowchart*). Es un tecnicismo exclusivo para el campo de la web en el caso del idioma inglés, y no tiene una traducción reconocida al español, por lo que se tratará como un anglicismo en este texto.

Un *wireflow* es un diagrama de flujo, y en consecuencia hereda sus propiedades. Un diagrama de flujo es una representación gráfica de un algoritmo, y por tanto, una secuencia finita y ordenada de pasos para resolver un problema. Un *wireflow* documenta un proceso, una tarea a realizar con el sistema. La cantidad de tareas que se puede realizar con un sistema puede ser numerosa, sin embargo, es habitual crear *wireflows* sólo para las tareas más comunes que realizarán los usuarios.

Los nodos en un *wireflow* son esquemas de páginas o trozos de ellas. En un nodo se resaltan las áreas de la pantalla con las que el usuario se espera que interaccione para lograr la tarea, por ejemplo, presionar un botón o ingresar un texto. Las áreas resaltadas se conectan mediante arcos dirigidos hacia otros nodos. Una relación entre dos nodos *A* y *B* indica que cuando el usuario interacciona con el área resaltada del esquema de página *A*, verá como resultado el esquema de página en *B*.

Los nodos *A* y *B* pueden ser pantallas distintas, o bien la misma pero en estados distintos. Si los nodos *A* y *B* son el mismo esquema de página, no es necesario incluir en *B* la información que no cambia de estado, sino que *B* podría ser un fragmento del esquema que resalta el cambio.

El arco dirigido entre dos nodos puede conectarlos de forma directa, pero no está restringido a ello. Un *wireflow* tiene todo el poder de un diagrama de flujo. Un arco podría pasar por una serie de pasos, entre los que se incluyen condicionales que pueden llevar a distintos destinos, por ejemplo, dependiendo de si el usuario ingresa un valor válido o no en un campo. El *wireflow* podría contemplar pasos no visibles para los usuarios como accesos a bases de datos, pero no se recomienda incluirlos si se van a realizar pruebas con usuarios finales.

Ejemplo 6. Avance 6 de Bingo: *wireflow* crear una partida

(Contenido pendiente)

Avance de proyecto 11 [prj_wireflow_design, 15 pts]

Identifique las tareas principales que realizarán los usuarios con su aplicación. Para cada una elabore un *wireflow*. Exporte sus *wireframes* a la carpeta `design/` de su repositorio de control de versiones, preferiblemente en formato vectorial (por ejemplo, SVG).

Documente sus modelos de comportamiento. En el documento `design.md` incluya un capítulo para *Wireframes*. Cree una sección para cada tarea común identificada. En la sección incluya el *wireframe* correspondiente y bajo éste explique las interacciones que el usuario realizaría para conseguir su propósito, en especial las menos triviales.

Un *wireflow* narra el flujo de interacciones que los usuarios realizan para lograr una tarea, y un sistema puede ofrecer muchas tareas posibles. Si se usan como punto de partida para el diseño del sistema, los *wireflows* rápidamente pueden abrumar a los diseñadores en detalles. Un diagrama de flujo confiere mucho poder y control a los desarrolladores, ya que son una forma imperativa de resolver problemas. Otros tipos de diseños pueden ayudar a controlar el nivel de detalle a través de mecanismos de abstracción de más alto nivel, como las máquinas de estado.

Una **máquina de estados finitos** (*finite-state machine*), también conocido como **autómata de estados finitos** (*finite-state automaton*) es un modelo abstracto compuesto de un conjunto de estados, de tal forma que un programa sólo puede estar en un único estado en un tiempo dado. El programa puede pasar a otro estado ante la aparición de un evento o entrada. En la transición de un estado a otro, la máquina puede realizar operaciones procedimentales, pero no atender nuevos eventos.

Así como los *wireframes* se combinan con diagramas de flujo, también pueden combinarse con otros tipos de diseños. Al combinar *wireframes* como nodos de una máquina de estados finito, se obtiene un tipo de diseño al cual no se le ha otorgado un nombre, y en este material se propone el acrónimo ***wirestate***.

Un *wirestate* tiene mayor nivel de abstracción que un *wireflow* gracias al paradigma de programación orientada a eventos. En un *wirestate* los nodos son *wireframes* que representan un estado de la aplicación web. Se puede pensar que mientras la aplicación web se encuentre en un estado, estará a la espera de eventos del usuario. Cuando el usuario produce un evento, el sistema reaccionará en respuesta realizando una transición hacia otro o el mismo estado. Al igual que los *wireflows* dos nodos podrían representar diferentes estados de la misma pantalla.

Las transiciones en un *wirestate* tienen dos partes rotuladas de la forma `evento | reacción`. El evento es generado por el usuario (por ejemplo, presionar un enlace) u otros sistemas informáticos (por ejemplo, la respuesta de un servicio web).

La reacción es una tarea que la aplicación web realiza en respuesta al evento. En el *wirestate* basta con darle un nombre la tarea, y ésta puede definirse luego mediante otros paradigmas. La tarea corresponde a una subrutina que se puede expresar posteriormente a través de un algoritmo (paradigma de programación procedimental), una función matemática (programación funcional), o un método de clase (programación orientada a objetos). De esta forma, un *wirestate* es un diagrama de muy alto nivel de abstracción que representa todos los estados del sistema, y cada transición ayuda a identificar algoritmos que el sistema deberá implementar.

Aunque un *wirestate* tiene similitudes con un diagrama de flujo, no lo es. Un autómata de estados finitos sólo consta de estados y transiciones, y éstas de eventos y reacciones. Se puede diferenciar el estado inicial y el estado final cuando el programa es ejecutado o termina su ejecución respectivamente. Lo demás no pertenece a un diagrama de estados. Los siguientes síntomas y heurísticas pueden ayudar a evitar el error común de tratar de combinarlos.

1. Si siente la necesidad de agregar un rombo para indicar un condicional. Los únicos condicionales permitidos son dos o más transiciones que parten de un mismo estado a través de eventos distintos generados por usuarios o sistemas externos. Si el condicional es de otra naturaleza, quizá deba ir como parte del algoritmo de una reacción.
2. Si tiene transiciones sin nombres de eventos ni reacciones. Recuerde que un estado es una espera por eventos del sistema. Cada transición que parte del estado debe originarse por un evento de usuario o de un sistema externo que se estaba esperando en dicho estado. Si no puede identificar

el evento para una transición, pueda que deba eliminarse.

3. Si tiene estados con una única transición de llegada y una única transición de salida. El estado puede ser válido, siempre y cuando esté a la espera de un único evento permitido de usuario o de sistema externo, por ejemplo, confirmar una pregunta. En otro caso, si las transiciones ocurren de forma inmediata, pueda que el estado sólo sea un paso de un algoritmo.

Ejemplo 7. Avance 7 de Bingo: *wirestate* moderar y jugar una partida

(Contenido pendiente)

Avance de proyecto 12 [prj_wirestate_design, 15 pts]

Elabore un diagrama de máquina de estados abstractos. Identifique los estados en los que su aplicación esperará por eventos de usuario, y dibújelos como estados abstractos en el diagrama. Identifique los eventos que generará el usuario en cada estado y los nuevos estados a los que cambiará la aplicación. Para cada transición, dele un nombre a la reacción que ejecutará la aplicación.

Si es necesario rotule el estado inicial y final de la aplicación web. Revise que su modelo cumple con las características de un autómata de estados finitos y no incluya elementos de diagramas de flujo.

Cambie los estados abstractos por esquemas de página {*wireframes*} que ayuden a explicar gráficamente el estado y los elementos de interfaz que el usuario acciona con el fin de cambiar de estado. Exporte sus *wirestate* a la carpeta `design/` de su repositorio de control de versiones, preferiblemente en formato vectorial (por ejemplo, SVG).

Documente sus modelo de estados. En el documento `design.md` incluya un capítulo titulado *Wirestate*. Cree una sección para cada reacción que realizará el sistema. En ella indique en una oración el propósito de la subtarea. Escriba una lluvia de ideas en formato del algoritmo (código preformateado o listas anidadas) de lo que el sistema realizará en respuesta al evento.

1.5.4. Prueba del diseño

Todo artefacto elaborado en un proceso de resolución de problemas ingenieril debe ser probado. La mayoría de los artefactos resultantes de la fase de diseño del sitio web giran sobre las interfaces con las que interaccionarán usuarios. El propósito de la prueba de diseño es obtener retroalimentación de expertos (evaluación heurística) o de los usuarios (pruebas de usabilidad) de forma inmediata conforme los diseños son elaborados.

Normalmente las herramientas de diseño de esquemas de página {*wireframing tools*} también proveen funcionalidades de prototipado {*prototyping tools*}. Permiten proveer manejadores de eventos a potenciales interacciones de los usuarios, los cuales son apropiados para aplicaciones sencillas. Sin embargo, si la lógica del dominio es compleja, implementar un prototipo computacional tiene la misma complejidad que implementar el sistema final, lo que aleja la posibilidad de obtener realimentación temprana.

Se dispone de una riqueza de métodos e instrumentos de evaluación de interfaces de usuario que pueden ayudar a superar el problema descrito (Tullis & Albert, 2013). Un ejemplo de una técnica más

rápida de elaborar es el caso de los prototipos en papel, que pueden evaluarse con una prueba de usabilidad.

Un prototipo en papel es una maqueta impresa o dibujada de la interfaz del programa. Aunque el prototipo no puede interaccionar directamente con un usuario, la persona desarrolladora puede "darle vida" al tomar el rol de la computadora. Se presenta a continuación un método con fines pragmáticos para evaluar la usabilidad o experiencia de usuario con prototipos en papel, que puede ser adaptado a conveniencia de acuerdo a las necesidades del proyecto.

Método pragmático de prueba de diseño

Preparación

1. Escoja al menos una tarea a evaluar, entre las tareas que los usuarios realizarían con el sistema.
2. Elabore, si no los tiene ya, los esquemas de página a nivel medio de fidelidad que intervienen en la tarea.
3. Imprima o dibuje en papel los esquemas, sus componentes, y recortarlos.
4. Simule la tarea que se va a evaluar con el prototipo en papel. Realice usted este paso de forma individual, llámesele auto-prueba.
5. Si en la auto-prueba identifica cambios necesarios, anótelos y luego regrese al paso 2, hasta que en la auto-prueba usted logre completar la tarea.
6. Prepare materiales de instrucción si es necesario. Es muy probable que cuando los visitantes interaccionen con la aplicación web final, usted no esté disponible para explicarles cómo usarlo. En tal caso, elabore instructivos de uso. Un primer instructivo puede ser textual apoyándose en capturas de pantalla o fotografías del prototipo. Puede elaborar un vídeo que cómo usar el sistema. Puede grabar un vídeo corto aprovechando el prototipo en papel que tiene de frente en una mesa.

Prueba de usabilidad

1. Invite una persona, idealmente que desconozca la aplicación web, aunque puede conocer sobre el dominio del problema que esta resuelve. Puede invitar más participantes si la aplicación requiere que dos o más personas interaccionen simultáneamente.
2. Explique al participante la tarea que se quiere resuelva con el prototipo, pero no cómo lograrlo. Indíquele que puede suspender la actividad en el momento que lo considere necesario y que se está evaluando al prototipo, no al usuario. Pídale que interaccione con el sistema como si fuera un sitio web normal. Si va a usar un protocolo de *think-aloud*, pídale al participante que en todo momento diga en voz alta lo que pasa por su mente.
3. Grabe la sesión, idealmente con vídeo desde un dispositivo que registre la actividad que ocurre en la mesa de trabajo.
4. No provea instrucciones al participante de cómo usar el sistema, a menos de que se vuelva necesario. Idealmente una aplicación debe ser intuitiva o autoexplicativa. La aplicación debe proveer mecanismos que permitan a los usuarios obtener ayuda. En caso de el participante la solicite, puede mostrarle el instructivo en papel o el vídeo precargado en un dispositivo de reproducción.
5. Ensamble el prototipo en papel sobre una mesa de trabajo, y explíquele al participante cómo interaccionar con este tipo de prototipos. Por ejemplo, el participante puede tocar con el dedo para accionar los controles en papel, pero no puede reubicarlos dentro de las pantallas, a menos

de que esto tenga sentido en la aplicación web final.

6. Permita que al participante interaccionar con el prototipo en papel. Cuando éste lo haga, generará un evento. Usted reacciona como lo haría la aplicación web, reubicando elementos de la interfaz. Si tiene que proveer una respuesta personalizada, puede hacerlo escribiendo o dibujando sobre la interfaz con un lápiz. También puede borrar o reemplazar elementos de la interfaz.
7. No explique ni converse mientras reacciona a los eventos del participante, a menos de que el sistema final lo hiciera. Una vez que haya terminado de ensamblar la interfaz, retire sus manos y permita que el usuario interaccione.
8. Si el participante se detiene de conversar y está usando protocolo de *think-aloud*, recuérdele decir en voz alta lo que está pensando. Si olvida la tarea, puede explicarla. Si el participante tiene dudas o dificultades, anótelas o trate de recordarlas luego. Si el participante se bloquea sin saber qué hacer en la interfaz, puede ofrecerle ayuda, pero asegúrese de que quede registro, como en la grabación de vídeo de la sesión. Esta ayuda que debió ofrecer, es probable que otros usuarios también la necesiten y debería ser incluida en el prototipo.
9. Una vez que el participante logre la tarea o desista, finalice la actividad y pase a la entrevista.

Entrevista

1. Tras la sesión puede puede solicitar al participante su opinión sobre la herramienta mediante un instrumento estandarizado como el Cuestionario de Experiencia de Usuario (UEQ, *User Experience Questionnaire*).
2. También puede realizarle una entrevista, sobre todo para conocer más sobre las dificultades que tuvo con la aplicación, y sugerencias de cómo se podría mejorar el prototipo para evitar estas dificultades a más usuarios.
3. Si no utilizó un protocolo de *think-aloud*, puede reproducir la grabación con el participante, y mientras esta transcurre, hacerle preguntas de qué pasaba por su mente durante cada evento, las dificultades que experimentó, y cómo se puede mejorar la aplicación.

Mejoras al prototipo

1. Una vez terminada la sesión con el participante, puede repasar las anotaciones que haya realizado y la grabación de la sesión. En una hoja de cálculo recabe una lista de problemas de usabilidad del prototipo.
2. Clasifique los problemas de usabilidad identificados por su severidad en: *alta* si impide al participante completar la tarea, *media* si dificulta completar la tarea aunque puede lograrse por medios alternativos, y *baja* si molesta o frustra a los participantes aunque no impide completar la tarea (Tullis & Albert, 2013).
3. Puede regresar a la etapa de preparación y adaptar el prototipo para superar los problemas de usabilidad, al menos los de alta severidad, y deseablemente los de severidad media, antes de pasar al próximo participante.
4. Realice varias pruebas de usabilidad. El número de pruebas dependerá de los recursos con los que cuente, en especial el tiempo y recurso humano. Si dispone de estos recursos, realice el proceso al menos tres veces, y puede detenerse cuando la realimentación que obtenga no provea nuevos hallazgos.
5. Si hace mejoras al prototipo entre cada prueba, los instrumentos estandarizados le ayudarán a determinar si las mejoras al prototipo incrementan la usabilidad o mejoran la experiencia de

usuario o no. Sería esperable que la lista de problemas de dificultad se reduzca con cada réplica de la prueba. Si no incluye mejoras entre pruebas, los problemas de usabilidad que resurgen son garantía de consistencia y no experiencias fortuitas de un participante particular.

Cada prueba de usabilidad que realice es una oportunidad para mejorar cualquiera de los artefactos realizados sobre el sitio web. La prueba podría hacer evidente una necesidad que no fue documentada durante el análisis de usuarios, o una pantalla o página que no fue considerada en el mapa del sitio ni en los esquemas de página, o un evento que no fue diagramado en la máquina de estados finitos. Es importante atender estos problemas antes de implementar la aplicación web, dado que es más barato y conveniente realizar modificaciones en diseños que en artefactos de software.

Los videos que grabe de las pruebas de usabilidad son una fuente rica de diseño de la solución. Al usted tomar el rol del sistema, las acciones que tuvo que realizar como respuesta a cada evento de usuario, son acciones que la aplicación web tendrá que realizar de forma automática. Mientras observa los videos, preste atención a estas acciones que realizó con los componentes en papel, y anótelos en las reacciones del sistema a los eventos del usuario en el *wirestate*.

Avance de proyecto 13 [prj_design_test1, 25 pts]

Realice la primera prueba de usabilidad siguiendo el Método pragmático de prueba de diseño. Prepare un prototipo en papel y haga la auto-prueba hasta que se sienta fluido(a) manipulando el prototipo. Prepare instrucciones para usar el prototipo, idealmente en vídeo de corta duración.

Realice la primera prueba de usabilidad con participantes ajenos a la aplicación web como se indica en el procedimiento sugerido. Es importante que grabe la sesión.

Una vez completada la sesión, pídale al participante completar el cuestionario UEQ. Entreviste al participante sobre las dificultades que enfrentó y para obtener sugerencias de mejora del prototipo. Recabe una lista de problemas de usabilidad.

Una vez finalizada la prueba, estudie el vídeo de la sesión y complete la lista de problemas de usabilidad. Clasifique los problemas por severidad en una hoja de cálculo. Agregue la hoja a su carpeta `design/`.

Avance de proyecto 14 [prj_design_improv1, 15 pts]

Haga correcciones y mejoras en los artefactos con el fin de superar al menos los problemas de alta y mediana severidad que haya identificado en la hoja de cálculo. Los cambios podrían mejorar:

- Documentos de análisis de contexto, contenidos, o usuarios.
- El mapa del sitio.
- Los esquemas de página {*wireframes*}.
- Los *wireflows* de las tareas.
- El *wirestate* del sistema.

Repase el vídeo grabado durante la primera prueba de usabilidad. Preste atención a las acciones que tuvo que realizar en respuesta a los eventos del participante. Agregue estas acciones a los algoritmos (reacciones) del *wirestate* en su documento `design.md`.

Avance de proyecto 15 [prj_design_test2, 25 pts]

Realice al menos una segunda prueba de usabilidad siguiendo el Método pragmático de prueba de diseño con participantes distintos a los de las pruebas previas. Si participa de un curso formal, consulte con su profesor(a) la cantidad y logística de las pruebas.

Recuerde siempre grabar la sesión y pedirle a los participantes completar el cuestionario UEQ. Entreviste a los participantes sobre las dificultades que enfrentaron y sugerencias de mejora del prototipo. Estudie el vídeo de la sesión. Agregue los nuevos problemas de usabilidad identificados a la hoja de cálculo y clasifique su severidad. Si un problema resurge, no lo reitere, sino que lleve un contador que indique la cantidad de veces que ha sido reportado. Problemas reportados con reiteración son más urgentes de atender.

Digitalice las respuestas al cuestionario de usabilidad y compárelas usando las herramientas de análisis provistas en su sitio web. Trate anecdóticamente de determinar si la experiencia de usuario mejora tras cada prueba con los cambios que incorpora en el prototipo.

Tras cada prueba de usabilidad, haga las correcciones y mejoras a los artefactos para superar al menos los problemas de alta y mediana severidad. Si durante la sesión realizó acciones distintas, agréguelas a los algoritmos de las reacciones del *wirestate*. Tras cada prueba, agregue estos cambios a control de versiones.

Al realizar pruebas con usuarios de su prototipo obtendrá una sensación de seguridad al saber que su diseño realmente satisface sus necesidades o deseos. La teoría confirma que el tiempo invertido en diseñar y probar los modelos es inferior al tiempo que se tiene que invertir en reingeniería posterior {*refactoring*} (Rosenfeld et al., 2015), la cual encima genera productos de inferior calidad, más difíciles de mantener, y con los que da menos gusto trabajar.

1.6. Resumen

El equipo web. La tripleta esencial: el arquitecto de la información se encarga del contenido (parte 2), el diseñador gráfico de la presentación (parte 3), y el informático del comportamiento (parte 4).

Los mapas de sitio y los bocetos permiten diseñar la *estructura* estática de la solución web. Para un sitio web dinámico, al menos uno de los nodos del mapa del sitio será una aplicación web. El diseño del comportamiento dinámico de las aplicaciones que conforman la solución web depende de los *paradigmas de programación* que se vayan a emplear. La cuarta parte de este material presenta varios paradigmas de programación

Capítulo 2. Fundamentos de la web

En este capítulo se presenta una breve historia de la web que ayudará a ubicar este fenómeno en el contexto social e introducir sus primeros conceptos. Seguidamente se presentará la arquitectura web, que permite la construcción de aplicaciones y soluciones a través de esta tecnología.

2.1. Historia de la web

Ejercicio 5 [history_timeline, 20 pts]

Represente en una línea de tiempo los eventos más relevantes de la historia de la web que se presenta a continuación. Para cada evento agregue el año y una o unas palabras que refieran al evento. Dibuje la línea de tiempo en una imagen vectorial en formato *Scalable Vector Graphics* (SVG). Puede utilizar el software libre Inkscape, del cual existe documentación impresa y en vídeo, disponible incluso en forma grauita y en español como estos tutoriales básicos. Agregue su año de nacimiento a la línea de tiempo y resálelo con algún color. Guarde la imagen de la línea de tiempo en la carpeta `intro/history_timeline`.

La web fue conceptualizada por el informático británico **Tim Berners-Lee**, quien sigue siendo uno de sus líderes. Unió en 1989 su experiencia previa en hipertexto y en redes de computadoras, cuando trabajaba para la Organización Europea para la Investigación Nuclear (CERN, por sus siglas en francés), como una propuesta para comunicar documentos científicos de manera fácil a través de internet. Su propuesta fue acogida y a finales de 1990, Berners-Lee desarrolló:

- el *Protocolo de transferencia de hipertexto* {HTTP, *HyperText Transfer Protocol*};
- el primer servidor web llamado `CERN-httpd`;
- el primer navegador llamado `WorldWideWeb`, que también era un editor web;
- el *lenguaje de marcado de hipertexto* {HTML, *HyperText Markup Language*};
- y las primeras páginas web que hablaban sobre el proyecto mismo.

La aparición pública de la web fue en 1991, cuando Berners-Lee describió el proyecto en un grupo de noticias {*Usenet newsgroup*} invitando a su uso. Había nacido un mecanismo eficiente que permite a más personas ser autores, compartir material de interés al mundo, y hacer referencia al existente a través de hiper-enlaces.

En 1992, el CERN detuvo el apoyo al proyecto de Berners-Lee por no ser una actividad prioritaria de la organización. Berners-Lee migró su proyecto al Instituto Tecnológico de Massachusetts {MIT, *Massachusetts Institute of Technology*}. Dos años después, en 1994, Berners-Lee funda el **World Wide Web Consortium** (**W3C**) con sede en el MIT, apoyado por varias instituciones y empresas con el fin de

crear estándares y recomendaciones para mejorar la calidad de la web.

La difusión de la web fue lenta. Inicialmente fue adoptada por universidades y grupos científicos. Los documentos web eran muy básicos, los navegadores corrían en modo texto o en equipo de cómputo usado en los centros de investigación. Esta realidad cambió en 1993, cuando el *National Center for Supercomputing Applications* (NCSA) de la Universidad de Illinois en Urbana-Champaign (UIUC) introdujo el navegador gráfico **Mosaic**, cuyo nombre era alusivo a la diversidad de protocolos que soportaba.

Mosaic tenía amplio soporte multimedia, y presentaba imágenes entrelazadas con el texto, en lugar de ventanas separadas para ambos. Además corría en una diversidad de sistemas operativos lo que hizo fácil su adopción, y por tanto, colaboró significativamente en la popularidad de la web. NCSA licitaba el código fuente de *Mosaic* a otras compañías para que crearan sus propios navegadores, incluso comerciales.

Tras de graduarse el líder del proyecto *Mosaic*, Marc Andreessen, cofundó la compañía Netscape en 1994 para comercializar su navegador **Netscape Navigator** con mayores amenidades que *Mosaic*. Por ejemplo, desplegaba el contenido web conforme éste llegaba por las lentas redes sin hacer al usuario esperar a que se descargara por completo, era gratis para uso no comercial, y ofrecía esquemas de licenciamiento de código más atractivos. Netscape se convirtió en el navegador más usado del mundo durante la segunda mitad de los años 1990.

2.1.1. La primera guerra de navegadores

En 1995 Microsoft ofreció el navegador **Internet Explorer** 1.0 para Windows 95, influenciado por el código licitado de NCSA Mosaic. La versión 2.0 fue gratuita incluso para uso comercial. Lentamente Internet Explorer fue tomando popularidad y mercado de Netscape, lo que generaría la primera guerra de navegadores.

Los ataques entre estos dos navegadores consistían en la inclusión de funcionalidades novedosas para atraer tanto usuarios como autores de sitios web. Por ejemplo, Netscape desarrolló JavaScript a finales de 1995, que un año más tarde sería imitado por el JScript de Microsoft, en la versión 3 de Internet Explorer. En esa versión particular, Microsoft incluyó una parcial implementación de las hojas de estilo en cascada {CSS, *Cascading Style Sheets*} sugeridas por el W3C pero aún no estandarizadas.

En 1997 Netscape fusionó su Netscape Navigator con otro conjunto de programas para correo electrónico, composición web, calendario, entre otros. Al suite se le llamó **Netscape Communicator**, nombre que serviría para confusiones. Microsoft integró Internet Explorer 4 con Windows y desalentó desde el sistema operativo el uso de cualquier otro navegador. Netscape no compitió contra esto, ni tenía sentido. Desde 1998 el 80% del mercado que tenía Netscape pasó a sumar el 96% que tenía Internet Explorer 5 en el 2002. La guerra de los navegadores había finalizado y también el rápido tren de innovaciones, tan evidente que Microsoft no volvería a hacer cambios significativos en su navegador desde el 2001 al 2006.

A inicios de 1998 Netscape liberó el código fuente de Netscape Communicator 4.0 en el proyecto Mozilla. La comunidad descartaría luego dicho código debido a su complejidad, poca modularización, al gran volumen de pulgas y otros inconvenientes resultado de la carrera por la innovación. La Fundación Mozilla empezó la construcción de un nuevo navegador modularizado hecho desde cero, compuesto de un motor de navegación {*web browser engine*}, al cual se le llamó, Gecko, y sería el

motor de "renderizado" {*rendering*} de Firefox y el nuevo Netscape Communicator, que finalmente sería descontinuado a inicios del 2008.

2.1.2. El proceso de estandarización

La guerra de navegadores (aproximadamente de 1995 a 1998) también tuvo consecuencias muy negativas. Ambos, Netscape Navigator e Internet Explorer, ofrecían características novedosas incompatibles entre sí, es decir, dialectos de HTML distintos que provocaron que los autores de millones de páginas tuvieran que escoger por uno o por el otro. Era común ver imágenes indicando que el sitio se veía mejor con un navegador particular, incluso hasta de una versión específica.

Mientras tanto el proceso de estandarización avanzaba lentamente. En junio de 1994 Berners-Lee y otros autores proponen varios borradores de estandarización, y en noviembre de 1995 la *Internet Engineering Task Force* (IETF) aprueba el HTML 2.0 que luego se convertiría en estándar internacional en 1997. El W3C propone HTML 3.0 en abril de 1995 pero la IETF no realiza ninguna acción. Los navegadores tomarán luego ideas de estas iniciativas en proceso y las implementarían a su manera para atraer usuarios.

A inicios de 1997 la IETF traslada la responsabilidad de estandarización al W3C, quien publicaría recomendaciones con una eficiencia mayor. Se publica el HTML 3.2 adoptando etiquetas y atributos de marcado visual de Netscape (como `b` y `font`). A final del mismo año (1997), el W3C presenta el trabajo de estandarización más notorio hasta la fecha, conocido como HTML 4.0. Un esfuerzo que considera las extensiones derivadas de la guerra de navegadores y las raíces del HTML. En ella, las etiquetas de marcado visual serían desaprobadas {*deprecated*} en favor de CSS; pero su uso se había difundido tanto que el W3C resolvió por generar tres variantes de HTML 4.0:

- **HTML 4.0 Strict**. Prohíbe elementos desaprobados (*deprecated*).
- **HTML 4.0 Transitional**. Permite elementos desaprobados (*deprecated*).
- **HTML 4.0 Frameset**. Permite construir páginas basadas en marcos {*frames*}, con los elementos `frameset` y `frame`.

Dos años después, a finales de 1999, se le haría una ligera modificación al estándar HTML 4.01, cuya variación estricta (HTML 4.01 Strict) sería adoptado como estándar internacional (ISO/EIC 15445:2000), suplantando la versión 2.0 de 1997. Tras ello, el trabajo de estandarización dejaría de lado al HTML para concentrarse en el XHTML.

En febrero de 1998 el W3C publicó el estándar XML {*Extensible Markup Language*}. En enero de 2000 se reformuló HTML 4.01 como una aplicación XML en lo que se conoce como **XHTML** 1.0. Una actualización XHTML 1.1 se emitió como recomendación en mayo de 2001 que permite modularizar el HTML para necesidades específicas. Los módulos más destacables han sido XHTML-Basic que incluye el conjunto mínimo de características que cualquier navegador debe soportar, incluso de dispositivos móviles, y XHTML-MP {*mobile profile*} con controles aptos para dispositivos móviles. A partir de 2002, el W3C descartaría HTML y se concentraría exclusivamente en formular XHTML 2.0.

En este documento se usarán los términos HTML y XHTML para referirse a cada notación por separado, o (X)HTML cuando el contexto aplique a ambas por igual.

2.1.3. La segunda guerra de navegadores

Netscape Navigaror e Internet Explorer en la primera guerra de navegadores tuvieron equipos de desarrollo enfocados en agregar funcionalidad no estandarizada tan rápido como fuese posible, con poco control de calidad. Esto llevó a que ambos navegadores no se apegaran a los estándares, fuesen "pulgosos", e inseguros. Pese de hacerlo público, el código fuente de Netscape fue descartado por la Fundación Mozilla. Por su parte, Microsoft tras ganar la guerra mantuvo su navegador en el letargo por varios años.

Un navegador poco mencionado durante la primera guerra fue **Opera**. Inició en 1994 como un proyecto de investigación de la compañía noruega Telenor. Su primera versión disponible al público fue en 1996 en forma comercial, o gratuita con publicidad, modelo que podría explicar su poca popularidad. A partir de la versión 8.5 de 2005, se distribuye sin costo ni publicidad alguna. Se caracterizó por ser un navegador en apegarse a los estándares, influir activamente en ellos, y en la inclusión de características amigables para el usuario, como es el uso de pestañas (*tabs*), gestos de dispositivos de apuntar (*mouse gestures*), velocidad, seguridad y correr en dispositivos móviles.

La Fundación Mozilla adoptó las amenidades descritas para su navegador. Las primeras versiones aparecieron a finales del 2002, con nombres que después serían cambiados por conflictos con productos existentes, hasta quedar Firefox, el cual se liberó en noviembre de 2004. Desde el 2003 Mozilla Firefox empezó lentamente a atraer usuarios de Internet Explorer, en especial por sus problemas de seguridad y la falta de iniciativa de Microsoft por corregirlos.

En el 2003 Microsoft anunció que descontinuaría *Internet Explorer for Mac* que era el navegador por defecto en este sistema operativo. Apple reaccionó creando un navegador que lo reemplazara. A partir del motor KHTML usado en el navegador Konkeror para Linux, generó **WebKit**, y lo usó en su navegador Safari que apareció en el mismo año.

Microsoft reaccionó a estos nuevos navegadores hasta octubre de 2006 con Internet Explorer 7 (la versión 6 data de 2001), imitando características de Opera y Firefox. Esto no logró detener la creciente tasa de usuarios que seguían cambiando a Firefox. Por su parte, una semana después, Mozilla liberó Firefox 2.0 con mejoras de usabilidad y seguridad, lo que pondría en evidencia la segunda guerra de navegadores. Esta vez luchando por proveer mejor experiencia de usuario y apego a los estándares.

En el 2008 Google liberó el navegador Chrome basado en WebKit, que sobresalía por su velocidad y era el navegador por defecto en el sistema operativo más usado del mundo: Android. Chrome lentamente comenzó a ganar mercado de Internet Explorer y Firefox hasta convertirse en el navegador más popular a la fecha (ene-2020), con aproximadamente 65% del mercado, seguido por Safari (~17%), y Firefox (~5%), de acuerdo a StatCounter y NetMarketShare. En 2013 Google creó su propia adaptación del motor de "renderizado" WebKit llamada Blink.

Microsoft liberó la versión 8 de Internet Explorer en marzo de 2009 imitando funcionalidades de la competencia e incrementando el apego a los estándares. Microsoft seguió produciendo nuevas versiones aproximadamente cada dos años, hasta el 2015 en que descontinuó Internet Explorer en favor de un nuevo navegador **Microsoft Edge** construido inicialmente "desde cero" y luego sobre el motor Blink y otras funcionalidades creadas por Google.

La segunda guerra de navegadores se mantiene hasta el presente, donde cada fabricante libera versiones con frecuencia, trata de influir en los estándares (como se verán en el próximo apartado), e implementar la mayor parte de ellos antes de la competencia.

Ejercicio 6 [browser_profit, 5 pts]

Construir un navegador web implica destinar considerables recursos económicos, humanos, y tecnológicos. Sin embargo, la mayoría de navegadores web están disponibles de forma gratuita. ¿Qué explica que compañías inviertan tantos recursos en estos productos si son gratuitos? Indague y explique en dos o tres párrafos en un documento de Markdown o AsciiDoc. Recuerde incluir referencias a los documentos en que basó su respuesta. Nota: el buscador de Google debe estar involucrado en su respuesta.

2.1.4. HTML el estándar vivo (HTML5)

Como se indicó anteriormente, el proceso de estandarización por parte del Consorcio Web (W3C) dejó de lado HTML y se dedicó exclusivamente a XHTML 2.0 entre agosto de 2002 y julio de 2006. Sin embargo, el futuro de la web basado en XHTML 2.0 sólo alcanzó el nivel de notas y no de recomendación. Mientras este trabajo de estandarización ocurría en el W3C, un grupo de interesados consideraron inadecuada la orientación que había tomado el W3C hacia en la especificación formal de documentos con XHTML 2.0. Este grupo de interesados estaba compuesto principalmente por fabricantes de navegadores que seguían las prácticas de Opera (Mozilla Foundation, Opera Software, y Apple), y que se autodenominaron **WHATWG** (*Web Hypertext Application Technology Working Group)*.

El WHATWG formuló un borrador llamado *Web Applications 1.0*, que luego fue renombrado a (X)HTML5, y que no rompía la compatibilidad con la web existente, lo cual eximiría a millones de sitios web de tener que migrar al potencial XHTML 2.0. (X)HTML5 extendería a HTML 4 y XHTML 1.1 con nuevos elementos, documentaría formalmente el comportamiento de los navegadores ante código con errores, y tendría una orientación más enfocada al desarrollo de aplicaciones web con JavaScript que en la especificación de documentos con XML. En el 2008 el W3C resolvió por adoptar la propuesta (X)HTML5 del WHATWG como estándar web.

El WHATWG continuó trabajando en (X)HTML5 con la siguiente organización. Cualquier persona del mundo puede participar como "contribuidor" al unirse a la lista de correo y ofrecer sugerencias. Un grupo cerrado de "miembros" escogidos por invitación deciden los cambios en (X)HTML. Una persona escogida como "el editor" refleja los acuerdos en la especificación. El WHATWG decidió no usar números de versiones para identificar los cambios, sino que los cambios que haga el editor en el documento aprobados por los miembros, se consideran oficiales. De esta forma, el WHATWG encontró inapropiado el término (X)HTML5 por contener un número de versión, y lo renombró **HTML, el estándar vivo** (*HTML Living Standard*).

Por su parte el W3C continuó usando el término (X)HTML5 y trabajando con versiones. El W3C generó cuatro versiones: (X)HTML 5.0 de 2014 con un borrador del *HTML Living Standard* de 2007 (2007-2014), (X)HTML 5.1 (2012-2016), y (X)HTML 5.2 (2015-2017). El WHATWG considera estas versiones simplemente como instantáneas (*snapshots*) del *HTML Living Standard*. El W3C reconoció que tener dos estándares es trabajo redundante y el desfase es perjudicial para los desarrolladores. Por lo tanto, en mayo de 2019 elevó el *HTML Living Standard* como oficial y participa en su formulación junto con el WHATWG.

El *HTML Living Standard* es publicado por el WHATWG en su sitio web. Es un documento de más de 1200 páginas que abarca no sólo el lenguaje (X)HTML. También formaliza el modelo de objetos (DOM,

Document Object Model) y otras tecnologías disponibles a través de JavaScript ideadas para facilitar el desarrollo de aplicaciones web.

El apego a los estándares es una de las prácticas más importantes a adquirir por un desarrollador web. Si sus productos siguen los estándares, obtendrá un sinnúmero de beneficios, como los siguientes:

- Su sitio se presentará y comportará de forma más consistente entre navegadores, incluso de dispositivos móviles o del internet de las cosas.
- Su sitio será más accesible para personas con necesidades especiales.
- Su sitio y su estructura será mejor indexada por los buscadores web, como Google, lo que permitirá escalar en los resultados.
- Su prestigio será mayor ante los clientes, visitantes, y otros desarrolladores.

Dado que el *HTML Living Standard* es la especificación de uno de los estándares que conforman la tecnología web, se recomienda al desarrollador web a aprender a consultar este documento como una práctica habitual.

2.2. Arquitectura web

El objetivo de la web es la comunicación amigable, para las personas, de información (inicialmente en forma de documentos) a través de computadoras. Berners-Lee conceptualizó en 1989 cómo emplear los documentos digitales y las redes de computadoras para lograr este objetivo. A esta conceptualización se le llama **arquitectura web**, la cual mantiene a la fecha sus principios iniciales con pocas modificaciones, pero con numerosas extensiones.

La arquitectura web está fundamentada en el *paradigma de computación distribuida* donde los recursos y aplicaciones están alojados en sistemas distintos y estos se comunican a través de paso de mensajes. Los entes distribuidos realizan tareas concurrentes distintas siguiendo el modelo cliente-servidor. La Figura 19 diagrama en forma simplificada la arquitectura web inicial de Berners-Lee, compuesta de un *servidor web* que publica recursos (páginas web en HTML y otros archivos) identificados de forma única (URI). Las personas interesadas en esos recursos usan un navegador que solicita copias al servidor mediante el protocolo HTTP, y presentan los recursos recibidos de forma amigable.

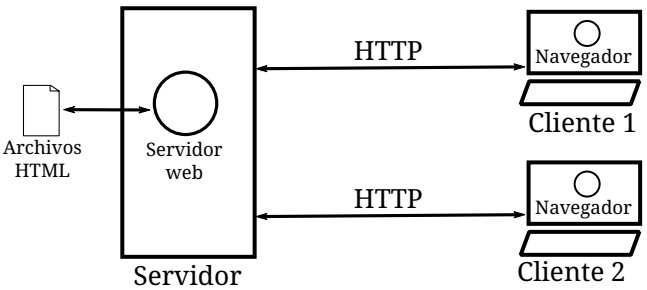


Figura 19. Arquitectura web simple

El autor en aras de publicar un sitio web, construye un conjunto de páginas web en notación (X)HTML

junto con imágenes, estilos y otros medios, y las almacena en una computadora que está conectada permanentemente a Internet, a la cual se le llama *el servidor*. Un software llamado **servidor web**, el cual tiene acceso de lectura a dichas páginas, estará en esta computadora escuchando por un puerto TCP {*Transmission Control Protocol*}, por defecto el puerto 80.

Un visitante interesado en los recursos del sitio, debe conocer la dirección de red del servidor, es decir su número IP, ya sea introduciéndolo directamente o a través del servicio de nombres de dominio {DNS, *Domain Name Service*} y el puerto donde el servidor web está escuchando. El visitante usa un software llamado **navegador web** {*web browser*} e ingresa en él la dirección del servidor y el puerto, empleando una notación estándar conocida como *localizador uniforme de recursos* (URL). El navegador intentará establecer una conexión TCP con el servidor al puerto indicado o al 80 si se omite. El servidor web aceptará la conexión. A partir de este momento el cliente y el servidor pueden comunicarse libremente, pero para que ambos puedan entenderse se necesita un "idioma" común llamado *protocolo de transferencia de hipertexto* (HTTP, HyperText Transfer Protocol).

El protocolo HTTP establece que el cliente, también conocido como *user agent*, siempre hace solicitudes de recursos para mostrarlos al usuario, y el servidor responde a ellas, no el recíproco. Las solicitudes del cliente y las respuestas del servidor están codificadas como se verá luego.

La arquitectura web descrita se mantiene vigente. Sin embargo, su evolución ha procurado facilitar la programación de *aplicaciones* que se encargan de generar automáticamente los documentos en el lado del servidor, y de *aplicaciones* que mejoran la experiencia del usuario del lado del cliente. A este tipo de programas se le conoce como **aplicación web**. La web ha sido útil no sólo para comunicar personas. Una aplicación que está diseñada para que su salida sea consumida por otros programas y no por personas se conoce **servicio web**. En lo que resta de esta sección se explicarán los conceptos que componen la arquitectura web con más detalle.

2.2.1. El servidor web

Un **servidor web** {*web server*} o **servidor HTTP** {*HTTP server*} es un programa cuya ejecución es persistente, a veces llamado servicio o demonio, en un equipo conectado a Internet, esperando conexiones de clientes usualmente por el puerto 80. Una vez establecida una conexión, el cliente solicita repetitivamente recursos al servidor mediante el protocolo HTTP y éste responde a cada una de ellas siguiendo los lineamientos del mismo protocolo.

El término *servidor web* refiere al software y no al hardware donde éste corre, dado que el paradigma de computación distribuida permite que tanto un servidor como un cliente puedan ejecutarse en el mismo o diferente hardware. Hay al menos cuatro formas de ofrecer recursos a través de la web, ordenados en forma descendente por control, y ascendente por facilidad y rapidez de uso.

1. Implementar un servidor web escribiendo un programa en un lenguaje de propósito general. Permite el mayor grado de control y eficiencia. Un ejemplo minimalista en C es Pico HTTP Server.
2. Implementar un servidor web usando bibliotecas de software {*web frameworks*}. Son ejemplos: ASP.NET, Spring (Java), Node.js, Django (Python), Ruby on Rails, y Wt (C++). La Wikipedia provee una lista comparativa de bibliotecas disponibles. Este material usa Node.js en el capítulo 7.
3. Utilizar un servidor web de propósito general. Estos programas se instalan y configuran para servir sitios web estáticos compuestos de páginas (X)HTML. Además cuentan con extensiones que facilitan la programación de sitios web dinámicos. La Tabla 8 lista los servidores web más

populares a setiembre de 2019 usados en sitios web públicos de acuerdo a Netcraft.

4. Utilizar un **sistema de gestión de contenidos** {CMS, *content management system*}, que es un software que facilita a los usuarios la creación y mantenimiento de sitios web dinámicos. Los CMS son programas que permiten a los usuarios editar en línea el contenido mismo del sitio de forma amigable, sin conocimiento de programación de computadoras. El contenido típicamente es almacenado en bases de datos relacionales o documentales. Son ejemplos: WordPress, Wix, Squarespace, Joomla!, y Drupal. La oferta de gestores de contenido es numerosa, como puede estudiarse en la Wikipedia.

Ejercicio 7 [web_server_selection, 5 pts]

Una empresa con bastantes años en el mercado pero sin presencia en internet le contrata para crear su sitio web. ¿Qué preguntas haría al personal de la empresa que le ayuden a usted a escoger cuál de los cuatro tipos de servidores web implementar? Considere que el personal no tiene experiencia en programación de computadoras.

Idee escenarios para sus preguntas. Los escenarios pueden tener esta estructura: "si las respuestas del personal fueran a y b, entonces el servidor recomendable sería w por esta razón: x". Escriba sus preguntas y escenarios en un documento Markdown o AsciiDoc en la carpeta `intro/web_server_selection/`.

Tabla 8. Servidores web de propósito general más populares según Netcraft, 2019.

Nombre	Fabricante	Licencia	Detalles
nginx	Igor Sysoev	Libre (BSD)	Nació como una alternativa de Apache. Se caracteriza por el alto rendimiento y bajo consumo de recursos. Corre en la mayoría de sistemas operativos. Sirve aproximadamente un tercio de los sitios web del mundo.
Apache HTTP Server	Apache	Libre (Apache License)	Rico en características y extensiones. Corre en la mayoría de sistemas operativos. Históricamente ha sido el servidor web más usado del mundo, pero ha perdido mercado por el incremento de popularidad de nginx. Sirve poco menos de un tercio de los sitios web del mundo.
Internet Information Services (IIS)	Microsoft	Propietario, comercial	Sólo se ejecuta en Windows Server. Sirve cerca de un 15% de los sitios web del mundo.
Google Web Server (GWS)	Google	Uso interno	Sirve aproximadamente 3% de los sitios web del mundo.
lighttpd	lighttpd	Libre (BSD)	Diseñado para ambientes de muy alto rendimiento donde la velocidad es un factor crítico. Es limitado en cuanto a funcionalidad. Sirve aproximadamente un 1% de las páginas web del mundo.

Ejercicio 8 [local_dev_env, 10 pts]

Escoja e instale un servidor web en su máquina local. Configúrelo para crear un servidor virtual que escuche en algún puerto, idealmente el 80. Establezca su repositorio de control de versiones local como la ubicación del sitio web (llamado *document root* en terminología de Apache). Consulte la documentación de su servidor web, o tutoriales en línea, para ayudarse en la configuración de su servidor web. Escriba en un archivo de texto un párrafo indicando la razón de elección del servidor web.

Verifique que pueda recorrer el repositorio local utilizando un navegador, y por tanto en http://localhost/. Su repositorio local y un servidor web local que le permite experimentar los cambios que haga en el código fuente, constituyen su **ambiente de desarrollo local**. Este ambiente es sumamente importante, porque permite al desarrollador implementar y probar sin afectar el sitio web en producción (el que atiende los visitantes reales).

Como prueba de su *ambiente de desarrollo local* haga lo siguiente. Cargue la dirección http://localhost/ en su navegador. Redimensione la ventana del navegador a un tamaño cercano a los 640x480 píxeles. Tome una captura de pantalla del navegador. Edite la imagen para que ocupe un tamaño reducido (unos 64kB o menos). Sugerencia, con Gimp puede cambiar la paleta de una imagen PNG a 256 colores (menú `Image | Mode | Indexed`). Agregue esta imagen a su repositorio de control de versiones.

2.2.2. El cliente web o navegador web

Un **navegador web** {*web browser*} o **cliente web** {*web client*} es un software que permite obtener, presentar, recorrer, e interaccionar con recursos disponibles en la web. De los componentes de la arquitectura web, es el más conocido por los usuarios ya que interaccionan directamente con él. Un navegador web es una pieza de software compleja, por lo que usualmente está dividida en al menos dos módulos, el motor y la interfaz.

- El **motor del navegador web** {*web browser engine* o *web rendering engine*} se encarga de obtener los recursos, analizar el contenido en (X)HTML, incrustar medios (imágenes, audio, vídeo), aplicar estilos, ejecutar código JavaScript, y presentar los resultados en la pantalla u otro medio.
- La **interfaz del navegador web** provee una barra de direcciones, marcadores, botones de navegación, y otros controles que usan al *motor del navegador web* internamente para facilitar al usuario su interacción.

La modularización entre el motor y la interfaz tiene importantes ventajas. El motor de navegación web se distribuye en forma de biblioteca y permite que otras aplicaciones de software puedan reusarlo. De esta forma, clientes de correo, aplicaciones de ayuda, u otro software que lo necesite, puede manipular documentos web. La Tabla 9 muestra los motores de navegación web más populares al año 2020 y ejemplos de navegadores que los usan.

Tabla 9. Motores de navegador web populares

Nombre	Fabricante	Licencia	Navegadores	Notas
Gecko	Mozilla Project	Mozilla Public License	Firefox	El más longevo de los activos (1997)

Nombre	Fabricante	Licencia	Navegadores	Notas
WebKit	Apple, KDE…	Libre (LGPL/BSD)	Safari	Es una adaptación de KHTML
Blink	Google	Libre (LGPL/BSD)	Chrome, Opera 15+, Vivaldi, Edge 2020	Es una adaptación de WebKit
EdgeHTML	Microsoft	Propietario	Microsoft Edge	Es una adaptación de Trident
Trident	Microsoft	Propietario	Internet Explorer	Descontinuado
Presto	Opera Software	Propietario	Opera	Descontinuado

El navegador web es un tipo de agente de usuario. En terminología web, un **agente de usuario** {*user agent*} es cualquier software del lado del cliente que consume recursos web, normalmente ideados para humanos. El término "agente de usuario" proviene del escenario más común, en que estos programas reciben los recursos web y los presentan a las personas, como es el caso de los navegadores. Sin embargo, los agentes de usuario no están restringidos a este escenario. Los siguientes son otros ejemplos de agentes usuario y su propósito:

- Buscadores (como Google o Bing) que indexan el contenido web.
- Validadores web que proveen veredictos sobre la corrección del código.
- Navegadores de voz {*voice browser*} que leen el contenido web y producen salida en parlantes.
- Aplicaciones móviles que facilitan el acceso a ciertos sitios web.
- Agregadores de noticias {*news aggregator*} que recopilan cambios que ocurren en sitios web y otras fuentes de información.
- Electrodomésticos inteligentes y dispositivos del internet de las cosas, que se comunican a través de la web con propósitos específicos.

Ejercicio 9 [web_engine_lang, 10 pts]

Para cada uno de los motores web de la Tabla 9, incluso los descontinuados, averigüe:

- El lenguaje de programación en que está desarrollado.
- El nombre del motor de JavaScript.
- La suma del porcentaje de mercado de sus navegadores acuerdo a un estudio recientes (por ejemplo, StatCounter o NetMarketShare). El porcentaje debe incluir tanto computadoras de escritorio como dispositivos móviles.

Elabore un gráfico (de barras o de pastel) en formato SVG que incluya la información anterior. Sugerencia: indague sobre generadores de gráficos SVG en línea {*SVG chart generators*}.

Ejercicio 10 [web_engine_preference, 5 pts]

¿Cuál motor de navegación debe preferir usted como desarrollador? Es decir, ¿cuál navegador usar para probar sus sitios y aplicaciones web, de tal forma que usted esté seguro que con este navegador su sitio se ve bien y sus aplicaciones se comportan de forma esperada? Sugerencia: reflexione cuidadosamente antes de responder esta pregunta.

2.2.3. El identificador uniforme de recursos URI

Un servidor web provee recursos en los que puede estar interesado un cliente. Todo recurso transferible entre el servidor y el cliente debe estar identificado de forma única en el mundo mediante un texto (*string*) llamado **identificador uniforme de recurso** {URI, *Uniform Resource Identifier*}. Los URI fueron definidos por Berners-Lee junto con el Grupo de Trabajo de Ingeniería de Internet {IETF, *Internet Engineering Task Force*} en el RFC 1630 de 1994.

Los acuerdos tomados en las reuniones del IETF son documentadas en archivos de texto que mantienen el nombre histórico de **RFC** {*Request for Comments*}. Cada RFC tiene un número y título único, y una vez publicado es inmodificable. Un nuevo RFC puede complementar o sustituir a uno previo, de esta forma, los RFC forman una base de documentos que reflejan los acuerdos históricos tomados por el grupo. Algunos RFC se convierten en normativa, y por tanto, son los documentos oficiales de estándares en Internet, lo que incluye a una parte de la web.

Los URI son extensibles. Berners-Lee y el IETF definieron dos tipos de URI:

- **Nombre uniforme de recurso** {URN, *Uniform Resource Name*}, definido inicialmente en el RFC 1737 de 1994 y modificado en posteriores (por ejemplo RFC 8141 de 2017), provee un nombre único en el mundo para el recurso, sin indicar dónde está ubicado. Tiene la ventaja de que el recurso se puede mover y su nombre se mantendría inalterado, lo que disminuye el fenómeno de "enlace roto". Sin embargo, el mecanismo para acceder al recurso a través del nombre, conocido como *resolución de nombre*, es aún un asunto de discusión, lo que explica la carencia de uso de los URN. La resolución de nombres necesita de entidades registrales que mapeen los nombres a sus ubicaciones. Un ejemplo de estas entidades es el *Identificador de objeto digital* {DOI, *Digital Object Identifier*} que mapea documentos académicos y gubernamentales usando un principio similar al URN.
- **Localizador uniforme de recurso** {URL, *Uniform Resource Locator*}, definido inicialmente en el RFC 1738 de 1994 y modificado en posteriores (por ejemplo RFC 3986 de 2005), es una dirección junto con alguna información adicional para acceder al recurso. Depende de la ubicación lógica del recurso, por tanto, si éste se mueve, el URL dejará de ser válido, estado que se conoce como *enlace roto* {*broken link*} o *enlace muerto* {*dead link*}. Los URL existen desde el inicio de la web y antes de que fueran estandarizados en el RFC 1738 (1994), por lo que su uso es ubicuo.

La sintaxis de los URN y los URL es compatible, por lo que se pueden considerar la misma notación. En el RFC 3986 de 2005, Berners-Lee y el IETF decidieron descontinuar el uso de los términos URN y URL, a favor de URI. En más de 15 años, esta práctica no ha sido adoptada mundialmente, que continúa empleando el popular término URL.

El RFC 3986 especifica formalmente la sintaxis de un URI usando la notación aumentada de Backus-Naur {ABNF, *Augmented Backus-Naur Form*}. Por su parte, este material usará la notación informal del Listado 1 con el fin de facilitar su comprensión.

Listado 1. Sintaxis informal de un URI

```
scheme://username:password@server:port/path?query_string#fragment
```

Ejercicio 11 [uri_components, 5 pts]

Identifique y señale en los siguientes ejemplos hipotéticos, cada una de los componentes del URI. Sugerencia: puede ayudarse con la lista de componentes presente más adelante en el texto.

```
http://www.amazon.com/
http://www.w3.org/TR/html401/struct/tables.html#edef-CAPTION
http://acme.co.cr:8080/index.php
https://24.168.39.221/cgi-bin/book?id=4596098&action=remove
ftp://msoto:w33n8rf1@down.antivirus.net/current/setup.exe
```

Escriba en un documento de texto sus resultados. Para cada URI, separe sus componentes usando la siguiente estructura:

```
URI       :
Esquema   :
Usuario   :
Contraseña:
Servidor  :
Puerto    :
Ruta      :
Parámetros:
Fragmento :
```

La mayoría de componentes de un URI son opcionales. Su obligatoriedad depende del esquema usado para acceder al recurso. Los componentes de un URI son los siguientes.

1. **Esquema** {*scheme*}, también llamado **protocolo** {*protocol*}. Indica el propósito y la sintaxis del resto del URI. La cantidad de esquemas puede incrementar en el tiempo, lo cual se regula con nuevos RFC. Algunos esquemas populares son `http`, `https`, `ftp` y `mailto`. Por ejemplo, al procesar el URI **`http:`**`//www.amazon.com/`, un agente de usuario hará un petición HTTP al servidor `www.amazon.com` en el puerto 80. Al procesar el URI **`mailto:`**`jeisson.hidalgo@ucr.ac.cr`, lanzará un cliente de correo para escribir un mensaje dirigido a `jeisson.hidalgo@ucr.ac.cr`.
2. **Servidor**. Es la *dirección IP* o el *nombre de dominio* si se tiene registrado uno en el *servicio de nombres de dominio* {DNS, *Domain Name Service*}. Permite identificar al servidor que provee el recurso, por ejemplo `drupal.org`. Dado a que DNS no es sensitivo a mayúsculas ni minúsculas,

tiene el mismo efecto acceder a `DRUPAL.ORG` o `Drupal.org`.

3. **Puerto**. Indica el puerto que será usado en la conexión TCP. Por ejemplo, `http://myserver.com:10000/` hará que el navegador establezca una conexión HTTP en el puerto 10000, probablemente para administración del servidor. Si se omite el puerto en el URI se asumirá el por defecto para el *esquema*. Por ejemplo, para `http` es 80, para `https` es 443 y para `ftp` es 21. La lista de protocolos y puertos estándar es administrada por la *Internet Assigned Numbers Authority* (IANA).
4. **Ruta**. Contiene la ruta para encontrar el recurso dentro del servidor. Puede ser relativa al sistema de archivos del servidor o un "alias" que ayude a especificar el recurso, por ejemplo, `https://www.rfc-editor.org/rfc/rfc3986.txt`. Es muy probable que sea sensitiva a mayúsculas si el sistema de archivos del servidor lo es también.
5. **Parámetros**. Si el recurso es generado por una aplicación en el servidor web, a esta se le pueden enviar parámetros en forma de parejas `parametro=valor` y separadas por el signo `&` si son varias parejas, por ejemplo, `http://localhost/cms.php?action=admin&lang=es&find=theme+colors`. A esta lista de parámetros se le suele llamar el *texto de consulta* {*query string*}.
6. **Identificador de fragmento**. Es un nombre que sirve para identificar una parte (fragmento) o un punto particular de un documento web. Cuando se especifica un *identificador de fragmento* en un URI, provoca que el agente de usuario cargue el documento y automáticamente se desplace {*scroll*} hasta el fragmento con el identificador dado. Por ejemplo, `https://tools.ietf.org/html/rfc3986#section-3.5`.
7. **Usuario y contraseña**. Si para acceder al recurso se necesita que el visitante se autentique, algunos protocolos permiten indicar las credenciales en el URI mismo. Es poco común y una práctica no recomendada incluir contraseñas, ya que el URI normalmente es público y carece de seguridad. Ejemplo, `https://jeissonh@bitbucket.org/jeissonh/repo.git`.

Ejercicio 12 [uri_encoding, 5 pts]

Los siguientes enlaces no son válidos en ASCII. Haga manualmente el *mínimo* de modificaciones necesarias para que se conviertan en URI ASCII válidos. Revise la sección 2 del RFC 3986 y haga la codificación en un editor de texto sin usar ninguna otra herramienta.

```
http://localhost/f.php?code="f(a,n,i,s){return i=n?s:f(a,n,i+1,s-a[i]%n);}"
https://filo.net/files/Lingüística 2ed por A. Zúñiga.pdf
```

Tras completar sus conversiones manuales, compárelas con el resultado de conversiones automáticas. Compare al menos con un navegador web (pegue el enlace original en la barra de direcciones y presione `Enter`), y con un codificador/decodificador URI disponible en línea. ¿Coinciden sus codificaciones con las automáticas? ¿Son correctas las codificaciones automáticas que obtuvo?

Los RFC son la documentación oficial de la arquitectura web. Sin embargo, son documentos dirigidos a fabricantes de servidores web, agentes de usuario, y tecnologías afines. Aunque la persona desarrolladora de aplicaciones web se enriquece de su lectura, existen recursos alternativos más orientados a esta audiencia. Un ejemplo es el proyecto *Mozilla Developer Network (MDN) Web Docs*, cuyo objetivo de la comunidad Mozilla es documentar los acuerdos tomados por el grupo WHATWG. Paulatinamente el sitio *MDN Web Docs* se ha ido convirtiendo en la referencia de facto para desarrolladores web.

El proyecto incluye también documentación sobre la arquitectura web proveniente de los RFC. Por ejemplo, la entrada correspondiente a esta sección se llama "Recursos y URI". El sitio web de *MDN Web Docs* está estructurado de forma amigable para la consulta de desarrolladores. Incluso cuenta con un curso gratuito en desarrollo web traducido a varios idiomas.

2.2.4. El protocolo de transferencia de hipertexto HTTP

La arquitectura web establece que un servidor web provee al mundo recursos identificados de forma única con textos URI. Un agente de usuario interesado solicita estos recursos y el servidor responde con ellos a través de paso de mensajes. Las convenciones usadas en el trasiego de estos recursos entre el cliente y servidor las establece el **protocolo de transferencia de hipertexto** {HTTP, *Hypertext Transfer Protocol*}.

Cuando el agente de usuario necesita un recurso, emite un mensaje de **solicitud HTTP** {*HTTP request*} al servidor. El servidor web carga el recurso desde el disco, una base de datos, la salida de un programa, u otra otra fuente, y responde con una **respuesta HTTP** {*HTTP response*}.

Una **sesión HTTP** {*HTTP session*} es una secuencia de solicitudes-respuestas entre el agente de usuario y el servidor web. En la versión HTTP/1.0 de 1996 se establece una sesión por cada transferencia. Es decir, se inicia la sesión, el cliente solicita un recurso, el servidor responde y se cierra la sesión inmediatamente. Para transferir un nuevo recurso se debe iniciar otra sesión HTTP. En la versión HTTP/1.1 de 1999 (RFC 2616) la sesión permite un número arbitrario de transferencias lo que agiliza la comunicación entre ambas partes. A partir de 2014 el protocolo HTTP/1.1 es regido por la familia de RFC de la Tabla 10.

Tabla 10. Familia de especificaciones HTTP/1.1 de 2014

RFC	Título	Notas
7230	*Message Syntax and Routing*	Introduce el protocolo, su arquitectura, los esquemas `http` y `https`, la sintaxis de sus mensajes, consideraciones de red, y otros formalismos.
7231	*Semantics and Content*	Especifica cómo los desarrolladores deben ensamblar las solicitudes HTTP para lograr un propósito específico, y cómo deben responder los servidores web apropiadamente.
7232	*Conditional Requests*	Especifica solicitudes HTTP que contienen una condición, y si se evalúa como verdadera por el servidor web, el recurso solicitado es enviado al cliente. Es útil, por ejemplo, para saber si un recurso se ha actualizado en el servidor y la versión en el caché del agente de usuario está o no actualizada.

RFC	Título	Notas
7233	*Range Requests*	Especifica cómo los clientes pueden solicitar trozos de un recurso, en lugar del recurso completo. Es útil para continuar una descarga tras una desconexión o para dispositivos con baja conectividad.
7234	*Caching*	Especifica el uso de memoria caché en la red (en el cliente, el servidor, o nodos intermedios) para almacenar copias de los recursos con el fin de acelerar el acceso a los mismos.
7235	*Authentication*	Permite a un servidor indicar cuáles recursos necesitan autenticación para ser accedidos, y los mecanismos en que los clientes pueden proveer las credenciales.

2.2.4.1. Los mensajes HTTP

La Figura 20 muestra un mensaje de solicitud HTTP/1.1 real y su correspondiente respuesta. La estructura de ambos mensajes está resaltada con rectángulos, y se puede ver que es la misma: una línea inicial, varias líneas de encabezado, una línea vacía que sirve de separador, y el cuerpo del mensaje. El RFC 7230 llama a estas partes *secciones del mensaje*. Ambos tipos de mensajes tienen las mismas cuatro secciones, pero varían sólo en el contenido de las primeras dos. Las primeras tres secciones de ambos mensajes son de texto en HTTP/1.1, y los cambios de línea son los separadores que sirven para distinguirlas, por eso se resaltan en la figura. En los siguientes apartados se explica cada una de las secciones que conforman los *mensajes HTTP*.

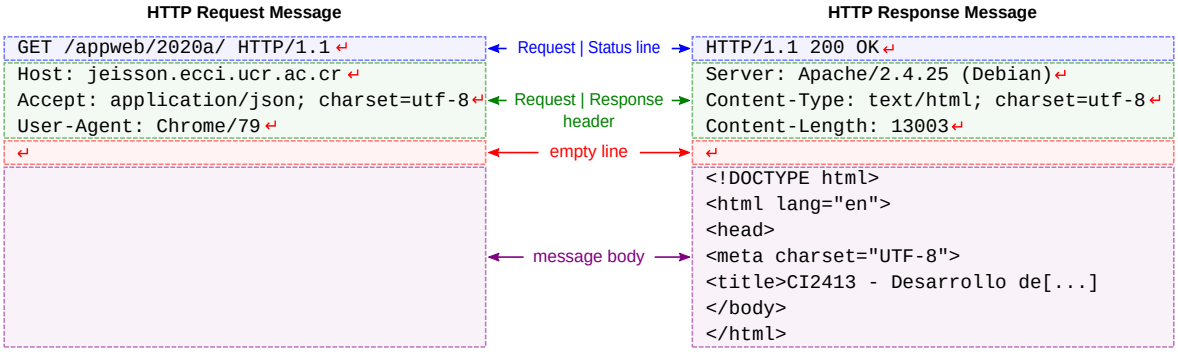


Figura 20. Mensajes de solicitud y respuesta HTTP

Ejercicio 13 [http_messages_for_resource, 5 pts]

Usando un navegador web, extraiga el mensaje de solicitud a un archivo `http_request.txt` y el mensaje de respuesta a un archivo `http_response.txt` de un recurso web. Puede usar la página del curso como prueba. Mantenga los cambios de línea intactos en los archivos de resultado. No es necesario incluir el cuerpo de los mensajes. Es probable que el navegador realice varias solicitudes y reciba varias respuestas. En tal caso, cree varios archivos numerados de la forma `http_request_N.txt` y `http_response_N.txt`, donde `N` es el número de archivo en el orden en que se solicitan los recursos.

Nota. Los navegadores sólo muestran el contenido del cuerpo de los mensajes al usuario, y ocultan las tres primeras secciones de los mensajes usados en la negociación con el servidor web. Algunos muestran esta información formateada en las herramientas del desarrollador. Para ver los mensajes en texto puro puede usarse la extensión Live HTTP Headers para Google Chrome y Mozilla Firefox. Esta extensión permite al usuario visualizar y estudiar las secciones ocultas de los mensajes que se intercambiaron entre el cliente y el servidor durante la carga de un recurso.

Ejercicio 14 [http_message_structure, 10 pts]

Dibuje en un programa de ilustración o en una hoja de papel los ejemplos de mensajes HTTP que obtuvo en el Ejercicio 13 [http_messages_for_resource, 5 pts]. Puede descargar y modificar la Figura 20.

Haga anotaciones sobre el dibujo indicando lo que significan cada una de sus líneas, con la intención de que le ayudarán en el futuro como referencia. Los siguientes apartados explican el contenido de cada una de sus partes. Si lo prefiere puede anotar la imagen con números, incluir la imagen en un archivo de Markdown, y explicar las líneas en el documento.

Si realizó el dibujo en papel, escanee o tome una fotografía y agréguela al repositorio de control de versiones. La imagen no debe superar los 200kB.

2.2.4.2. La línea de solicitud

La primera sección del mensaje de solicitud HTTP se llama la **línea de solicitud HTTP** {*HTTP request line*}. En esta línea el cliente web indica al servidor el recurso de interés identificado con un URI, y la acción (método) que desea el servidor tome con dicho recurso. El Listado 2 muestra la sintaxis de la línea de solicitud HTTP. La Figura 20 muestra un ejemplo del contenido de esta línea.

Listado 2. Sintaxis de la línea de solicitud HTTP

```
METHOD URI HTTP_version
```

El protocolo HTTP define 9 acciones que un cliente web puede solicitar al servidor, reciben el nombre de **métodos de solicitud** {*request methods*} y se listan en la Tabla 11. El protocolo HTTP indica que todo servidor web debe al menos implementar GET y HEAD, e idealmente OPTIONS.

Tabla 11. Métodos de solicitud de recursos HTTP

Método	Descripción
`GET`	Solicita una copia del recurso cuya identificación esta dada por el URI que le sigue. Por defecto, se solicita una copia completa del recurso. Sin embargo, a través de directivas en el *encabezado* (la sección que sigue a la línea de solicitud) se pueden hacer solicitudes condicionales o parciales. Una **solicitud condicional** {*conditional GET*} permite al cliente obtener una copia del recurso sólo si ha cambiado en el servidor a partir de una fecha dada. Una **solicitud parcial** {*partial GET*} permite al cliente obtener sólo un segmento o trozo del recurso. Estos detalles se explican en el RFC 7232.
`HEAD`	Es idéntico al método GET excepto en que el servidor web no debe retornar un recurso en el cuerpo del mensaje {*message body*} en el mensaje de respuesta HTTP. Esto permite al cliente obtener metadatos de un recurso, como para saber si ha cambiado en el servidor, o para otros fines. Es muy usado por motores de búsqueda como Google o Bing.
`OPTIONS`	Retorna los métodos HTTP que el servidor web soporta para un recurso dado (con un URI) o en general por el servidor web mismo (con un asterisco (`*`) en lugar de un URI).
`POST`	Envía datos, normalmente ingresados en un formulario web, en el cuerpo del mensaje de solicitud HTTP {*message body*}. El servidor web pasa los datos al recurso identificado por el URI, normalmente un programa que procesará o almacenará dichos datos.
`PUT`	Sube o cuelga {*upload*} un recurso identificado por el URI, y cuyo contenido es enviado en el cuerpo del mensaje de solicitud HTTP {*message body*} al servidor, el cual reemplaza el recurso anterior si existe. Se diferencia de POST en la semántica del URI. En `POST` el URI especifica una aplicación que recibe los datos y los procesa, mientras que en `PUT` el URI es la identificación del recurso mismo.
`PATCH`	Aplica modificaciones parciales a un recurso identificado por el URI. Fue agregada en el RFC 5789. El cuerpo del mensaje contiene un documento que indica los cambios a realizar en el recurso, cuya sintaxis no es estándar, sino dependiente de la aplicación que realiza las adaptaciones en el recurso.
`DELETE`	Solicita al servidor eliminar el recurso identificado por el URI. Normalmente está deshabilitado por defecto en la mayoría de servidores web.
`TRACE`	Permite rastrear servidores intermedios {*proxy*} que intervienen en el procesamiento de la solicitud HTTP, en especial si implementan algún caché.
`CONNECT`	Transforma la solicitud en una conexión a un túnel TCP/IP. Se usa, por ejemplo, para cambiar de una conexión HTTP a una comunicación segura HTTPS.

2.2.4.3. Encabezado de solicitud

La sección de **encabezado de la solicitud HTTP** {*HTTP request header*} permite al cliente proveer información adicional sobre la petición al servidor. En esta sección se escriben parejas `Campo: Valor` llamadas *campos del encabezado HTTP* por el RFC 7231, donde las mayúsculas y minúsculas se tratan igual {*case insensitive*}.

Los campos del encabezado son extensibles, es decir, se pueden crear nuevos en el tiempo, mientras haya un acuerdo entre los clientes y servidores. La IANA es la entidad encargada del registro de los campos estándar y provisionales. Por su parte, la familia de RFC 7230 define los 21 campos del

encabezado de la Tabla 12, de los cuales sólo `Host` es obligatorio.

Tabla 12. Campos estándar en el HTTP Request header

Categoría	Campo	Descripción
Controles	`Cache-Control`	5.2 of [RFC7234]. FOWD.58
	`Expect`	Permite al cliente web especificar expectativas que necesitan ser soportadas por el servidor web para que éste sea capaz de realizar la solicitud adecuadamente. La única expectativa definida por el RFC 7231 es `Expect: 100-continue`, establecida por los clientes que van a enviar un cuerpo de mensaje extremadamente largo.
	`Host`	5.4 of [RFC7230]. Especifica el servidor y el puerto que tiene el recurso solicitado, ya que la primera línea (Request line) no incluye estos valores y se necesitan para desambiguar en caso de servidores proxy u otros intermediarios. Permite a un servidor web (con un mismo IP) servir varios sitios web con diferentes nombres de dominios (servidores virtuales). Es el único atributo obligatorio en HTTP/1.1. Ej.: `Host: www.miservidor.com:8080`.
	`Max-Forwards`	Usado en los métodos `TRACE` y `OPTIONS` para indicar el máximo número de veces que una solicitud puede ser reenviada por servidores intermediarios {*proxy servers*}.
	`Pragma`	5.4 of [RFC7234]
	`Range`	3.1 of [RFC7233]
	`TE`	4.3 of [RFC7230]
Condicionales	`If-Match`	3.1 of [RFC7232]
	`If-None-Match`	3.2 of [RFC7232]
	`If-Modified-Since`	3.3 of [RFC7232]
	`If-Unmodified-Since`	3.4 of [RFC7232]
	`If-Range`	3.2 of [RFC7233]
Negociación de contenido	`Accept`	Indica los tipos de contenido (MIME-types) que el cliente puede aceptar, como una lista de parejas `tipo/subtipo`. Un asterisco indica cualquier tipo o subtipo es aceptado. Por ejemplo `Accept: text/*` indica que el cliente espera una respuesta que contenga cualquier tipo de texto en el cuerpo del mensaje.
	`Accept-Charset`	5.3.3
	`Accept-Encoding`	5.3.4. FWD.58
	`Accept-Language`	5.3.5

Categoría	Campo	Descripción
Autenticación	`Authorization`	4.2 of [RFC7235]
	`Proxy-Authorization`	4.4 of [RFC7235]
Solicitud de contenido	`From`	5.5.1
	`Referer`	5.5.2. Permite especificar el URI de otro recurso, por el cual se obtuvo el URI del recurso solicitado al servidor. Debería ser `Referrer:` pero en el estándar se escribió con un error ortográfico.
	`User-Agent`	5.5.3. Permite conocer el sistema operativo y el agente de usuario que hace la solicitud. Se pueden aprovechar para personalizar el sitio al agente de usuario y para almacenar/registrar estadísticas. Consta de varias parejas `módulo/valor` separadas por espacios y en orden de importancia. Se recomienda a todos los agentes de usuario proveer este atributo. Ej.: `User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.6; rv:5.0.1) Gecko/20100101 Firefox/5.0.1`.

2.2.4.4. La línea vacía

Tanto el encabezado de solicitud como el de respuesta constan de una cantidad arbitraria de líneas. Para poder determinar cuándo el encabezado termina y cuándo comienza el cuerpo del mensaje, se usa una línea vacía.

La **línea vacía del mensaje HTTP** (*HTTP empty line*) consta de un cambio de línea en formato del sistema operativo MS-DOS. Es decir, de los caracteres retorno de carro y avance de línea, comúnmente expresados como CRLF (*carriage-return* y *line-feed*). En notación C se escribe como `"\r\n"`. Aunque algunos servidores web aceptan un LF simple (`"\n"`). Esta línea no debe tener ningún otro espacio en blanco.

2.2.4.5. El cuerpo del mensaje

El **cuerpo del mensaje HTTP** {*HTTP message body*} es la única sección opcional del mensaje HTTP y su contenido varía dependiendo de la naturaleza del mensaje. En HTTP/1.1 el cuerpo del mensaje es la única sección que puede transportar información binaria, ya que todas las demás son de texto.

2.2.4.6. La línea de estado

En la **línea de estado HTTP** {*HTTP status line*} del mensaje de respuesta HTTP, el servidor indica al cliente web un código y una explicación textual del resultado de procesar la solicitud previamente hecha. El Listado 3 muestra la sintaxis de la línea de estado. Un ejemplo se encuentra en la Figura 20.

Listado 3. Sintaxis de la línea de estado HTTP

```
HTTP_version status_code reason_phrase
```

El servidor web indica la versión de HTTP (`HTTP_version`) con la que responde al cliente. Indica un código numérico estándar (`status_code`) que resume la respuesta. Finalmente provee un texto corto (`reason_phrase`) escogido por el servidor web y comprensible para seres humanos que el cliente web no necesita analizar. El `status_code` es entero de tres dígitos, donde el primer dígito indica una de las cinco *clases de respuesta* definidas por el estándar HTTP:

1. *Información*. La solicitud fue recibida y está siendo procesada.
2. *Éxito*. La solicitud fue recibida, entendida, aceptada y procesada exitosamente.
3. *Redirección*. Se necesitan más acciones del cliente para completar la solicitud.
4. *Error del cliente*. La solicitud es incorrecta por un error de sintaxis, o no se puede realizar.
5. *Error del servidor*. El servidor falló al procesar una solicitud aparentemente válida.

Por ejemplo, el error `404 NOT FOUND` indica que el cliente especificó un URI de un recurso que no existe en el servidor, por tanto la solicitud no se puede realizar. El RFC 2616 de HTTP/1.1 define 41 `status_code`, algunos de los más comunes se listan en la Tabla 13.

Tabla 13. Códigos de estado HTTP comunes

Código	Descripción
`200 OK`	La solicitud HTTP fue exitosa. Por ejemplo, si la solicitud fue un `GET`, la respuesta contendrá el recurso solicitado en el cuerpo del mensaje. Si la solicitud fue un `POST`, los datos enviados por el cliente fueron procesados exitosamente.
`204 NOT CONTENT`	El servidor procesó la solicitud exitosamente, pero no es necesario generar un recurso como respuesta y por ende el cuerpo del mensaje está vacío.
`206 PARTIAL CONTENT`	El servidor responde con un fragmento del recurso solicitado, tal como fue pedido por el cliente web. Esto es de utilidad para continuar descargando un recurso incompleto o permitir varias descargas simultáneas en partes diferentes del mismo recurso. Esta funcionalidad es ampliamente explotada por los administradores de descargas {*download managers*}.
`301 MOVED PERMANENTLY`	El recurso ha sido movido a otro URI y el cliente debe obtener el nuevo recurso con una nueva solicitud.
`302 FOUND`	No debería usarse. Normalmente se quiere decir 303.
`303 SEE OTHER`	El recurso está disponible en otro URI, el cual el cliente debe obtener con una solicitud nueva. Puede usarse para evitar sobrecargar un servidor web o redireccionar algún recurso.
`400 BAD REQUEST`	La sintaxis de la solicitud es incorrecta.
`401 UNAUTHORIZED`	El cliente intenta acceder a un recurso que está protegido y no ha provisto credenciales o las ha fallado. La respuesta incluye un reto al usuario, el cual debe proveer un nombre de usuario y contraseña.
`403 FORBIDDEN`	La solicitud es válida pero el servidor se niega a completarla, por ejemplo, tras varios intentos el usuario falla el proceso de autenticación.
`404 NOT FOUND`	El recurso no se encuentra pero podría estarlo en el futuro.

Código	Descripción
`410 GONE`	El recurso no se encuentra y nunca más lo hará. Es útil para avisar a los motores de búsqueda que eliminen el recurso de sus índices.
`500 INTERNAL SERVER ERROR`	Mensaje genérico, para cuando no hay uno 5XX más específico.
`501 NOT IMPLEMENTED`	El método de solicitud {*request method*} o alguna característica es válida ante el estándar HTTP, pero el servidor web no lo implementa.
`503 SERVICE UNAVAILABLE`	El servidor web no está disponible por sobrecarga o en mantenimiento. Normalmente es una condición temporal.

2.2.4.7. Response header

La sección **encabezado de respuesta HTTP** {*HTTP response header*} permite al servidor enviar información adicional sobre la respuesta al agente de usuario en forma de líneas, cada una corresponde a un campo. Estos campos utilizan la misma notación que la sección de encabezado de solicitud HTTP y también se pueden extender mientras los agentes estén de acuerdo en ellos. Algunos se incluyen en la Tabla 14.

Tabla 14. Campos estándar en el encabezado de respuesta HTTP

Campo	Descripción
`ETag:`	Contracción de "Entity Tag". Es una cadena que permite identificar el estado del recurso, de tal forma que si se hace una modificación del recurso en el servidor, su ETag variará. Esto permite al cliente saber si su copia en el caché está actualizada o ha variado en el servidor. Ej.: `Etag: "239876f-4b8c-429fe67474a80"`.
`Location:`	Permite al servidor indicarle al agente de usuario que el recurso solicitado ha sido movido a un nuevo URI. El agente usuario puede entonces solicitar el nuevo recurso, lo que se conoce como una "redirección".
`Server:`	Contiene información sobre el servidor web que genera la respuesta. Consta de varias parejas `módulo/valor` en orden de importancia para identificar al servidor. Ej.: `Server: Apache/2.2.9 (Debian) PHP/5.2.17-0.dotdeb.0`.
`WWW-Authenticate:`	Solicita al usuario autenticarse para acceder a un recurso en una respuesta `401 UNAUTHORIZED`. Es seguido por el número de intentos permitidos. Ej.: `WWW-Authenticate: 3`.

2.2.4.8. HTTP/2

El RFC 7540 regula la versión 2 de HTTP, conocida como HTTP/2, la cual no reemplaza a HTTP/1.1, sino que ambas son vigentes y el agente de usuario escoge con cuál de las dos establecer la sesión. El propósito de HTTP/2 es hacer un uso aún más eficiente de la red, mediante las siguientes optimizaciones:

1. *Compresión de los encabezados*.
2. *Concurrencia de intercambios* en una misma sesión.

3. *Priorización de intercambios*.
4. *Envíos del servidor no solicitados por el cliente*. El servidor puede especulativamente anticipar cuáles recursos ocupará el cliente y enviarlos antes de que éste los solicite.

2.2.5. Ejemplo de una sesión HTTP

Supóngase que el sitio web ubicado en `www.ejemplo.com` está en construcción y tiene sólo dos recursos, un `index.html` y una imagen `img/ucr.png`. El contenido del recurso `index.html` se encuentra en el Listado 4.

Listado 4. Contenido de una página web hipotética

```
<html>
   <head><title>Desarrollo de aplicaciones web</title></head>
   <body>
      <img src="img/ucr.png" alt="Escudo UCR" />
      <img src="img/ecci.png" alt="Escudo ECCI" />
      <h1>Presentación</h1>
      En este curso el estudiante aprenderá a[...]
   </body>
</html>
```

Un visitante accede con su navegador a `www.ejemplo.com`. La interacción entre el navegador web y el servidor web se aprecia en el diagrama de secuencia de la Figura 21.

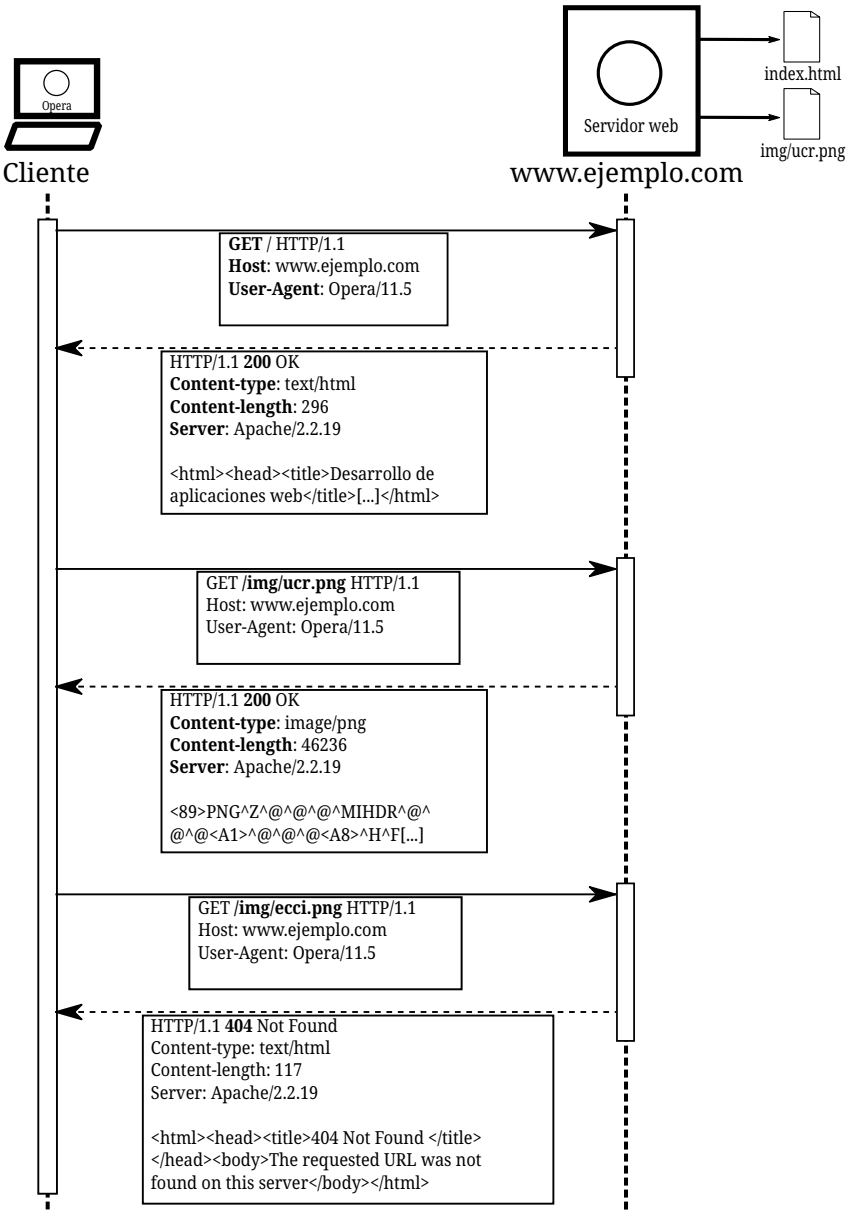


Figura 21. Diagrama de secuencia de una sesión HTTP

Al escribir la dirección `http://www.ejemplo.com/` el navegador contacta a la máquina pasarela {*default gateway*} que tenga configurado el sistema operativo de la máquina local y le pide que le resuelva la dirección IP del dominio `www.ejemplo.com`. Al obtener la dirección IP, el navegador establece una conexión TCP en el puerto 80 con `www.ejemplo.com` y éste acepta. A partir de este momento se ha establecido una sesión HTTP; el navegador y el servidor web pueden intercambiar mensajes HTTP.

El navegador solicita el recurso que el usuario indicó en el URI, en este caso es "/". Ensambla un

mensaje de solicitud {*HTTP Request*} con el método `GET`, escribe algunos campos de encabezado (HTTP request header fields) y un cambio de línea. Lo envía al servidor.

El servidor recibe la solicitud HTTP y de acuerdo a su configuración local, obtiene que el recurso "/" equivale al archivo `index.html`. Ensambla una respuesta HTTP {*HTTP response message*} con el código de estado 200 {*status code*} indicando que el recurso fue encontrado y la solicitud se procesó exitosamente. Agrega unos campos de encabezado de respuesta {*HTTP response header fields*} y anexa el contenido del recurso `index.html` literalmente en el cuerpo del mensaje. Lo envía al cliente.

El navegador recibe el mensaje de respuesta HTTP y viendo que la solicitud fue exitosa, extrae el recurso `index.html` guiado por los campos de encabezado HTTP y lo almacena en su caché, probablemente en forma de archivo. El motor del navegador {*web rendering engine*} inicia el análisis *parsing* y despliegue {*rendering*} del recurso en la ventana del usuario. Al analizar la línea 4 del `index.html`, el navegador descubrirá que necesita otro recurso para presentarlo en la página, y ensambla otra solicitud al servidor, de la misma forma que hizo con `index.html`, pero esta vez por `img/ucr.png`.

El servidor web recibe la nueva solicitud y la procesa de la misma forma que hizo previamente con `index.html`. El recurso también existe. La única diferencia es que `img/ucr.png` es un recurso binario, de ahí la importancia de los campos del encabezado que ayudarán al navegador a procesar el recurso adecuadamente, en especial el `Content-type: image/png`. El servidor envía el mensaje de respuesta.

El navegador recibe el mensaje y al ver que es exitoso, agrega el recurso en su caché y guiado por los campos del encabezado del mensaje, interpreta el contenido binario como una imagen PNG y el motor del navegador la coloca en su lugar dentro de la ventana. Aunque podría no hacerlo, el navegador guarda una copia del recurso su memoria caché identificado por su URI, por una razón de eficiencia. Cada vez que el navegador tiene que cargar un recurso (URI), revisa primero su caché. Si tiene una copia, solicitará al servidor información sobre el recurso con el método `HEAD`, para determinar si el recurso ha cambiado en el servidor. Si no ha cambiado, cargará la copia del caché y evitará consumo de recursos (ancho de banda) y espera de los usuarios.

Al analizar la línea 5 del documento `index.html` (Listado 4), el navegador encontrará de que necesita también el recurso `img/ecci.png`. Ensambla una nueva solicitud `GET` y la envía al servidor como hizo anteriormente. El servidor web recibe la solicitud y trata de localizar el recurso. Al notar que no se encuentra en su sistema de archivos, construye una respuesta HTTP con código de estado `404 Not Found`, indicando que el agente del usuario solicitó un recurso que no puede ser resuelto. En el cuerpo del mensaje agrega un texto más explicativo y en los campos del encabezado HTTP detalles de cómo interpretar este texto. Lo envía al cliente.

El navegador recibe el mensaje de respuesta `404 Not Found`. Al no obtener un recurso para desplegar {*rendering*}, opta por dibujar un rectángulo genérico indicando la ausencia y rellena con el texto alternativo que se encuentra en el documento `index.html`. Continúa de esta forma analizando y desplegando el recurso `index.html` hasta alcanzar su final.

Ejercicio 15 [multiple_web_servers, 5 pts]

¿Se puede tener dos servidores web instalados y activos en un mismo equipo con una única interfaz de red? Si la respuesta es afirmativa ¿cómo harían para diferenciar cuáles peticiones de los clientes le pertenece a cada uno?

Ejercicio 16 [resource_location, 5 pts]

¿Cuáles son los pasos que realiza un servidor web para encontrar en el sistema de archivos local un recurso solicitado a través de un URI? Por ejemplo, si intenta acceder a:

```
http://localhost/bio/foto.jpg
```

¿a qué archivo de su sistema de archivos, exista o no, intenta acceder el servidor web? ¿A qué recurso intenta acceder el servidor web cuando se le solicita simplemente la siguiente dirección?

```
http://localhost/
```

Ejercicio 17 [refresh_http_session, 5 pts]

Un usuario tiene cargada una página web y presiona el botón de refrescar. ¿Cómo debería el navegador web implementar esta operación? ¿Existe algún mecanismo eficiente para reducir el tráfico de red?

Ejercicio 18 [telnet_web_talk, 15 pts]

Hable HTTP/1.1 con un servidor web. Escoja un sitio web que visite con frecuencia. Indague cómo utilizar el comando `telnet` o `nc` (netcat en macOS) para conectarse al puerto 80. Ensamble los mensajes de solicitud HTTP como lo haría un navegador. Conviene preparar estos mensajes primero en un editor de texto.

Solicite al menos una página HTML, un recurso binario que sabe que existe en el servidor, y un recurso que no existe. Además de `GET`, pruebe al menos otros tres métodos de solicitud HTTP. Transcriba su sesión HTTP a un documento de texto.

2.3. Infraestructura web

Hacer disponible un sitio web a los usuarios no es el mismo proceso de instalar y correr un programa en un sistema operativo. Se debe escoger, adquirir o alquilar, configurar, y mantener la infraestructura que permitirá al sitio web "correr". La **infraestructura web** {*web infrastructure*} es el conjunto de recursos necesarios para poner en operación la arquitectura web. Estos recursos incluyen hardware, software, conectividad, energía, e instalaciones físicas.

Aunque no se cuenta como parte de la infraestructura, esta requiere recurso humano especializado para administrarla, cumplimiento de normativas, estrategia y otros procesos que no son triviales de articular, con el fin de asegurar un 99.99% de disponibilidad del sitio web. Por esto, existe una

numerosa oferta a cargo de organizaciones que proveen servicios de infraestructura. A continuación se presentan algunos de estos servicios comunes, como alojamiento web, registro de nombres de dominio, y monitoreo web.

2.3.1. Alojamiento web

El **alojamiento web** {*web hosting*} es la infraestructura básica requerida para que un sitio web pueda funcionar. Si usted desea publicar un sitio web que esté disponible al menos el 99% del tiempo para sus usuarios, deberá abordar tareas como las siguientes.

- Adquirir varias computadoras y un espacio físico con ventilación donde ubicarlas.
- Adquirir generadores eléctricos y baterías {UPS, *uninterruptible power supply*} para evitar que el servicio se interrumpa pese a fallos eléctricos.
- Adquirir equipo de red y de seguridad, diseñar una topología eficiente y segura para conectar sus equipos.
- Adquirir suficiente ancho de banda con redundancia de enlaces, idealmente con más de un proveedor de servicios de Internet {ISP, *internet service provider*} para tratar de que el sitio web se mantenga disponible pese a fallos de conexión.
- Adquirir equipo y software que ayude a detectar y recuperarse ante fallos de hardware, por ejemplo, el malfuncionamiento de un disco duro, o sobrecalentamiento de un procesador.
- Instalar y configurar software como el servidor web, servidor de bases de datos, software de seguridad.

Si usted realiza la lista no-exhaustiva anterior, estará haciendo un **alojamiento interno** {*in-house hosting*}. Tiene la ventaja de conferir seguridad física y control sobre la infraestructura. Sin embargo, adolece de muchas desventajas que pueden hacerse evidentes ante una gran demanda de usuarios, fallos de hardware, cortes eléctricos, o caídas de conectividad. Muchos equipos web prefieren evitar el alojamiento interno y delegar las responsabilidades a una organización que se especialice en brindar servicios de alojamiento web. El término "alojamiento" u "hospedaje" es una metáfora de las personas que pagan para ocupar un espacio físico y acceder a los recursos de un hotel o apartamento.

La oferta de proveedores de alojamiento web es considerable y variada, desde lo gratuito y limitado hasta lo ostentoso. La elección del proveedor no debe tomarse a la ligera. Un servicio lento o intermitente creará en los visitantes una mala imagen del sitio web sin importar qué tan bien esté programado o diseñado. Si se va a contratar alojamiento web, conviene encontrar un balance entre el costo y el rendimiento. La primera decisión a tomar es el tipo de alojamiento a adquirir, que usualmente se clasifica en las siguientes cuatro categorías.

1. **Alojamiento compartido** {*shared hosting*}. Su sitio web comparte los recursos con otros sitios, como el procesador, memoria RAM, el mismo servidor web, el servidor de bases de datos, entre otros. Usted no tiene derechos de administración, ya que un cambio en la configuración afectaría a los otros sitios. Su sitio web compite por los recursos compartidos. Si otro sitio consume mucho CPU o memoria, o tiene vulnerabilidades de seguridad, el suyo podría verse afectado negativamente. Es la variante más común y la más económica.
2. **Servidor virtual privado** {VPS, *Virtual Private Server*}. Usted alquila una máquina virtual, la cual es de su uso exclusivo (privado), y por ende tiene permisos de administración sobre ella. Puede, por ejemplo, reiniciarla o instalar software en el sistema operativo. Normalmente se acceden por

una intefaz de línea de comandos remota (SSH, *secure shell*) o un escritorio remoto {*remote desktop*}. Dado que varias máquinas virtuales comparten el mismo equipo físico, su sitio competirá por los recursos de hardware con otras máquinas virtuales. Es ligeramente más costoso que el alojamiento compartido, pero con ventajas sustanciales.

3. **Alojamiento dedicado** {*dedicated hosting*}. Usted alquila un equipo físico que es para su uso exclusivo, no compartido con otros sitios web. Al igual que en un servidor virtual privado usted puede escoger el sistema operativo a instalar y tiene privilegios de administración sobre él, sin embargo, es mucho más costoso por la exclusividad del equipo. Existen modalidades de este servicio que le permiten acceder físicamente al centro de datos y administrar el hardware de su servidor {*collocated hosting*} (Connolly & Hoar, 2018).
4. **Alojamiento en la nube** {*cloud hosting*}. Permite que su sitio web esté distribuido en una cantidad arbitraria de máquinas virtuales. Esta cantidad puede crecer o decrecer para ajustarse dinámicamente a la demanda de recursos por parte de sus usuarios. Usted paga también dinámicamente por los recursos solicitados, de tal forma que ante poca demanda de su sitio, menores serán los costos de alojamiento. Requiere dominio del paradigma de computación distribuida y de interfaces de desarrollo {API, *application programming interface*} particulares para poder alojar recursos virtualizados dinámicamente.

Para pequeños sitios web documentales o desarrollos incipientes, el autor puede beneficiarse de servicios de alojamiento web gratuitos. Existen aplicaciones web que listan y comparan estos servicios, por ejemplo Free Web Hosting. Antes de decidirse por uno, revise los términos de servicio y otros detalles para saber que cubren sus necesidades y evitar sorpresas.

Ejercicio 19 [student_production_site, 10 pts]

Usted tiene un repositorio de control de versiones personal y uno de sus clones es un ambiente de desarrollo local. Publique su repositorio en un sitio web en producción.

Si participa de un curso formal, acuerde con su profesor(a) el servicio de alojamiento y hágale saber la dirección de su sitio web personal. Pueda que la universidad disponga de un servidor web dedicado para este fin y usted obtenga acceso.

Como alternativa puede suscribir un servicio de servidor virtual privado en la nube. Programas como GitHub Student Developer Pack pueden proveer estos recursos de forma gratuita por un tiempo limitado. Servicios de alojamiento en la nube, como Google Cloud Platform, Amazon Web Services, y Microsoft Azure, proveen periodos de prueba de un año limitados a un consumo prestablecido. En particular el programa AWS Educate de Amazon ofrece recursos gratuitos a estudiantes universitarios, y Google Cloud Platform Free Tier ofrece servicios gratuitos por un periodo indefinido de prueba.

2.3.2. Registro de nombre de dominio

Disponible en la edición completa de este material.

2.3.3. Ambiente de desarrollo, producción, y pruebas

Normalmente un equipo de desarrollo dispone de tres instancias del sitio web:

1. **Ambiente de producción** {*production environment*}. Esta instancia aloja el sitio web que atiende a los usuarios reales bajo el nombre de dominio oficial del sitio web. Normalmente se compone de documentos, programas, y bases de datos. Se considera estable y evita el reporte de errores a los usuarios. Los programas pueden estar optimizados, sin incluir código fuente {*release mode*} o en formato ofuscado por motivos de seguridad.
2. **Ambiente de desarrollo** {*development environment*}. Esta instancia es de uso exclusivo de los desarrolladores. Cada miembro del equipo web tiene un ambiente de desarrollo propio. Puede contener un subconjunto del sitio web total. Se tiene acceso al código fuente original de los programas, y estos reportan errores y advertencias al programador.
3. **Ambiente de pruebas** {*test environment* o *sandbox*}. Esta instancia es una réplica del sitio web en producción, con la diferencia de que reporta advertencias a los desarrolladores. Es necesiaria cuando los ambientes de desarrollo son subconjuntos del sitio web completo, especialmente las bases de datos. Permite importar los cambios hechos por los desarrolladores y probarlos en un ambiente controlado antes de que sean publicados a los usuarios.

Normalmente los ambientes anteriores están en computadoras diferentes. La complejidad puede crecer si los ambientes son distribuidos, por ejemplo, si el almacenamiento está en un servidor dedicado o se usan aplicaciones en sistemas operativos distintos. Es un trabajo arduo dar mantenimiento a esta infraestructura. Existen herramientas de uso difundido que ayudan en esta labor basadas en contenedores (por ejemplo, Docker), o en virtualización (por ejemplo, Vagrant).

2.3.4. Promoción del sitio web

Disponible en la edición completa de este material.

2.4. Resumen

Pendiente.

Parte II. Contenido

Contenido se refiere a la información que se intercambiará entre los usuarios en un sitio web. El contenido puede ser texto, imágenes, sonidos, vídeos, o cualquier trozo de información susceptible de ser identificado por un URI. Esta parte consta de capítulos que permiten representar contenido de tal forma que pueda ser intercambiado a través de la arquitectura web.

1. El lenguaje de marcado extendido (XML) es una notación estricta y formal para estructurar documentos. Confiere al desarrollador el poder de representar fácilmente casi cualquier tipo de información de una manera que es manipulable para la computadora. XML es la adaptación hecha por el Consorcio Web del lenguaje SGML, con muchos propósitos, entre los que sobresalen: el intercambio de información entre aplicaciones, y escribir una variante más eficiente de HTML para aplicaciones web llamada XHTML.
2. El lenguaje de marcado de hipertexto (HTML) es la notación para escribir contenido textual en páginas web, tanto documentales como de aplicaciones web.
3. Los formatos de medios permiten representar imágenes, audio, y vídeo en archivos o flujos, que son incrustados o transferidos a través de la arquitectura web. Requiere conocer formatos y codificaciones soportadas por esta tecnología.

Capítulo 3. El lenguaje de marcado extensible XML

Los seres humanos tienen diversas necesidades de información, y cuando esta se debe almacenar en la computadora, se suele hacer en forma de bases de datos o documentos digitales. Las *bases de datos* restringen en alguna medida la representación de información del mundo real a una estructura que debe definirse formalmente para poder ser eficientemente manipulada por el computador. Elaborar una base de datos es una tarea compleja que consume recursos considerables. Los *documentos* por el contrario, son menos estrictos por lo que pueden representar casi cualquier tipo de información del mundo real, pero tal libertad limita el poder de manipulación del computador. El XML es un lenguaje formal que busca las ventajas de ambos: la representación flexible de información en forma de documentos que pueden ser manipulados aprovechando el poder del computador.

El **((lenguaje de marcado extensible)** {XML, *eXtensible Markup Language*}, es un conjunto de reglas estándar para representar información en forma de *documentos digitales*, que se caracteriza por permitir marcado definido a conveniencia del autor. Representar digitalmente un documento implica transformarlo en algún tipo de código legible por la computadora para que esta sea capaz de almacenarlo, procesarlo, buscarlo, transmitirlo, mostrarlo e imprimirlo (Goldfarb & Prescod, 1999).

A modo de ejemplo, supóngase que un profesor quiere anotar en su computadora varias ideas sobre el curso que va a impartir. No crea una base de datos para ello, sino que crea un nuevo documento y escribe algún texto como el que se aprecia en el Listado 5

Listado 5. Ejemplo un documento sin estructura

```
Aplicaciones web
Autor: Jeisson Hidalgo-Céspedes <jeisson.hidalgo@ucr.ac.cr>

Este documento contiene un resumen de temas que servirán de
apoyo a los estudiantes del curso CI-2413 impartido en la Universidad...



INTRODUCCIÓN A LA TECNOLOGÍA WEB

De todas las aplicaciones que se han construido sobre Internet, la
World Wide Web (WWW o simplemente web) ha sido la más popular, tanto que,
muchas personas cuando escuchan el término "Internet" realmente[...]

La popularidad de la web puede adjudicarse a su facilidad de uso y por
ser el más exitoso de los sistemas distribuidos en la actualidad. Los
documentos se almacenan en computadoras distintas, con sistemas[...]


Historia de la web

La web fue conceptualizada en un artículo de 1989 de Tim Berners-Lee,
quien se convertiría en uno de sus grandes líderes. A finales de 1990,
Berners-Lee desarrolló el Protocolo de transferencia de hipertexto[...]
```

En un inicio el documento resulta muy práctico, pero conforme crece en tamaño y complejidad se descubren las dificultades de procesamiento del computador. Por ejemplo, no hay forma de decirle al computador que automáticamente enumere los títulos y cree una tabla de contenidos, porque simplemente no se ha identificado cuáles son los títulos en el documento. Sin embargo, el autor podría utilizar alguna convención para distinguir los títulos, por ejemplo, precederlos por varios cambios de línea en proporción inversa al nivel del título, o precederlos de caracteres especiales (por ejemplo, el carácter numeral `#`), o hacer que siguiente línea esté compuesta únicamente de símbolos de igualdad (`====`). Cualquiera que sea el caso, el autor debe ser cuidadosamente consistente con su notación. Luego tendría que programar algún software que guiado por estas convenciones cree el índice en forma automática, y provea otras funcionalidades similares.

Supóngase que otro profesor, con la misma necesidad de representar información de un curso en su computadora, crea una convención distinta, y además un software distinto. Además del trabajo redundante, estos dos autores no podrán compartir sus trabajos ya que desconocen sus notaciones o son incompatibles. Aquí es donde tiene cabida XML. XML es simplemente un conjunto de convenciones para escribir documentos y un conjunto de reglas estándar que le dice a los software cómo deben procesar esos documentos guiados por las necesidades de los autores. El documento sin estructura del Listado 5 podría representarse como el Listado 6 en XML.

Listado 6. Ejemplo de un libro hipotético en XML (book.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "book.dtd">

<book name="AplicacionesWeb" version="1.0">
   <title>Aplicaciones web</title>
   <authors>
      <author name="Jeisson" percent="100">
         <fullname>Jeisson Hidalgo-Céspedes</fullname>
         <email>jeisson.hidalgo@ucr.ac.cr</email>
      </author>
   </authors>

   <chapter name="Preliminar" type="prelims">
      <section name="Prologo">
         <title>Prólogo</title>
         <p>Este documento contiene un resumen de temas que servirán de
            apoyo a los estudiantes del curso CI-2413 impartido...</p>
      </section>
      <section name="Agradecimientos" type="acknowledgements" author="Jeisson">
         <title>Agradecimientos</title>
         <p>Quiero agradecer en primer lugar, a usted, que con sus ojos
            da vida a estas inanimadas palabras pintadas en...</p>
      </section>
   </chapter>

   <chapter name="Introduccion">
      <title>Introducción a la tecnología web</title>
      <p>De todas las aplicaciones que se han construido sobre Internet...</p>
      <p>Su popularidad puede deberse a su &amp;quot;facilidad de uso&amp;quot; y...</p>

      <section name="HistoriaWeb">
         <title>Historia</title>
         <p>La web fue conceptualizada en un artículo de 1989 Tim Berners-Lee</p>
      </section>
   </chapter>
</book>
```

Un documento XML es un archivo de texto. El texto se clasifica en tres tipos: datos de carácter, elementos, y entidades. Los **datos de carácter** {*character data*} corresponden al texto normal que el autor quiere escribir, como el presentado originalmente en el Listado 5. La computadora poco puede hacer con este texto, por ejemplo, es difícil crear la tabla de contenidos porque desconoce cuáles partes del texto son los títulos.

XML solicita al autor que además de escribir el texto, distinga claramente cada parte del mismo. En terminología XML, una parte de un documento se conoce como **elemento**. Por ejemplo, los elementos que componen un libro son: portada, tabla de contenidos, partes, capítulos, secciones, párrafos, palabras, títulos, notas, etc.

Finalmente, las **entidades XML** {*XML entities*} permiten darle nombre a un trozo de información

(incluso binaria) y reutilizarla en una o varias partes del documento. Si se necesita representar información binaria, ésta no se almacena directamente en el documento XML, sino que se aloja en recursos externos (archivos binarios o bases de datos), y se les hace referencia desde el documento XML a través de entidades.

Los elementos se pueden anidar, formando un árbol. Los elementos dentro de otros se llaman **elementos hijos** {*child element*} y a los contenedores, **elementos padres** {*parent element*}. Sólo puede existir un único **elemento raíz** {*root element*}, también llamado **elemento documento** {*document element*}, que contiene a todos los demás. En el Listado 6 el elemento con identificador `book` es elemento raíz que abarca desde la línea 4 hasta la 36, y por tanto, a todo el documento, de ahí su nombre. El elemento es sin duda el constructo XML más frecuente que usará un autor, sin embargo, antes de empezar a escribirlos, el autor debe proporcionar otros detalles sobre el documento XML que se explican a continuación.

Ejercicio 20 [xml_experience_example, 5 pts]

Describa un ejemplo de una situación en que usted se hubiera beneficiado de haber representado información en XML. Puede ser un proyecto de programación que hizo en algún curso previo, o un proyecto profesional en que haya participado. Sugerencia: piense en archivos de texto sin estructura o bases de datos que haya utilizado previamente.

Ejercicio 21 [xml_os_configuration, 5 pts]

En los sistemas operativos basados en Unix, es común que los programas guarden su configuración en archivos de texto. Por ejemplo, la configuración de dispositivos y particiones se configura en `/etc/fstab`, los programas a ejecutar de forma periódica en `/etc/crontab`, el intérprete de comandos Bash guarda su configuración en un archivo oculto `.bashrc` en la carpeta del usuario, el servidor web Apache en varios archivos de texto en la carpeta `/etc/apache2`, el servidor de SSH en `/etc/ssh/`, entre muchos otros. Cada uno de estos programas utilizan una notación diferente para representar su configuración. Liste ventajas y desventajas si todos estos programas utilizaran notación XML para representar sus configuraciones.

Ejercicio 22 [xml_logs, 5 pts]

Muchos programas registran eventos o errores en bitácoras. Por ejemplo, todas las solicitudes HTTP que el servidor web Apache recibe se registran en `/var/log/apache2/access.log`, y los eventos de arranque o de solicitudes al *kernel* de Linux se registran en `/var/log/messages`. Normalmentelas bitácoras son archivos de texto y cada programa utiliza su propio formato. ¿Sugeriría usted que las bitácoras se escriban en XML? Justifique su respuesta.

3.1. Encabezado de documento

Los documentos XML constan de dos partes: encabezado {*head*} antes del elemento raíz, y cuerpo *body*

que inicia con el elemto raíz (Figura 22). El encabezado de un documento XML recibe el nombre de **prólogo de documento XML** {*XML document prolog*} y almacena información que describe al cuerpo del documento, como la versión de XML, la codificación, el tipo de documento (DTD) al que pertenece, y otros. El cuerpo del documento XML recibe el nombre de **instancia de documento XML** {*XML document instance*}, que contiene los datos reales del documento.

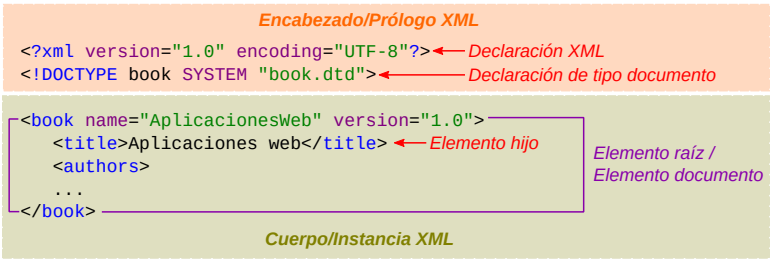


Figura 22. Partes de un documento XML

El *prólogo de un documento* suele incluir la *declaración de XML* y la *declaración del tipo de documento*, en ese orden, además de otras características como comentarios e instrucciones de procesamiento. Todas son opcionales, pero se aconseja maximizar la cantidad de información en el prólogo ya que ayuda al procesamiento posterior del documento (Goldfarb & Prescod, 1999). El prólogo termina cuando abre la etiqueta de inicio del elemento raíz, es decir, donde inicia la instancia de documento.

3.1.1. La declaración XML

La **declaración XML** indica que el archivo de texto es un documento que sigue las reglas XML e indica la codificación del mismo. Tiene la forma mínima `<?xml version="1.0"?>` y a modo de ejemplo el Listado 7 presenta su forma más amplia. El prólogo tiene sintaxis de una instrucción de procesamiento, es decir, una etiqueta que se abre y cierra con un signo de pregunta. Tiene tres atributos: la versión, la codificación, y el documento autónomo:

1. La *declaración de versión XML* con el atributo `version`, indica la versión de XML a la que se apega el documento, normalmente `"1.0"`.
2. La *declaración de codificación* con el atributo `encoding` indica la codificación de caracteres del documento. Aunque los procesadores XML tienden a reconocer automáticamente la codificación, es mejor indicarlo explícitamente, por ejemplo, `"UTF-8"`, `"UTF-16"` o `"iso-8859-1"` (Goldfarb & Prescod, 1999). El valor de este atributo no es sensitivo a mayúsculas. Lo importante es que este valor refleje realmente el tipo de codificación que usó para generar el documento.
3. La *declaración de documento autónomo* con el atributo `standalone` no se suele utilizar con frecuencia ni tampoco se recomienda su uso. Su comprensión es compleja y se explica ampliamente en (Goldfarb & Prescod, 1999).

Listado 7. Ejemplo de un prólogo XML en su forma más extensa

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

Ejercicio 23 [xml_text_encoding, 10 pts]

Investigue la diferencia entre un conjunto de caracteres (character set) y la codificación del texto (text enconding). ¿Cómo esto afecta a los documentos XML?

Escoja una palabra en español con al menos dos caracteres no ASCII. Guarde la palabra en cuatro archivos de texto independientes, cada una con codificaciones diferentes. Sugerencia: puede usar Latin-1 y las tres codificaciones de Unicode.

Utilice un visualizador de archivos binarios (por ejemplo, el comando `xxd` en Unix) para examinar cada uno de los cuatro archivos. En un documento `xml_text_encoding.md` cree una tabla, donde la primera columna es el nombre del archivo, y las subsecuentes columnas representan un único byte. En cada celda escriba la representación decimal (un número entre 0 y 255) de cada byte del archivo, en el orden en que aparecen.

Ejercicio 24 [xml_inventory_1, 5 pts]

Suponga que un cliente tiene repetidas dificultades para localizar eficientemente activos en su empresa dispersa en varias sucursales, y le ha contratado para que le desarrolle un sistema de administración del inventario, el cual debe ser accesible vía web. Usted decide utilizar documentos XML para representar los activos y así transferirlos entre el navegador y el servidor web mediante AJAX.

Cree un documento XML vacío, que en ejercicios posteriores utilizará para ir representando el inventario. Utilizando un editor de texto, cree un archivo con extensión `.xml`. Incluya la declaración XML. Declare la codificación como `utf-8` y asegúrese de que su editor de texto esté realmente utilizando esta codificación. *Sugerencia*: indague las facilidades que disponga su editor de texto para ayudar en la edición de código XML, como autocerrar etiquetas.

Guarde su documento en una carpeta `xml_inventory`. Para los ejercicios posteriores que utilizan este archivo, no cree uno nuevo, sino que por cada ejercicio realice al menos un commit con los cambios que haya hecho a este archivo.

3.1.2. La declaración del tipo de documento

Es sabido que las cartas, tesis, guiones y las guías telefónicas son documentos que tienen una estructura muy diferente. Es decir, poseen elementos distintos, cada uno con su propia distribución y orden. Se dice que son *tipos de documentos* distintos. En XML cada tipo de documento se define en una notación formal que plasma su estructura llamada *definición de tipo de documento* {DTD, *document type definition*} y se estudiará luego. Un documento puede declarar que es de un tipo de documento particular utilizando la **declaración de tipo de documento** {*document type declaration*}, como se hizo en la segunda línea del Listado 6, indicando que ese documento es de tipo `book`:

```
<!DOCTYPE book SYSTEM "book.dtd">
```

Se puede pensar en un DTD como una clase en programación orientada a objetos, mientras que un documento XML que sea de ese tipo de DTD es como un objeto que instancia esa clase. El "objeto" documento XML debe cumplir a cabalidad con la estructura descrita en su DTD, cuando esto ocurre, se dice que el documento es de **tipo válido** o simplemente **válido**, de lo contrario, se dice que el documento es de **tipo no válido** o simplemente **no válido** (o incluso, *inválido*) (Goldfarb & Prescod, 1999).

Un documento XML puede no tener un tipo de documento definido, es decir, carece o no cumple con un DTD, por lo que es un documento *no válido* pero puede respetar la sintaxis XML, en tal caso se dice que sólo es un documento **bien formado** {*well formed*}. Los documentos bien formados pero no válidos (no cumplen un DTD) suelen utilizarse para documentos pequeños que deben escribirse de forma rápida. Los documentos válidos siempre están bien formados y son necesarios cuando son muy extensos o deben procesarse por algún sistema computacional (Goldfarb & Prescod, 1999).

La declaración `DOCTYPE` indica el tipo de documento y además instruye al procesador de XML dónde encontrar la definición del tipo de documento (DTD), el cual puede estar escrito en el mismo documento XML o puede encontrarse en una entidad externa (un archivo en disco o red) o una combinación de ambas. Lo más común es que se encuentre en un recurso externo, como se hizo en la segunda línea del Listado 6, la cual instruye al procesador XML que el tipo de documento está en el archivo `book.dtd` en la misma ubicación que el documento XML. La palabra `SYSTEM` indica al sistema que busque el recurso especificado en el URI (Universal Resource Identifier) que le continúa. Más adelante se estudiará la nomenclatura de un DTD.

El *identificador* del elemento que continúa inmediatamente después de `<!DOCTYPE`, por ejemplo, `book` en el Listado 6, indica al procesador XML a partir de qué elemento del DTD se debe incluir, es decir, sólo se incluye el subárbol cuya raíz es precisamente ese elemento.

En XML los **identificadores** se usan para dar nombre a los elementos, entidades u otros. Por ejemplo, la línea 4 del Listado 6 incluye tres identificadores: `book`, `name` y `version`. Los identificadores deben iniciar con una letra y pueden estar seguidos de cero o más letras o los caracteres punto, guión o dígitos. Los identificadores en XML son sensitivos a mayúsculas y minúsculas {*case sensitive*}, esto implica, por ejemplo que `book`, `Book` y `boOK` se tomen como identificadores distintos. El autor puede emplear cualquier identificador válido, excepto aquellos que inician con la cadena “xml” en cualquiera de sus combinaciones de mayúsculas y minúsculas, ya que están reservados para propósitos de estandarización (Goldfarb & Prescod, 1999).

Ejercicio 25 [xml_inventory_2, 5 pts]

Declare el tipo de documento en el archivo XML que creó en el ejercicio anterior. El nombre *XXX* que escoja para su DTD debe completar la frase "Este documento es un *XXX*", o "Este documento contiene un *XXX*". Luego cree un archivo de texto vacío en codificación `utf-8` con nombre `XXX.dtd` y en la misma carpeta donde está su archivo `.xml` creado en el ejercicio anterior.

3.2. Elementos y atributos

Los componentes más comunes de un documento XML son los elementos, los cuales pueden contener atributos. Ambos se estudian en las siguientes subsecciones. Se discute al final de esta sección cómo

los autores confieren semántica a los elementos y los atributos de un documento.

3.2.1. Elementos

Los **elementos XML** {*XML element*} son las partes que componen un documento. El autor debe indicar entre todo el texto que compone el documento, qué trozo de texto conforma cada parte del documento. Sintácticamente un elemento se forma *marcando* la parte del documento entre dos etiquetas. Es decir, el texto (datos de carácter) que conforma un elemento se encierra entre un par de etiquetas. Una **etiqueta** {*tag*} es el texto entre un par de signos "menor que" y "mayor que". Sintácticamente, un *elemento* es la combinación de una etiqueta de inicio (que puede tener atributos), un contenido opcional, y una etiqueta de cierre, como muestra el Listado 8:

Listado 8. Sintaxis de un elemento

```
<id_etiqueta_inicio atributos="valor">contenido</id_etiqueta_cierre>
```

Un elemento tiene dos etiquetas, una de inicio y otra de cierre (o etiqueta de fin). La **etiqueta de inicio XML** {*XML start-tag*} se compone de un carácter menor que (`<`), un `id_etiqueta_inicio` que es un simple identificador y debe respetar las restricciones de los identificadores, seguido por cero o más parejas `atributo="valor"`, y un carácter de mayor que (`>`).

La **etiqueta de cierre XML** o **etiqueta de fin XML** {*XML end-tag*} consta de los caracteres menor que (`<`) y un slash (`/`), luego un `id_etiqueta_cierre` que obligatoriamente debe ser el mismo identificador que el de la etiqueta de inicio del elemento y un carácter de mayor que (`>`). La etiqueta de cierre nunca tiene atributos.

El **contenido del elemento XML** {*XML element content*} puede ser vacío o texto XML, es decir, datos de carácter, otros elementos (y por ende etiquetas) o ambos. Si un elemento no tiene contenido se dice que es un **elemento vacío**, y se puede escribir en cualquiera de las dos formas del Listado 9.

Listado 9. Posibles sintaxis de un elemento vacío

```
<id_elemento atributos="valor"></id_elemento>
<id_elemento atributos="valor"/>
```

La segunda de las formas del Listado 9 consta de una sola etiqueta llamada **etiqueta vacía XML** {*XML empty-element tag*}, la cual finaliza en un slash (`/`) indicando que el elemento ha terminado. Es poco común encontrar etiquetas vacías sin atributos.

Un elemento vacío no puede tener ninguna forma de contenido, ni siquiera espacios en blanco o cambios de línea. Es decir, que si en la primera forma del Listado 9 se incluyera un cambio de línea, se consideraría como un dato de carácter y ese sería el contenido del elemento. Por el contrario, si un elemento tiene contenido (no es vacío) y se omite alguna de las etiquetas (la de inicio o la de fin) el procesador XML deberá alertar de que el documento no está bien formado.

Debe quedar claro que las etiquetas no son elementos. Las etiquetas inician con un `<` y terminan con un `>`, lo demás no son etiquetas (Goldfarb & Prescod, 1999). Los elementos tienen etiquetas y contenido. Los elementos se escriben en el documento XML y son instancias de los *tipos de elementos*,

los cuales se declaran en la definición del tipo de documento (DTD).

3.2.2. Atributos

Los elementos pueden tener atributos que son "una forma de incorporar características o propiedades a los elementos de un documento" (Goldfarb & Prescod, 1999, p. 353). Los atributos se escriben siempre en la etiqueta de inicio (Listado 10). Un atributo se compone de un identificador, un signo de igual (=) y un valor entre comillas dobles, simples, o una combinación de ambas siempre y cuando la comilla que cierra sea del mismo tipo de la que abre.

Listado 10. Ejemplo de atributos de elementos

```
<element att1="value1" att2="value2" att3='value3'>content</element>
```

Cuando se crea un nuevo tipo de documento, es difícil decidir qué representar con elementos y qué con atributos. Las siguientes diferencias pueden ayudar en la decisión. Más adelante se retomarán estas diferencias a un nivel más formal.

1. Un elemento puede tener varios elementos hijos del mismo tipo, por lo que pueden representar partes del documento que se repiten (como los capítulos de un libro) o que se anidan (como las secciones de un libro que pueden tener sub-secciones). Un atributo no puede repetirse en un elemento.
2. Se puede controlar el orden de aparición de los elementos, mientras que los atributos pueden aparecer en cualquier orden.
3. Los elementos pueden tener estructura, como elementos hijos de distinta naturaleza. Los atributos son pequeños textos o números.
4. Los elementos tienden a ser visibles a los lectores o consumidores del documento, los atributos tienden a estar ocultos.

Ejercicio 26 [xml_inventory_3, 20 pts]

Suponga que los activos de su sistema de inventario se relacionan jerárquicamente. Por ejemplo, un inventario para la Escuela de Computación estaría compuesto de un edificio, el cual tiene tiene salas, las salas contienen escritorios, pizarras, teléfonos, computadoras, etc; una computadora tiene procesadores, módulos de memoria RAM, discos duros; y así por el estilo. El cliente necesita además distinguir cuáles activos son fijos y cuáles no, la condición de cada activo (en funcionamiento u ocioso), el estado del activo (buen estado, defectuoso, dañado), y la posibilidad de escribir detalles textuales de cualquier longitud.

La figura Figura 23 muestra un ejemplo de un inventario hipotético. Los rectángulos a la izquierda representan los activos y las flechas hacia la derecha indican los responsables de cada activo. Represente en su documento XML la jerarquía de activos mostrada en esta figura. Sus elementos deben ser genéricos, de tal forma que puedan reutilizarse para inventarios de cualquier otra organización. Sus elementos deben además contener atributos y deben asociarse reflejando la jerarquía natural del inventario. De ser posible, use identificadores en inglés para sus elementos y atributos.

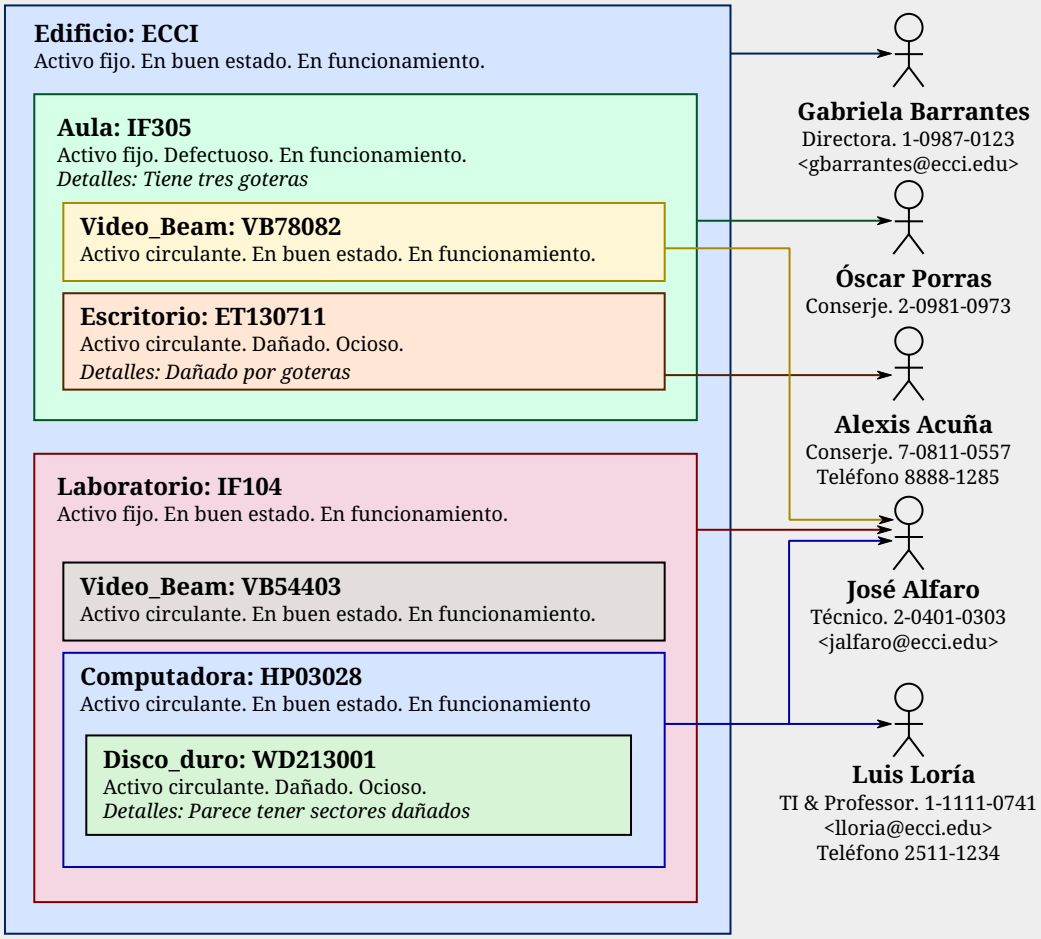


Figura 23. Inventario ficticio y reducido de la escuela de computación de la UCR (administración 2010-2013)

Ejercicio 27 [xml_inventory_4, 10 pts]

Su cliente necesita saber cuál o cuáles empleados están a cargo de un activo (identificados por cédula), qué rol cumplen en la empresa, y tener una forma de comunicación con ellos (correo, teléfono, o ambos). Represente en su documento XML los empleados de la Figura 23.

Si un activo no tiene encargados, se asume que son los mismos del activo que lo contiene. Por ejemplo, los encargados del video beam `VB54403` son los mismos que del laboratorio `IF104`. Esta característica debe reflejarse en su diseño, de tal forma que un activo podría no tener encargados declarados explícitamente. Note que existe una relación `N:M` entre los activos y los empleados. Para este ejercicio, usted no debe redundar los detalles de los empleados en cada activo a cargo.

3.2.3. Semántica de los elementos XML

Cuando se escribe un documento XML el autor escoge los identificadores de los elementos, los atributos, y la estructura a su gusto, y la plasma en un DTD. Cuando una persona observa el marcado de un documento XML y se pregunta "¿qué significa esto?" o bien "¿qué aspecto tendrá después?", estará haciendo preguntas sobre su **semántica**. Por ejemplo, si se encuentra una etiqueta `<T>` esta podría indicar que se trata de un título, una tabla, o alguna otra cosa (Goldfarb & Prescod, 1999).

La computadora no puede inferir la semántica de un documento XML dado, sino que es responsabilidad del autor de la gramática describir esta semántica para que otros autores puedan comprenderlo. Esta descripción se puede realizar en un documento PDF (Portable Document Format), un libro, o cualquier otro medio de comunicación (Goldfarb & Prescod, 1999). Más adelante se hará en comentarios en el DTD.

Como señalan Goldfarb y Prescod, "lo que le interesa a la computadora es el aspecto que se supone debe tener un elemento cuando está formateado, la forma en que debe actuar si es interactivo o qué hacer después de extraer los datos. Estas cuestiones se encargan de especificarlas las hojas de estilo y los programas informáticos" (Goldfarb & Prescod, 1999, p. 350).

Avance de proyecto 16 [prj_distributed_wirestate, 25 pts]

Diseñe el protocolo de paso de mensajes con los que se comunicarán las instancias o procesos de su aplicación web. Este diseño necesita de al menos dos personas que estén ubicadas en sitios distantes, que tengan copias iguales del prototipo en papel elaborado en el Capítulo 1. Si realiza el proyecto de forma individual, solicite ayuda a otras personas con conocimientos de informática.

Las aplicaciones web siguen el paradigma de computación distribuida. Es decir, constan de al menos dos procesos (instancias de programa) que ejecutan el mismo código (simetría) o código distinto (asimetría), normalmente en computadoras diferentes. Es probable que su aplicación web sea asimétrica, dado que la web sigue el modelo cliente-servidor, y por tanto su aplicación necesite de al menos un proceso servidor y de al menos un proceso cliente para funcionar. Por ejemplo, en una sesión de Bingo Social participan al menos cuatro procesos: un servidor web, un proceso cliente para el anfitrión, y un proceso cliente para cada jugador, con al menos dos jugadores para que haya competencia.

Los procesos, aunque se ejecuten en el mismo equipo físico, están aislados unos de otros, y por tanto, no pueden acceder al mismo espacio de direcciones de memoria en el que podrían compartir el estado de la aplicación. Sin embargo, los procesos de una aplicación distribuida deben colaborar para satisfacer los requerimientos de los usuarios. La forma de colaborar en el paradigma distribuido se conoce como comunicación entre procesos {IPC, *inter-process communication*}, y el mecanismo más elemental de comunicación es el paso de mensajes {*message-passing*} que usualmente ocurre a través de una red de computadoras. Los mensajes son trozos de información que un proceso puede enviar a otro (comunicación punto a punto), o que uno o más emisores pueden enviar a uno o más receptores (comunicación colectiva).

Cada persona participante tomará el rol de un proceso de la aplicación web. Por ejemplo, en el Bingo Social un participante será el servidor, otro será el anfitrión, y otros dos serán el jugador 1 y jugador 2. Los participantes colaborarán para lograr una o más tareas básicas del sistema, como "jugar una partida completa de bingo".

Los participantes estarán ubicados en sitios distantes (por ejemplo, en casas distintas) y no podrán comunicarse físicamente (hablar directamente), ni por audio (una llamada telefónica o mensaje de voz), ni por video (teleconferencia o grabaciones). La única forma permitida de comunicación será un chat textual (por ejemplo WhatsApp o WeChat). Sin embargo, antes de iniciar la sesión, los participantes podrán acordar un protocolo de cómo enviar los mensajes de texto, de tal forma que puedan disminuir la cantidad de información a enviar por el chat pero que logre comunicar el mensaje de forma clara a los demás. Antes de iniciar la sesión, los participantes han de tener:

1. Los componentes del prototipo en papel de la aplicación web sobre un escritorio sin ensamblar, o ensamblados si la aplicación tiene un estado inicial que siempre es igual para todas las sesiones.
2. El protocolo de comunicación acordado en un documento compartido modificable (el siguiente avance de proyecto provee más detalles).
3. El chat textual con los demás participantes, deseablemente con acceso a una cámara fotográfica. Es necesario que el historial del chat quede registrado porque se requerirá en avances posteriores.
4. El autómata de estados finito diseñado en la Sección 1.5.2 para la aplicación web y los algoritmos de las reacciones del sistema. Puede estar en un documento compartido modificable.

Una vez que la sesión inicia, los participantes intentarán completar la tarea propuesta con los

recursos anteriores. Podrán usar el autómata de estados finitos y los algoritmos de las reacciones como guía. Por ejemplo, cuando el participante se encuentre en un estado, estará a la espera vigilando por los eventos que salen del estado. Cuando ocurre un evento, seguirá el algoritmo asociado en la reacción. Si el algoritmo indica hacer alguna modificación sobre el prototipo de la aplicación, lo hará directamente sobre su escritorio. Si el algoritmo indica que debe enviar un mensaje a otro proceso, lo hará por el chat al participante correspondiente usando la nomenclatura acordada en el protocolo.

Para saber si el participante que recibió el mensaje realizó la acción esperada, el receptor puede tomar una fotografía del estado de su escritorio y enviarla de regreso al emisor. Si el emisor comprueba que la acción realizada por el receptor es la esperada, podrán continuar con el procedimiento. Si ocurre un problema, por ejemplo el receptor no comprendió el mensaje, faltó información, la información es incorrecta, o el cambio en la fotografía no era el esperado, podrán decirlo directamente en el chat con el fin de que el emisor corrija el mensaje y lo vuelva a enviar. Estas correcciones se repiten hasta que el emisor logre enviar un mensaje que ayude al receptor a lograr el efecto deseado en su prototipo en papel. Los mensajes de error, e intentos intermedios, serán descartados posteriormente del chat, pero son necesarios para construir el mensaje correcto y completo que sí formará parte del protocolo de comunicación final.

Es probable que durante la sesión detecten deficiencias en el autómata o los algoritmos. Pueden incorporar las correcciones en los diseños compartidos, de tal forma que sean visibles para los demás participantes. Si los cambios son muy grandes o incompatibles con el progreso ya realizado, pueden reiniciar la sesión.

Este avance termina hasta que los participantes logren completar la tarea distribuida propuesta con los prototipos en papel. El producto serán las correcciones o mejoras al *wirestate* y a los algoritmos de las reacciones (`design.md`), que deben reflejarse en el repositorio de control de versiones del proyecto.

Avance de proyecto 17 [prj_message_passing_design, 15 pts]

Analice el historial del chat del avance de proyecto anterior y corrija el protocolo de comunicación. Descarte los intentos erróneos, y dudas, y quédese con los mensajes que resultaron eficaces y las fotografías de confirmación. Edite las fotografías para que ocupen cerca de 200kB o menos y agréguelas a la carpeta `design/messages/`. Escriba el historial de comunicación en un documento `design/messages1.md` en Markdown o `design/messages1.adoc` en AsciiDoc. Este documento tiene el historial de mensajes de envío y fotografías de respuesta entrelazadas que lograron completar la tarea.

Utilice su historial de mensajes como guía para escribir su protocolo de comunicación en una sección "Protocolo de paso de mensajes" al final de su documento `design.md`. El protocolo debe constar de cada tipo de mensaje, los nombres de los campos que contienen, el tipo de datos del valor de cada campo, qué proceso envía cada mensaje, y qué proceso recibe cada mensaje. Trate de que los identificadores de los mensajes y campos sean concisos, y claros. Para cada tipo de mensaje conviene proveer un ejemplo tomado del historial de mensajes.

Asegúrese de que el autómata de estados finito (*wirestate*) y los algoritmos utilicen los mismos nombres escogidos para los mensajes. Por ejemplo, un evento que sale de un estado podría tener el nombre de un mensaje (al menos entre paréntesis) para indicar que el evento ocurre cuando el proceso recibe ese mensaje por la red. Si en el algoritmo reacción a un evento el proceso debe enviar un mensaje, indicar el tipo de mensaje y los valores de sus campos. Si el proceso debe esperar por un mensaje indicarlo con la instrucción "Recibir mensaje X", donde X es el tipo de mensaje.

Avance de proyecto 18 [prj_message_passing_test2, 20 pts]

Una vez que en el avance anterior haya refinado el protocolo de paso de mensajes, el autómata de estados finitos, y los algoritmos de los manejadores de eventos (reacciones), realice con sus participantes una segunda sesión de prueba, similar a la descrita en la actividad `prj_distributed_wirestate`, con algunos cambios:

1. Los participantes deben rotar los roles. En especial, el rol del proceso servidor debe ser realizado por un participante distinto al de la prueba anterior.
2. Los participantes (procesos) deben seguir al pie de la letra el autómata, los algoritmos (reacciones), y el protocolo de paso de mensajes. Por tanto, se espera reducir la cantidad de mensajes en el chat a sólo los especificados en estos tres diseños.
3. Los participantes no deben permanecer pendientes del chat, a menos de que los diseños lo soliciten. Es decir, los participantes mirarán al chat sólo cuando se encuentren en un estado que espera por mensajes de red, o cuando estén ejecutando una instrucción "Recibir mensaje X". En cualquier otra circunstancia deben mantener la ventana del chat minimizada u oculta. Una vez que se ha recibido un mensaje esperado, se copia su contenido y se minimiza la ventana para evitar distracciones. La única excepción es para el emisor, quien queda a la espera de la fotografía de confirmación, en caso de que los participantes decidan seguir usando esta estrategia.
4. Si se sigue el modelo al pie de la letra, se espera por un mensaje X en un algoritmo, y llega un mensaje de otro tipo, llega después de otro, o nunca llega en el chat, es un error en el diseño que debe corregirse. Si un participante está en un estado del autómata y por su chat llega un mensaje que no está listado en los eventos del estado, también es un error que debe repararse. Las correcciones pueden hacerse en el momento mientras los participantes estén enterados, y si es necesario se puede reiniciar la sesión.

La sesión termina cuando los participantes logren completar la tarea que se haya escogido probar. Los productos del avance son las mejoras y correcciones al autómata (`wireflow.svg`), los algoritmos (`design.md`), el protocolo de red (`design.md`) y los ejemplos de mensajes (`design.md`), los cuales deben registrarse en control de versiones. No es necesario depurar ni registrar el historial del chat en un documento (`design/messages2.md`), pero los participantes pueden hacerlo si lo consideran de utilidad.

Avance de proyecto 19 [prj_message_passing_xml, 15 pts]

Convierta su protocolo de paso de mensajes a XML. En avances previos, para cada tipo de mensaje usted creó un ejemplo en el archivo `design.md`. Copie cada ejemplo y tradúzcalo en notación XML. Decida los elementos y atributos de cada mensaje. Asegúrese de que los identificadores sean significativos.

Si trabaja en equipo, haga esta actividad de forma individual, independiente de los demás miembros del equipo. Luego discutan los resultados, sus ventajas y desventajas, y acuerden la notación final.

Finalmente cree una copia del archivo `design/messages1.md` como `design/messages1.xml`. Convierta cada mensaje a la notación XML acordada. Puede eliminar las imágenes de confirmación o convertirlas en comentarios. Sugerencia: utilice expresiones regulares para hacer esta conversión de forma rápida.

Necesitará un elemento raíz para el documento `design/messages1.xml`. Decida su identificador y atributos y agréguelo al documento.

3.3. Otras formas de marcado

Los elementos y atributos son las formas más comunes de marcado en el cuerpo del documento. Sin embargo, hay otras formas de marcado importantes: las entidades, las secciones CDATA, los comentarios, y los espacios de nombres.

3.3.1. Entidades predefinidas

Una **entidad XML** {*XML entity*} es similar a una constante de un lenguaje de programación. Una constante tiene un nombre y un valor. Cada vez que se escribe el nombre de la constante, el compilador lo reemplaza por su valor. Una entidad XML es como una constante cuyo valor es cualquier trozo de texto: un carácter, un párrafo, un archivo (incluso binario), o todo un libro. A las entidades se les da nombre, y su valor se puede obtener mediante una **referencia de entidad XML** {*XML entity reference*}, con la notación `&entidad;`. Por ejemplo, `<` en un documento XML es reemplazado por el símbolo "<". Aunque XML define muchos tipos de entidades, en este documento sólo se estudiarán las *entidades predefinidas*.

Las **entidades predefinidas XML** {*predefined XML entities*} "son marcas que representan caracteres que de otro modo se interpretarían con un significado especial" (Goldfarb & Prescod, 1999, p. 358), como los caracteres menor que (<) y ampersand (&). Existen cinco entidades predefinidas mostradas en la Tabla 15. "Cuando el procesador XML analiza el documento, reemplaza las referencias de entidad por los caracteres reales", en lugar de interpretarlos como caracteres de marcado (Goldfarb & Prescod, 1999, p. 359).

Tabla 15. Entidades predefinidas en XML

Referencia de entidad	Valor	Descripción
`&`	&	*ampersand*
`<`	<	*less than*

Referencia de entidad	Valor	Descripción
`>`	>	*greater than*
`&apos;`	'	*apostrophe*
`&quot;`	"	*quotation mark*

Nótese que las *etiquetas* se delimitan con paréntesis angulares (`<` y `>`), y las *referencias a entidades* entre el signo `&` y el punto y coma (`;`). Se dice que el texto encerrado dentro de estos cuatro caracteres se conoce como **marcado XML** {*XML markup*}, lo restante como **datos de carácter XML** {CDATA, *XML character data*}. La combinación del marcado más los datos de carácter forman el **texto XML** {*XML text*} (Goldfarb & Prescod, 1999).

Ejercicio 28 [xml_markup_cdata, 10 pts]

Seleccione el texto del Listado 6 e imprima la selección en papel. Alternativamente puede imprimir a un PDF y utilizar las herramientas de anotación de un lector de PDF. Utilizando dos resaltadores distinga el *marcado XML* de los *datos de carácter XML* en el listado impreso. Agregue una leyenda indicando qué representa cada color. Todo el documento, desde el primer carácter hasta el último, debe quedar resaltado. Recuerde que el espacio en blanco también forma parte del documento.

Si imprimió en papel, escanee o tome una fotografía del resultado. Edite su imagen, especialmente sus dimensiones, para que ocupe poco espacio (menos de 200kB) en un formato de compresión de imágenes (por ejemplo, JPEG). Agregue su imagen o archivo PDF a su repositorio de control de versiones.

3.3.2. Secciones CDATA

Aunque el uso de entidades predefinidas parece sencillo, hace que el documento XML se vuelva difícil de leer y editar para el autor. Este fenómeno es evidente cuando las entidades se usan repetidamente, por ejemplo al escribir segmentos de código fuente. Por este motivo se crearon las *secciones CDATA*. Una **sección CDATA** {*XML CDATA section*} indica al procesador XML que no interprete una parte del texto aunque contenga marcado, es decir, que lo trate como *datos de carácter* {CDATA, *character data*}. La sintaxis de una sección de datos de carácter se aprecia en el Listado 11.

Listado 11. Sintaxis de una sección CDATA

```
<![CDATA[ contenido ]]>
```

En el `contenido` puede aparecer cualquier cadena de texto, excepto la que cierra la sección CDATA (`]]>`) llamada **CDEnd** (contracción de *character data end*). La línea 29 del Listado 6 usa dos entidades preestablecidas. Este mismo elemento puede escribirse con una sección CDATA. El Listado 12 muestra el elemento escrito de ambas formas.

Listado 12. Sintaxis de una sección CDATA

```
<p>Su popularidad puede deberse a su &quot;facilidad de uso&quot; y...</p>
<p><![CDATA[Su popularidad puede deberse a su "facilidad de uso" y...]]></p>
```

3.3.3. Comentarios

Los **comentarios** son cadenas de caracteres que no son parte de los datos de carácter, es decir, es parte del marcado pero que debe ser ignorado por el procesador XML. Permiten al autor hacer anotaciones que le ayuden a comprender o recordar partes del documento. El texto del comentario se introduce entre los caracteres inicio de comentario (`<!--`) y final de comentario (`-->`), como el Listado 13.

Listado 13. Sintaxis de un comentario XML

```
<!-- Esto es un comentario -->
```

Todo el texto dentro del inicio y fin de comentario será ignorado, incluyendo los caracteres de marcado (menor que, mayor que, *ampersand*, y punto-y-coma), excepto la secuencia de cierre del comentario (`-->`). Para algunos procesadores XML la secuencia de dos giones (`--`) es prohibida dentro de un comentario. Aunque no es obligatorio, se recomienda separar los dos guiones (`--`) del texto de los comentarios por un espacio en blanco, como se hizo anteriormente y no como el Listado 14:

Listado 14. Comentario incorrecto en XML

```
<!--Favor separarme con espacios-->
```

Ejercicio 29 [is_xml_valid, 5 pts]

¿Está el siguiente documento XML bien formado? Si su respuesta es negativa, señale los errores y las líneas en que aparecen:

Listado 15. ¿Está este documento XML bien formado? (is_well_formed.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>

<quiz name="os_quiz_1">
    <question name="bootstrap" type="single-choice">
        <text>¿A qué se refiere <i>bootstrap</i> del sistema operativo?
        <options>
            <option code=1>Nadie se puede colgar en sus propias botas</option>
            <option code=2>Una biblioteca de desarrollo web</option>
            <option code=3>El arranque del sistema operativo</option>
        </options>
    </question>
    <-- ------------------------------------------ -->
    <question name="bootstrap" type="textual" min-lenght="15">
        <text>¿Qué fenómeno del OS representan los filósofos comensales?</text>
    </question>
</quiz>
```

3.3.4. Espacios de nombres

Es natural querer combinar diferentes tipos de documentos. Por ejemplo, en una presentación incluir una fórmula o un gráfico, o en un documento incluir una hoja de cálculo (como una tabla). Si un documento XML se incrusta dentro de otro, y ambos utilizan un mismo identificador con propósitos distintos, habrá una colisión de nombres. Por ejemplo, el documento web del Listado 16 tiene una figura SVG incrustada entre las líneas 8 a 15. Ocurre una colisión con los elementos `<a>`: el de la línea 9 pertenece a la imagen SVG y el de la línea 17 pertenece a XHTML.

Listado 16. Ejemplo de una colisión de nombres (el elemento <a>) (namespace_collision.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html>

<html>
<body>
    <p>Dado un triángulo cualquiera:</p>
    <figure>
        <svg version="1.1">
            <a href="https://en.wikipedia.org/wiki/Heron%27s_formula">
                <path d="M20 20 L250 40 L100 150 Z" fill="orange" stroke="black"/>
            </a>
            <text x="40" y="90" font-family="Verdana">a</text>
            <text x="110" y="20" font-family="Verdana">b</text>
            <text x="175" y="110" font-family="Verdana">c</text>
        </svg>
    </figure>
    <p>Se puede calcular su área con la <a href="goo.gl/qjv2jt">fórmula de Herón</a>:</p>
    <p class="formula">s = sqrt(s(s-a)(s-b)(s-c)), s = (a+b+c)/2</p>
</body>
</html>
```

Para evitar colisiones se utilizan *espacios de nombres* (*namespaces*). Los **espacios de nombres** permiten distinguir los elementos que pertenecen a un tipo de documento de los que pertenecen a otro tipo de documento, llamados también vocabularios. Los espacios de nombres distinguen los elementos de diferentes vocabularios (tipos de documentos) agregando un prefijo a los identificadores de los elementos, de la forma `Prefijo:NombreElemento`. El Listado 17 distingue el vocabulario al que pertenecen todos los elementos del documento.

Listado 17. Todos los elementos de un documento y el espacio de nombres al que pertenecen (namespace_2.xml)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html>
3
4 <h:html xmlns:h="https://www.w3.org/1999/xhtml/" xmlns:s="https://www.w3.org/2000/svg">
5 <h:body>
6     <h:p>Dado un triángulo cualquiera:</h:p>
7     <h:figure>
8         <s:svg version="1.1">
9             <s:a href="https://en.wikipedia.org/wiki/Heron%27s_formula">
10                <s:path d="M20 20 L250 40 L100 150 Z" fill="orange" stroke="black"/>
11            </s:a>
12            <s:text x="40" y="90" font-family="Verdana">a</s:text>
13            <s:text x="110" y="20" font-family="Verdana">b</s:text>
14            <s:text x="175" y="110" font-family="Verdana">c</s:text>
15        </s:svg>
16    </h:figure>
17    <h:p>Se puede calcular su área con la <h:a href="goo.gl/qjv2jt">fórmula de Herón</h:a>:</h:p>
18    <h:p class="formula">s = sqrt(s(s-a)(s-b)(s-c)), s = (a+b+c)/2</h:p>
19 </h:body>
20 </h:html>
```

En el Listado 17 se dice explícitamente a qué vocabulario o espacio de nombres pertenece cada elemento del documento. Se usó el prefijo `h` para identificar a los elementos que pertenencen a XHTML y el prefijo `s` para aquellos que pertenecen a SVG. Estos prefijos se definen con el atributo `xmlns`, que es una contracción de ***xml n**amespace*. Recuerde que los atributos que inician con `xml` están reservados por el estándar.

Todos los elementos XML tienen un atributo `xmlns` implícito. Cuando el autor lo hace explícito, define un espacio de nombres de la forma: `xmlns:Prefijo="URI"`. El `Prefijo` es sólo un identificador de uso interno en el documento. El elemento donde aparece el atributo `xmlns` y todos sus hijos pueden utilizar el `Prefijo` para hacer explícito el espacio de nombres al que pertenencen. Es decir, el elemento y sus hijos pueden utilizar la notación `Prefijo:Elemento` en sus etiquetas de inicio y de cierre. Se usa un URI para identificar al espacio de nombres de forma única. El URI no es necesario que exista, pues el procesador XML no lo consulta. Sólo se debe usar consistentemente para identificar al espacio de nombres y evitar que colisione con espacios de nombres definidos por otros usuarios.

Se pueden definir tantos prefijos como vocabularios se usen en el documento. En el Listado 17 se definieron dos prefijos: `h` y `s`, ambos en el elemento raíz `html` de la línea 4. El atributo `xmlns` puede repetirse en un mismo elemento porque el prefijo se considera parte del nombre del atributo. El prefijo también forma parte del nombre de los elementos que lo utilicen. Con el uso de ambos espacios de nombres en el Listado 17, es claro para un software XML distinguir que el elemento `a` de la línea 9 es un enlace SVG, y que el elemento `a` de la línea 17 es un enlace `XHTML`.

Si al definir un espacio de nombres con el atributo `xmlns` se omite el prefijo, se estará indicando el espacio de nombres global. Todos los elementos hijos que no expliciten un prefijo pertenecen al espacio global. En la línea 4 del Listado 18 se indica que el espacio de nombres global es XHTML, a excepción de los elementos que inician con el prefijo `s`, pues son elementos SVG. El espacio de

nombres `s` se definió en el elemento `svg` de la línea 8, pero es más usual hacerlo en el elemento raíz como es el caso de `xhtml` en la línea 4. Este ejemplo muestra otro detalle. Los enlaces `a` de SVG usan el estándar XLink, el cual requiere su propio espacio de nombres (línea 9).

Listado 18. Define el espacio de nombres global como XHTML, y el espacio de nombres `s` como SVG (namespace_3.xml)

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE html>
3
4  <html xmlns="https://www.w3.org/1999/xhtml/">
5  <body>
6      <p>Dado un triángulo cualquiera:</p>
7      <figure>
8          <s:svg version="1.1" xmlns:s="https://www.w3.org/2000/svg"
9              xmlns:xlink="http://www.w3.org/1999/xlink">
10             <s:a xlink:href="https://en.wikipedia.org/wiki/Heron%27s_formula">
11                 <s:path d="M20 20 L250 40 L100 150 Z" fill="orange" stroke="black"/>
12             </s:a>
13             <s:text x="40" y="90" font-family="Verdana">a</s:text>
14             <s:text x="110" y="20" font-family="Verdana">b</s:text>
15             <s:text x="175" y="110" font-family="Verdana">c</s:text>
16         </s:svg>
17     </figure>
18     <p>Se puede calcular su área con la <a href="goo.gl/qjv2jt">fórmula de Herón</a>:</p>
19     <p class="formula">s = sqrt(s(s-a)(s-b)(s-c)), s = (a+b+c)/2</p>
20 </body>
21 </html>
```

3.4. La definición del tipo de documento (DTD)

La **definición de tipo de documento** {DTD, *Document Type Definition*} es "el conjunto de normas XML para la representación de documentos de un determinado tipo" (Goldfarb & Prescod, 1999, p. 49). El DTD describe la gramática del documento (W3C, 2008), es decir, los tipos de elementos, sus atributos, entidades, algunas restricciones, y cómo se relacionan entre ellos. Esta sección explica la sintaxis que XML utiliza para representar estos conceptos y sus relaciones en un DTD.

Un documento XML que respete cabalmente las reglas establecidas en su DTD se dice que *es válido*. A modo de ejemplo, el Listado 19 muestra el DTD referido en el libro del Listado 19 llamado `book.dtd`. Un DTD se compone principalmente de definiciones de tipos de elementos, de tipos de atributos, y de entidades, y se estudian a continuación.

Listado 19. Ejemplo de una definición de tipo de documento (DTD) para representar libros (book.dtd)

```
<!ENTITY % SectionType
   "forework|preface|acknowledgements|dedication|contents">

<!ELEMENT book (title, authors, front-cover?, chapter+, back-cover?)>
<!ATTLIST book
   name ID #REQUIRED
   version CDATA #REQUIRED>

<!ELEMENT title (#PCDATA)>

<!ELEMENT authors (author+)>

<!ELEMENT author (fullname, email?)>
<!ATTLIST author
   name ID #REQUIRED
   percent CDATA #IMPLIED>

<!ELEMENT fullname (#PCDATA)>

<!ELEMENT email (#PCDATA)>

<!ELEMENT front-cover EMPTY>
<!ATTLIST front-cover
   img CDATA #REQUIRED>

<!ELEMENT chapter (title?, (p|section)+)>
<!ATTLIST chapter
   name ID #REQUIRED
   type (prelims|contents) "contents">

<!ELEMENT p (#PCDATA|em|strong|code)*>

<!ELEMENT em (#PCDATA)>
<!ELEMENT strong (#PCDATA)>
<!ELEMENT code (#PCDATA)>

<!ELEMENT section (title?, (p|section)+)>
<!ATTLIST section
   name ID #REQUIRED
   type (%SectionType;) "contents"
   author IDREF #IMPLIED>

<!ELEMENT back-cover EMPTY>
<!ATTLIST back-cover
   img CDATA #REQUIRED>
```

3.4.1. Declaraciones de tipo de elemento

En el DTD se escriben las **declaraciones de tipo de elemento XML** {*XML element type declarations*} que son restricciones que deben respetar todos los elementos en un documento XML válido. Estas declaraciones tienen la sintaxis del Listado 20.

Listado 20. Sintaxis de la declaración de tipo de elemento

```
<!ELEMENT IdElemento EspecContenido>
```

El `<!ELEMENT` inicia la *declaración de un tipo de elemento*. El `IdElemento` es el identificador que aparece en sus etiquetas. Debe ser único en el DTD y respetar las restricciones de los identificadores XML. La `EspecContenido` se refiere a la **especificación de contenido**, que indica qué objetos pueden figurar en el contenido del elemento, hay cuatro posibilidades listadas en la Tabla 16 (Goldfarb & Prescod, 1999):

Tabla 16. Especificación de contenido de elementos XML

Tipo	Descripción
`EMPTY`	Impide que el elemento tenga contenido y por tanto sus etiquetas son vacías. Ejemplo: `front-cover` en la línea 22 en el DTD del Listado 19.
`ANY`	Puede contener cualquier elemento o carácter, lo cual no se recomienda ya que no es estructurado. Pueden ser útiles cuando se está escribiendo el DTD para lograr que los documentos sean válidos mientras se está refinando dicho DTD.
element-content	Sólo puede contener los subelementos listados dentro de paréntesis en la especificación de contenido. Por ejemplo, la línea 26 del Listado 19 indica que los elementos `chapter` sólo pueden tener un título opcional y varios párrafos o secciones.
mixed-content	Permite al elemento contener subelementos, datos de carácter (`#PCDATA`), o ambos, como ocurre con el elemento de párrafo `p` en la línea 31 del Listado 19.

Cuando un elemento tiene subelementos (o elementos hijos) en el contenido (últimos dos casos de la lista anterior), es necesario especificar un **modelo de contenido XML** {*XML content model*}, que establece el orden y número de apariciones de los subelementos. El orden de los elementos hijos se establece con los caracteres coma (`,`), barra vertical (`|`) y paréntesis redondos. Una secuencia de subelementos separados por coma indica que tales subelementos deben aparecer y en el mismo orden. La barra vertical indica que debe aparecer sólo uno de los dos subelementos que están en sus extremos. Los paréntesis agrupan partículas de contenido que son tratadas como unidades por los operadores anteriores, lo que permite combinarlos.

El número de apariciones de los subelementos se especifica con los *indicadores de frecuencia* que se escriben al final del subelemento y sin espacios en blanco. Existen tres indicadores de frecuencia. El signo de interrogación (`?`) indica que el elemento hijo es opcional, es decir, puede o no puede aparecer. El asterisco (`*`) indica que el subelemento puede aparecer 0 ó más veces (opcional y repetible). El signo de más (`+`) indica que el subelemento debe aparecer 1 o más veces (necesario y repetible) (Goldfarb & Prescod, 1999).

Ejercicio 30 [xml_inventory_5, 15 pts]

Escriba la definición de tipo de documento para su inventario, es decir un archivo DTD en la misma carpeta donde se encuentra su inventario XML. Escriba en su DTD una declaración de tipo de elemento por cada uno de los que componen su inventario XML. Su especificación de contenido debe permitir jerarquías de activos de cualquier nivel de profundidad. Es decir, los activos deben poderse anidar.

Ejercicio 31 [xml_inventory_6, 10 pts]

Utilice comentarios en su DTD para explicar a un lector que esté empleando su especificación, qué significa cada elemento y sus atributos.

3.4.2. Declaraciones de la lista de atributos

Los *atributos* "son una forma de incorporar características o propiedades a los elementos de un documento" (Goldfarb & Prescod, 1999, p. 353). Un elemento puede tener una lista de atributos, la cual se declara en el DTD en la sección conocida como **declaración de lista de atributos**, usualmente después de la declaración del tipo de elemento. Tiene la sintaxis del Listado 21.

Listado 21. Sintaxis de la declaración de la lista de atributos

```
<!ATTLIST IdElement
    IdAtt1 AttType1 DefaultValue1
    IdAtt2 AttType2 DefaultValue2
    ...>
```

El `<!ATTLIST` indica que se va a declarar la lista de atributos del elemento identificado por `IdElement`. Los atributos continúan uno tras otro separados por espacios en blanco, sin impotar si son cambios de línea o no. Cada declaración de atributo se compone de un identificador (`IdAttN`), el tipo de datos (`AttTypeN`), y el valor por defecto (`DefaultValueN`). `IdAttN` indica el nombre del atributo, debe ser un identificador válido y no es obligatorio que sea único en el documento. El tipo del atributo `AttTypeN` puede ser alguno de los listados en la Tabla 17.

Tabla 17. Tipos de atributos en un DTD

Tipo	Descripción
`CDATA`	Datos de carácter {*character data*}. Permite casi cualquier cadena de caracteres.
`NMTOKEN`	*Name token*. Sólo permite letras, números y algunos caracteres especiales, como los que se utilizan para declarar identificadores, pero no obliga a que sea único en el documento.
`NMTOKENS`	Una lista de *name tokens*.

Tipo	Descripción
`(a|b|c|d)`	Enumerados. El atributo sólo puede tomar uno de los valores indicados en la lista dentro de paréntesis, los cuales se separan por barras verticales (`|`); por ejemplo el atributo type en la línea 29 del Listado 19.
`ID`	Identificador. Declara que el atributo es un identificador único, es decir, su valor debe cumplir con las restricciones de los identificadores y en el documento XML no puede haber dos o más atributos con el mismo valor.
`IDREF`	Referencia a un identificador. Declara que el atributo tiene como valor una referencia a un identificador existente. Su valor puede repetirse muchas veces en el documento XML pero debe respetar las restricciones de los identificadores.
`ENTITY`	Referencia a una entidad (o atributos de entidad). El valor del atributo debe ser el nombre de una entidad existente en el documento XML.

El `DefaultValueN` en la declaración de la lista de atributos del Listado 21 indica si el atributo se puede omitir o no, y el valor por defecto en caso de que se omita. Puede tomar las tres variantes listadas en la Tabla 18.

Tabla 18. Valores por defecto de atributos en un DTD.

Tipo	Descripción
`#REQUIRED`	Indica que el atributo es requerido y no se puede omitir, por tanto no tiene un valor por defecto. Son ejemplos `name` y `version` de `book` en el Listado 19.
`value`	Un valor que está en el dominio de valores permitido por el tipo de atributo. Indica que ese será el valor que el procesador XML tome si el autor no especifica uno en el documento. Por ejemplo, según la línea 29 del Listado 19, si el autor no especifica el tipo del capítulo, el procesador XML supondrá que se trata de un capítulo de contenido y no uno preliminar.
`#IMPLIED`	Permite al autor omitir el valor del atributo pero sin forzar a escribir un valor por defecto determinado. El procesador XML asignará algún valor que considere adecuado o lo ignora.

Ejercicio 32 [xml_inventory_7, 10 pts]

Para cada elemento que recibe atributos, escriba en su archivo DTD una declaración de lista de atributos. Trate de usar atributos identificadores (`ID`) y valores enumerados (`a|b|c|d`) siempre que sea posible. Al menos debe hacerlo en los dos siguientes casos:

1. Utilice valores por defecto, de tal forma que el sistema pueda suponer los valores más comunes para algunos atributos de los activos, lo cual ahorra memoria en el documento XML y hace más rápido el procesamiento (parsing) del mismo.
2. Obligue al analizador XML {*XML parser*} a considerar como no válido a un inventario que contenga activos a cargo de empleados que no existen en la compañía.

Ejercicio 33 [xml_inventory_8, 10 pts]

Asegúrese de que su documento XML es válido ante el DTD que escribió. Utilice un analizador XML genérico {*XML parser*}. A modo de sugerencia puede emplear el Gnome XML C parser. Si utiliza alguna distribución de Linux instale el paquete `libxml2-utils`, y emita el comando

```
xmllint --valid --noout /path/to/file.xml
```

Si su documento XML es válido, el comando anterior no generará salida en pantalla. Si utiliza Windows, puede descargar los binarios provistos por Igor Zlatkovic, descomprimirlos y mover los archivos de las subcarpetas `bin\` a una carpeta incluida en su variable ambiente `%PATH%`. Luego puede utilizar una línea de comandos para emitir el comando previo. Si su editor de texto o ambiente de desarrollo tiene extensiones, puede instalar una para XML. Por ejemplo, Notepad++ provee *XML Tools Plugin*, el cual agrega opciones de menú para validar desde el editor.

Avance de proyecto 20 [prj_message_passing_dtd, 15 pts]

Cree una definición de tipo de documento (DTD) para su protocolo de paso de mensajes en XML. Defina todos los tipos de elementos, sus contenidos, y sus atributos. Tenga presente a la hora de escoger identificadores y el nombre del tipo de documento, que la intención de su DTD es permitir documentar el paso de mensajes durante las sesiones de la aplicación web. Es decir, el contenido de un documento XML que se apegue a este tipo de documento es un historial de intercambio de mensajes.

Modifique el archivo `design/messages1.xml` para declarar su tipo de documento y referir el DTD recién creado. Asegúrese de que el documento `design/messages1.xml` sea válido de acuerdo a las reglas del DTD. Utilice un analizador XML para este fin (Véase el ejercicio `xml_inventory_8` para un ejemplo de análisis). Repita estos pasos si tiene más archivos `design/messagesN.xml`.

3.4.3. Elementos versus atributos

No es trivial decidir qué aspectos del mundo real modelar como elementos y cuáles como atributos. El autor ha de escoger alguna convención que sea coherente y escribir un DTD para obligar a todos los documentos del mismo tipo a cumplir con el diseño escogido. Aunque no hay ninguna regla para decidir qué debe modelarse como un elemento o un atributo, Goldfarb y Prescod sugieren las siguientes ideas.

Los atributos no pueden contener "subatributos" ni elementos, sino que son pequeños trozos de texto sin estructura o listas de condiciones. Por tanto, si algo del mundo real contiene otras partes que también necesitan modelarse, debe hacerse con elementos y no con atributos. Los atributos se usan para añadir poca información, sencilla y sin estructura (Goldfarb & Prescod, 1999).

Los elementos deben respetar el orden y el número de ocurrencias descritos en el DTD, mientras que los atributos pueden aparecer en cualquier orden y no pueden repetirse. Por esto, las "cosas" que deben repetirse o que respetan cierto orden deben modelarse como elementos y no como atributos

(Goldfarb & Prescod, 1999).

Si las ideas anteriores no son criterio suficiente, puede usarse la siguiente heurística. Los elementos se utilizan para representar partes de los objetos, o datos que son parte del contenido principal, y deben aparecer en las impresiones o interpretaciones del documento. Los atributos se utilizan para representar propiedades de los objetos, es decir, información sobre los datos reales (metadatos) y que no necesariamente se imprime (Goldfarb & Prescod, 1999).

Ejercicio 34 [xml_inventory_9, 25 pts]

Su especificación para representar inventarios debe ser genérica. Es decir, muchas empresas de distinta naturaleza podrían utilizar su notación para representar sus respectivos inventarios, y su DTD debe ser capaz de representarlos sin tener que incluirle modificaciones. De esta forma, usted puede publicar su DTD en algún servidor web y quienes empleen su especificación, incluirían el DTD oficial en sus documentos XML (tal como ocurre con el DTD de XHTML, por ejemplo).

Para probar la generalidad de su DTD, cree otro documento XML y represente el inventario de la Figura 24. Verifique que su documento es válido ante su DTD.

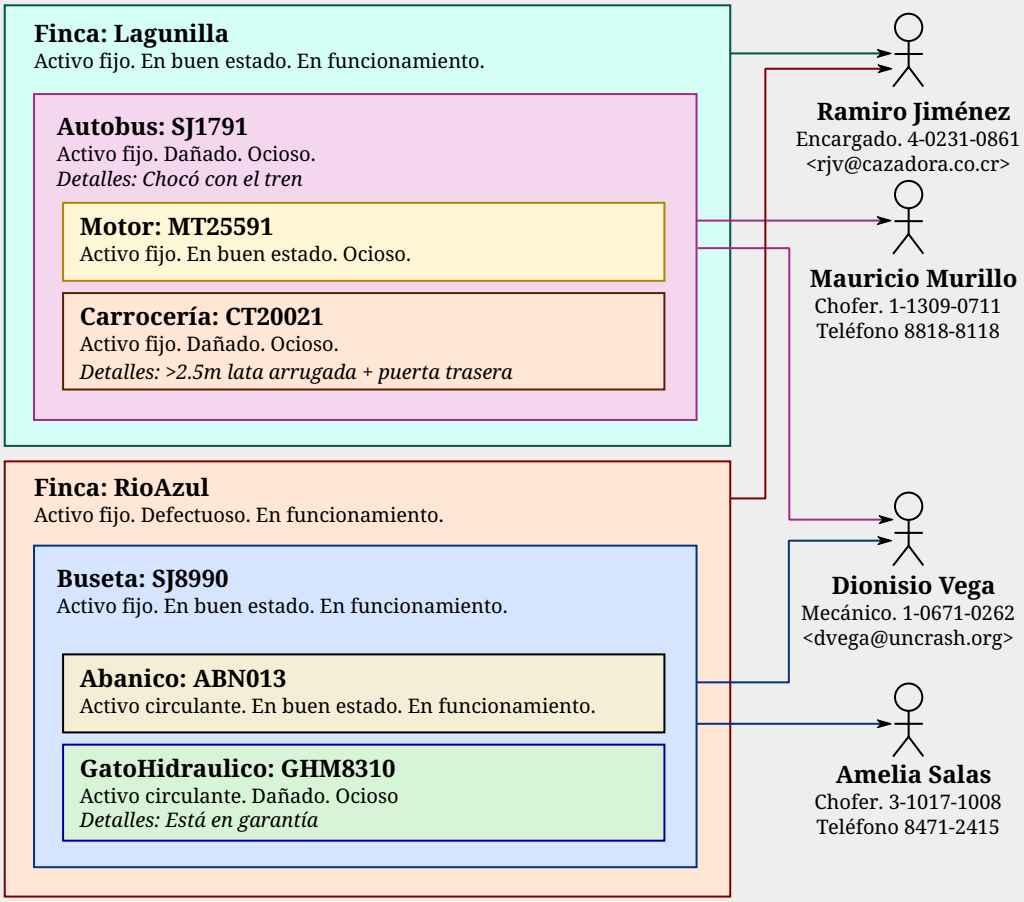


Figura 24. Inventario reducido de una compañía hipotética de transportes

Ejercicio 35 [xml_university_database_1, 25 pts]

Represente en un documento XML la base de datos Universidad cuyo estado se encuentra en las siguientes tablas y su estructura en el modelo entidad-relación de la Figura 25.

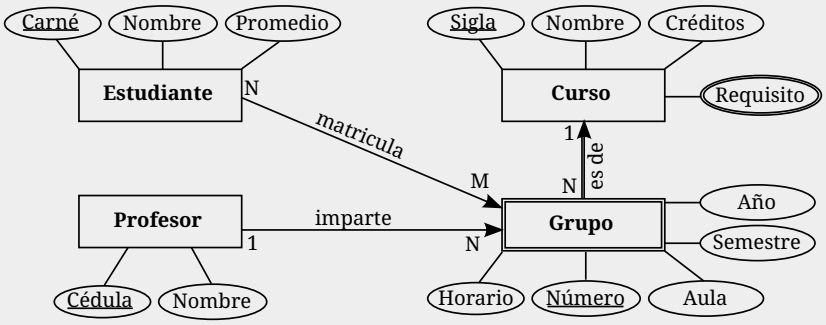


Figura 25. Una base de datos Universidad

Tabla 19. Estudiante

Carné	Nombre	Promedio
a45891	Eliécer Montero	7.14
a98765	Yanet Arias	8.96
b10976	Anais Solano	8.30

Tabla 20. Profesor

Cédula	Nombre
109830876	Edgar Casasola
208431187	Maureen Murillo
135900821	Francisco Arroyo
110101101	Jeisson Hidalgo

Tabla 21. Curso

Sigla	Nombre	Créditos
CI-1101	Programación I	4
CI-1201	Programación II	4
CI-1320	Redes de Computadoras I	5
CI-2413	Desarrollo de aplicaciones web	4

Tabla 22. Requisito

Curso	Requisito
CI-1201	CI-1101
CI-2413	CI-1201
CI-2413	CI-1320

Tabla 23. Grupo

Curso	Numero	Anno	Semestre	Horario	Aula	Profesor
CI-1101	04	2011	1	KV13-15	301IF	109830876
CI-1101	08	2011	1	KV07-09	304IF	208431187
CI-1320	01	2011	1	KV17-19	303IF	135900821
CI-2413	01	2011	1	KV15-17	305IF	110101101

Tabla 24. Matricula

Estudiante	Curso	Grupo
a45891	CI-1101	04
a45891	CI-1320	01
a98765	CI-1320	01
b10976	CI-2413	01

Ejercicio 36 [xml_university_database_2, 15 pts]

Escriba una definición de tipo de documento para representar la base de datos Universidad. Asegúrese de que el documento que escribió en el ejercio anterior sea válido ante este DTD.

Ejercicio 37 [xml_university_database_3, 10 pts]

Si no lo hizo, trate de desnormalizar las relaciones `Requisito` y `Matricula` en su documento XML de la base de datos Universidad. Es decir, estas relaciones no deben aparecer como una lista de elementos, sino como hijos de otros elementos. Actualice el DTD y asegúrese de que el nuevo documento XML sea también válido. ¿Se puede también desnormalizar la entidad débil `Grupo`?

3.4.4. Documentación de los elementos

En la Sección 3.2.3 se indicó que los creadores de un tipo de documento tienen en su mente la semántica de los elementos y atributos. Cuando otras personas leen o necesitan escribir un

documento del mismo tipo en XML, enfrentarán dificultades si desconocen esta semántica. Por ejemplo, un elemento `<T>` podría significar un texto, una tupla, una tabla, un título, u otro concepto. XML es un conjunto de reglas estándar para marcar documentos, pero no ofrece funcionalidad para describir la semántica de los elementos y sus atributos. Los autores elaboran manuales en páginas web, PDF, u otros formatos. Una práctica común y no excluyente, es plasmar la semántica como comentarios en el tipo de documento.

Documentar un tipo de documento consiste en incluir comentarios, usualmente detallados y extensos, con la semántica que otros autores necesitarán para poder comprender, y escribir documentos válidos de ese tipo. Un extracto de un DTD documentado se aprecia en el Listado 22. Un primer comentario contextualiza el tipo de documento, su propósito, los autores, versiones o revisiones, u otra información general. Luego se provee un comentario para cada elemento, el cual describe el elemento, provee un ejemplo, documenta cada atributo, y puede dar sugerencias sobre el contenido esperado.

Listado 22. Extracto de documentación de un DTD

```
<!--
Document Type Definition for XML Book documents
v1.1 2020-05-27 Jeisson Hidalgo-Cespedes <jeisson.hidalgo@ucr.ac.cr>
-->

<!-- A book is sliced in parts, chapters, and sections. Most sections are content
sections in a book, but some are special. The SectionType entity is an enumeration
of the type of sections that a book can have, such as preface or dedication. -->
<!ENTITY % SectionType
   "forework|preface|acknowledgements|dedication|contents">

<!-- A book is a textual or visual play distributed along several pages that are glued,
numerated, and protected by a cover. The book element represents the document, e.g:

    <book name="webappdev" version="1.1.17">

A book has two attributes:

name:
    An identifier used by book applications to distinguish a book from others. It may
    be used as the name of the xml file, for example 'webappdev.xml'.

version:
    A book may change on time. The version attribute allows authors to distinguish
    several versions of the same book. Authors decide their own version scheme. E.g:
    version="3.2.153" may refer the third edition, second printing, and 153 revision.

A book must contain a title, authors, and chapters. Authors may provide a front-cover
and a back cover.
-->
<!ELEMENT book (title, authors, front-cover?, chapter+, back-cover?)>
<!ATTLIST book
   name ID #REQUIRED
   version CDATA #REQUIRED>
```

Avance de proyecto 21 [prj_message_passing_doc, 10 pts]

Documente los elementos de su protocolo de paso de mensajes en el archivo DTD, como se hizo en el Listado 22. Asegúrese de proveer una descripción general del tipo de documento, documentar todos los elementos y sus atributos. Mientras documenta cada elemento y atributo revise si los identificadores escogidos son significativos. Una vez finalizada la documentación, verifique que los documentos XML sigan siendo válidos.

3.4.5. Entidades XML

Disponible en la edición completa de este material.

3.5. La familia de tecnologías XML

Las secciones anteriores discutían sobre el núcleo de la especificación del XML 1.0, llamado **XML Core**, El W3C y otras organizaciones continuaron extendiendo la especificación original, definiendo estándares que en conjunto se conocen como la **familia de tecnologías XML** [Arci02]. Las siguientes secciones resumen algunas de estas familias para modelar y validar (*XML Schema*), ligar (*XLink*), apuntar (*XPointer*), controlar la presentación y transformar (*XSLT*), y formatear objetos (*XSL-FO*). Finalmente se presentan las dos interfaces de programación provistas por los analizadores de XML {*XML parsers*}: *Simple API for XML* (SAX) y *Document Object Model* (DOM).

Disponible en la edición completa de este material.

Capítulo 4. El lenguaje de marcado de hipertexto (X)HTML

El **lenguaje de marcado de hipertexto** {HTML, *Hypertext Markup Language*} es una notación estándar para escribir el contenido de las páginas web. Nació como una aplicación del *lenguaje de marcado generalizado estándar* {SGML, *Standard Generalized Markup Language*} en 1991 para escribir documentos científicos (WHATWG, 2020). Un gran volumen de estandarización fue la versión 4.0 de 1997. En 1998 el W3C publicó una simplificación de SGML llamada XML. En el 2000 el W3C publicó XHTML 1.0 como una adaptación de HTML sobre XML y actualizó HTML a 4.01 con el fin de que fuesen cercanamente compatibles. Después de ello, el W3C centró su atención en XHTML2 descontinuando HTML, hasta que un grupo independiente de interesados trabajaron en una especificación que se convertiría en (X)HTML5 compatible tanto con HTML 4.01 como XHTML 1.0.

La persona desarrolladora web se puede estar preguntando cuál de estos dos lenguajes escoger para un sitio en que va a trabajar. Necesitará entender las diferencias para fundamentar su decisión. HTML es bastante liberal, mientras XHTML es inherentemente estricto. El Listado 23 muestra un trozo de código HTML y el Listado 24 el mismo contenido en XHTML. Las principales diferencias entre estos dos lenguajes son las siguientes.

1. Los identificadores en HTML no son sensitivos a mayúsculas ni minúsculas, mientras que en XHTML lo son, y los definidos por la recomendación usan minúsculas consistentemente.
2. HTML permite dejar elementos sin cerrar como `p` y `li`, otros nunca se cierran como `br` e `img`, mientras que en XHTML todo elemento debe cerrarse obligatoriamente.
3. HTML permite **minimizar atributos** {*attribute minimization*}, esto es, omitir las comillas alrededor de los valores de atributos que sólo contienen letras, números o los caracteres guión (`-`), punto (`.`), guión bajo (`_`) o dos puntos (`:`). Por su parte, XHTML exige que todo valor de atributo debe estar encerrado entre comillas dobles o simples.

1. HTML permite que algunos atributos no tengan valor como `disabled` en `<textarea disabled>`, la presencia del atributo es suficiente para generar su efecto. XHTML exige que todo atributo tenga un valor entre comillas, y para estos casos, el Consorcio Web tomó la convención de repetir el nombre del atributo en su valor, ejemplo: `<textarea disabled="disabled">` [Cast06].
2. Los elementos `html`, `head` y `body` son opcionales en HTML, pero obligatorios en XHTML. Sin embargo, es una mala práctica omitirlos en HTML.
3. XHTML puede tener secciones CDATA, mientras que HTML no.

Listado 23. Un documento HTML 4.01 strict válido (pure_html.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=utf-8">
<TITLE>Mis libros</title>

<P>Compre uno de mis libros:
<UL>
   <LI><IMG SRC=cpp.png ALT="C++"> Ejercicios intrincados en C++
   <li><Img Src=mate.png Alt=Mate>El chofer del bus es mejor en mate que yo
</uL>
```

Listado 24. Un documento XHTML 1.0 strict válido (pure_xhtml.xhtml)

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8"/>
   <title>Mis libros</title>
</head>

<body>
   <p>Compre uno de mis libros:</p>
   <ul>
      <li><img src="cpp.png" alt="C++"/> Ejercicios intrincados en C++</li>
      <li><img src="mate.png" alt="Mate"/>El chofer del bus es mejor en mate que yo</li>
   </ul>
</body>
</html>
```

Es evidente que escribir HTML es más relajado que XHTML, si ambos producen el mismo efecto en el navegador ¿por qué molestarse con XHTML? HTML obliga al navegador a programar excepciones y reglas circunstanciales que dilatan el procesamiento y propician a la ambigüedad. XHTML obliga a una sintaxis restringida que facilita al navegador y otros software XML a interpretar código fácil y eficientemente. Además XML abre un conjunto de funcionalidades y tecnologías que no están disponibles para HTML, como la posibilidad de incrustar otras especificaciones como SVG y MathML en los documentos XHTML.

Si una persona piensa escribir algunas páginas manualmente, quizá HTML sea mejor alternativa. Si planea que esas páginas formen parte de un sitio web grande, sean almacenadas en bases de datos, o procesadas por algún tipo de software, XHTML ofrecerá más ventajas. En general se cumple que código XHTML válido es también código HTML válido, o al menos con pocas modificaciones, pero no el recíproco. Por eso este capítulo conferirá énfasis a XHTML y las diferencias importantes con HTML

se resaltarán en su contexto.

Ejercicio 38 [html_to_xhtml, 10 pts]

El documento del Listado 25 es HTML válido. Conviértalo en un documento XHTML 1.0 estricto válido. Comprueba que su documento resultante sea válido, usando un analizador XML {*XML parser*}. La Sección 3.4 provee detalles de cómo validar un documento XML ante un DTD.

Listado 25. Un documento HTML 4.01 strict válido (convert_html_xhtml.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=utf-8">
<TITLE>Conversión HTML a XHTML</title>

<H1>Conversión HTML a XHTML</H1>
<OL>
    <LI>Cambie todos los identificadores a minúsculas.
    <LI>Asegúrese de que todos los elementos estén cerrados.
    <LI>Agregue comillas a todos los valores de atributos.<BR>
        Si hay atributos minimizados, repita su valor entre comillas.
    <LI>Agregue los elementos <CODE>html</CODE>, <CODE>head</CODE> y <CODE>body</CODE> si no
lo están.
</oL>
<HR>
<H1>Conversión XHTML a HTML</H1>
<OL>
    <LI>No cierre los elementos que se componen de sólo una etiqueta: <CODE>img</CODE>, <
CODE>br</CODE>, <CODE>hr</CODE>, ...
    <LI>Si las hay, elimine las secciones CDATA y convierta los caracteres especiales (<
CODE>&lt;</CODE>, <CODE>&gt;</CODE>,<CODE>&amp;</CODE>) en sus respectivas referencias de
entidad (<CODE>&amp;lt;</CODE>, <CODE>&amp;gt;</CODE>, <CODE>&amp;amp;</CODE>).
</oL>
<HR>
<P><BUTTON DISABLED>Cerrar</BUTTON>
```

4.1. Estructura global

Un **documento web** es muy similar a cualquier otro documento que una persona podría escribir en un procesador de palabras, y por ende tiene títulos, párrafos, tablas, imágenes, enlaces, estilos y otros elementos familiares. La principal diferencia es que un documento web debe seguir estrictamente la sintaxis de (X)HTML. Al hacerlo, cualquier software (X)HTML podrá manipular el documento, sea un navegador, un programa de diseño, un motor de bases de datos, y hasta el mismo procesador de palabras mencionado anteriormente. Dado que el navegador es el más común, se usará en este texto por claridad en lugar de *software (X)HTML*.

Un documento web se compone de cuatro partes: la declaración del tipo de documento, el elemento documento, el encabezado y el cuerpo. El Listado 26 muestra un documento minimalista XHTML5 con estas cuatro partes. El Listado 28 muestra el mismo documento en la versión XHTML 1.0 del 2000, aún en uso de acuerdo a estudios de Catalin Rosu y W[3]Techs. En HTML las cuatro secciones están presentes, pero a diferencia de XHTML los elementos `<head>` y `<body>` no son requeridos. El Listado 27 muestra el documento mínimo válido en HTML5 y el Listado 29 en *HTML 4.01 Strict* también del 2000 y aún en uso de acuerdo a los mismos estudios.

Listado 26. Mínimo documento válido en XHTML5 (min_5.xhtml)

```
<!DOCTYPE html>
<html lang="es" xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta charset="utf-8"/>
        <title>Título del documento</title>
    </head>
    <body>
        <!-- Cuerpo del documento -->
    </body>
</html>
```

Listado 27. Mínimo documento válido en HTML5 (min_5.html)

```
<!DOCTYPE html>
<html lang="es">
    <meta charset="utf-8"/>
    <title>Título del documento</title>
    <!-- Cuerpo del documento -->
</html>
```

Listado 28. Mínimo documento válido en XHTML 1.0 (min_1.xhtml)

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8" />
    <title>Título del documento</title>
</head>

<body>
    <!-- Cuerpo del documento -->
</body>
</html>
```

Listado 29. Mínimo documento válido en *HTML 4.01 Strict* (min_401.html)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">

<html lang="es">
    <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8">
    <title>Título del documento</title>
    <p>El elemento <code>body</code> es opcional en HTML.</p>
</html>
```

4.1.1. La declaración del tipo de documento web

La declaración del tipo de documento, expresada con la etiqueta `<!DOCTYPE ⋯>`, informa al navegador el tipo de documento (HTML o XHTML) y la versión usada en el resto del documento. Con esta declaración el navegador sabrá cómo interpretar la sintaxis y contra cual especificación validar su código. La tabla Tabla 25 muestra tipos de documentos y versiones en uso. Para una lista completa que incluye tipos de documentos para XHTML-Basic y XHTML Mobile Profile, puede consultarse la entrada de tipos de documento en la Wikipedia.

Tabla 25. Tipos de documento (X)HTML de acuerdo a su versión y variación

Language	Ver.	Variation	Document type
HTML	4.01	strict	`<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">`
		transitional	`<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">`
		frameset	`<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN" "http://www.w3.org/TR/html4/frameset.dtd">`
XHTML	1.0	strict	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`
		transitional	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
		frameset	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">`
	1.1	—	`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">`
(X)HTML	5.0	—	`<!DOCTYPE html>`

Aunque en HTML es opcional, siempre es conveniente declarar el tipo de documento. Es difícil memorizarlos, por lo que el WHATWG decidió simplificar la declaración para (X)HTML5 a `<!DOCTYPE html>`, que es la versión recomendada para nuevos documentos (véase Listado 26 y Listado 27).

Si se omite la declaración del tipo de documento la mayoría de navegadores entran en ***quirks mode***, un modo de compatibilidad para poder desplegar páginas web obsoletas que contienen errores {*quirks*}, con resultados que eran dependientes del navegador. Si la declaración del tipo de documento

está presente, el navegador entra en **modo estándar** {*standard browser mode*}, y si el documento es válido, el comportamiento debería ser homogéneo entre navegadores.

Ejercicio 39 [html_body_text, 5 pts]

Descargue un DTD de HTML 4.01 ó XHTML 1.1 y averigüe si es permitido o no escribir texto puro directamente en el contenido del elemento `<body>`? Justifique su respuesta.

Existen varias herramientas gratuitas de aseguramiento de la calidad {QA, *Quality Assurance*} que permiten someter un documento web a un conjunto de pruebas y obtener retroalimentación de si está bien formado, se conforma al tipo de documento que declara, no tiene enlaces rotos, sus estilos son válidos y otras pruebas. Algunos ejemplos de estas herramientas son los siguientes.

1. W3C Markup Validation Service. Es parte del juego de herramientas de control de calidad del Consorcio Web. Permite validar un documento HTML o XHTML. Tiene soporte experimental para (X)HTML5.
2. El W3C Unified Validator (Unicorn) también del Consorcio Web, permite validar tanto un documento (X)HTML como sus estilos (CSS). Tiene soporte para CSS 3.
3. Validator.nu. Un conjunto de validadores que utilizan varios esquemas (además de DTD) para aseguramiento de la calidad. Es especialmente útil para (X)HTML5.

Existen otros validadores disponibles no mencionados en la lista anterior. Un código que resulte ser válido tras someterse a varios validadores, se considerará de mejor calidad y con más garantía de funcionar apropiadamente entre navegadores.

Ejercicio 40 [dtd_statistics, 10 pts]

Haga estadísticas. Someta al validador del W3C al menos diez páginas web que frecuenta (por ejemplo, las que tiene registradas en sus favoritos). ¿Cuántas de ellas son válidas? ¿Qué tipos de documentos (DTD) son los menos y más usados? ¿Reflejan sus resultados estadísticas generadas por otros autores (por ejemplo: 90% HTML, 10% XHTML)?

Ejercicio 41 [validity_fix, 5 pts]

Seleccione dos páginas web que sometió al validador del W3C en el ejercicio anterior y no se conformaron al estándar que declaran. Si usted tuviera que corregirlas, describa rápidamente qué cambios tendría que hacerle a cada una de ellas.

Cuando escriba un documento HTML, almacénelo con extensión `.html` y los documentos XHTML con extensión `.xhtml`. Aunque la extensión no debería ser algo de importancia, de forma lamentable, parte software web se guía por este detalle en lugar de la declaración del tipo de documento (incluso los validadores del W3C). Si aloja los documentos en un servidor web, es probable que además deba configurarlo para que sirva los archivos XHTML con el MIME {*Multipurpose Internet Mail Extensions*} `application/xhtml+xml`. Aunque ya es de poco uso, Internet Explorer 8 y anteriores no despliegan páginas XHTML sino que las ofrecen al usuario para que las guarde en disco.

(X)HTML5 pretende simplificar la web de un modo pragmático. Aunque aquí se incluye el "5" en "(X)HTML5" para diferenciarlo de sus predecesores, sus autores lo llaman realmente "HTML, el estándar vivo", sin ningún número de versión, y por eso no se especifica en el `DOCTYPE`, ni tampoco su DTD como se aprecia en la línea 1 del Listado 26 y del Listado 27, lo cual hace que deje de ser una aplicación XML válida. Por esto se dice que XHTML5 no existe como un estándar XML. El estándar es HTML5, el cual permite utilizar tanto la notación de HTML como XML, pero sin mezclarlas. Por esto XHTML5 no es una aplicación XML como sí lo es XHTML 1.0. XHTML5 simplemente es un término para referirse al HTML5 que utiliza notación XML.

Dado que XHTML5 no es una aplicación XML, los navegadores ignorarán la declaración XML (`<?xml ⋯ ?>`) en la primera línea del documento, por lo cual es común omitirla, como se hizo en el Listado 26. La declaración de codificación debe hacerse entonces en el encabezado con un elemento `meta` (línea 4) mucho más sencilla que en XHTML 1.0 (compárese con la línea 7 del Listado 28).

4.1.2. El elemento documento

El elemento documento o elemento raíz de un documento (X)HTML es `<html>`. Aunque no es obligatorio, se recomienda siempre especificar el idioma del documento con el atributo `lang` (Listado 26 y Listado 27). En XHTML el atributo `xmlns` es necesario para indicar que sus identificadores pertenecen al espacio de nombres de XHTML (línea 2 del Listado 26) y así evitar que colisionen con otras especificaciones. Especificar el espacio de nombres es la principal diferencia que permite al navegador distinguir si un documento web es HTML5 o XHTML5, además de la extensión del archivo y los códigos MIME. Los demás atributos del elemento `<html>` son los comunes para casi todos los elementos, llamados **atributos globales (X)HTML** {*(X)HTML global attributes*}. El estándar (X)HTML define más de 20 atributos globales. Unos pocos se listan en la Tabla 26.

Tabla 26. Atributos comunes para la mayoría de elementos (X)HTML

Atributo	Descripción
`id`	Un nombre que identifica de forma única a un elemento, de tal forma que se le puede hacer referencia posteriomente, en especial en hojas de estilo y programas de JavaScript.
`class`	Sirve para agrupar varios elementos bajo un identificador común. A todos los elementos de una misma clase se les puede aplicar estilos o acceder desde un programa de JavaScript. Por su parte, un elemento puede pertenecer a varias clases, las cuales se separan por espacios en blanco en el valor de este atributo.
`xml:lang`	Indica el idioma en el que se encuentra el contenido del elemento. Su valor es un código de idioma en el estándar ISO-639. Sólo está disponible en XHTML. Tiene prioridad sobre `lang`.
`lang`	Igual que `xml:lang`. Está disponible tanto en HTML como XHTML, por lo que se prefiere en (X)HTML5.
`dir`	La dirección del texto que se encuentra en el contenido del elemento. Puede tomar tomar los valores `ltr` para izquierda a derecha y `rtl` para derecha a izquierda.

Ejercicio 42 [ltr_rtl, 5 pts]

Obtenga una copia del archivo que hizo en el ejercicio Ejercicio 38 [html_to_xhtml, 10 pts]. Cambie la dirección del texto de derecha a izquierda del elemento `<body>` u otros elementos. Pruebe en un navegador su efecto. ¿Es lo mismo que usar el estilo de justificación derecha del texto?

En XHTML el elemento raíz `<html>` sólo admite dos elementos hijos: un encabezado (`<head>`) y el cuerpo del documento (`<body>`). En HTML estos elementos hijos son opcionales, pero se recomienda incluirlos. Si se omiten, el navegador debe distribuir los elementos al encabezado o al cuerpo de acuerdo a donde irían normalmente, lo cual requiere más procesamiento y por tanto tiempo de carga del documento.

4.1.3. Encabezado del documento web

El **encabezado del documento (X)HTML** {*(X)HTML document header*} contiene información sobre el documento mismo pero que no es considerado parte del contenido, a lo que frecuentemente se le llama *metadatos* y son útiles para los navegadores o motores de búsqueda. Los elementos que pueden aparecer en el encabezado son: el título del documento, metadatos, la base del documento, hojas de estilo, y programas de JavaScript. En esta sección se estudiarán los dos primeros.

El **título del documento (X)HTML** {*(X)HTML document title*} se escribe como un texto simple en el contenido del elemento `<title>` y es obligatorio en XHTML. No es parte del contenido del documento porque las personas lo usarán antes de tener acceso a él. Por eso, es uno de los textos más importantes y siempre debe escogerse con cuidado, tratando de que sea corto (menos de 60 caracteres (Connolly & Hoar, 2018)) y descriptivo. Aparece en la barra de título del navegador, y en los resultados de los motores de búsqueda como Google y Bing. Los lectores lo verán en el historial de su navegador y con suerte en sus favoritos.

Los **metadatos** {*metadata*} son detalles sobre el contenido del documento como el autor, palabras clave, codificación del texto, y otros que no se pueden especificar con los elementos `<title>` y `<base>` (WHATWG, 2020). Se escriben en forma de parejas atributo-valor con el elemento vacío `<meta>` como el Listado 30. El nombre del metadato se escribe en el atributo `name` y el valor del metadato en el atributo `content`. Si el nombre del atributo es el mismo que una de las directivas del encabezado HTTP (Sección 2.2.4.3) se puede escribir en el atributo `http-equiv` en lugar de `name`, como la línea 2 en el Listado 30. Cuando un documento web es solicitado, el servidor web puede revisar el encabezado por estos metadatos `http-equiv` y agregar las respectivas directivas en el encabezado del mensaje de respuesta HTTP {*HTTP Response header*} que precede al documento.

La recomendación (X)HTML no define un conjunto de valores válidos para ninguno de los atributos de `<meta>`, sino que permite libremente construir **perfiles de metadatos** {*metadata profiles*}. El Listado 30 muestra un ejemplo de encabezado con título y metadatos comunes en XHTML 1.0. Para más detalles, revísese la especificación HTML 4.01. En versiones más recientes de (X)HTML, el W3C ha adoptado una notación más elaborada para la especificación de metadatos llamada **Marco de descripción de recursos** {RDF, *Resource Description Framework*}.

Listado 30. Encabezado de un documento XHTML 1.0 con metainformación y título

```
1 <head>
2    <meta http-equiv="Content-Type" content="application/xhtml+xml; charset=utf-8"/>
3    <meta name="viewport" content="width=device-width, initial-scale=1"/>
4    <meta name="author" content="Jeisson Hidalgo-Céspedes"/>
5    <meta name="copyright" content="&copy; 2011 ECCI-UCR"/>
6    <meta name="keywords" content="desarrollo web, curso, evaluación"/>
7    <title>Carta al estudiante</title>
8 </head>
```

El metadato para declarar la codificación del documento (en la línea 2 del Listado 30) es tan común que en (X)HTML5 se agregó el atributo `charset` (que conceptualmente debió llamarse `encoding`) al elemento `<meta>`. De esta forma, los documentos (X)HTML5 pueden especificar la codificación de la forma compacta `<meta charset="utf-8"/>`, como se hizo en la línea 4 del Listado 26.

Uno de los metadatos más importantes es adaptar la región de visualización de la pantalla {*viewport*} al dispositivo, como la línea 3 del Listado 30. En general, cuando los navegadores de dispositivos móviles inteligentes cargan una página web, lo hacen para una ventana de un navegador de escritorio, y luego escalan el resultado a las dimensiones de la pantalla del dispositivo. Esta decisión se hace por el hecho de que si se usan las dimensiones del dispositivo, la mayoría de sitios web se verán mal. En consecuencia, el usuario normalmente debe desplazarse {*scroll*} hacia la información que le interesa dentro de la página y realizar un acercamiento {*zoom in*} para poder leerla.

La buena práctica del **diseño web adaptable** {*responsive web design*} establece que el dispositivo *no* debe adaptarse al sitio web, sino que el sitio web debe adaptarse al dispositivo del usuario. Para ayudar en esta práctica, Apple Inc. introdujo el perfil de metadatos `viewport` para su navegador Safari que eventualmente fue implementado en la mayoría de navegadores modernos. El ***viewport*** corresponde al área donde se dibujará la página web. El *viewport* no tiene un único valor, sino varias parejas `atributo=valor`. El atributo `width=device-width` indica al navegador que el ancho que tendrá la página web será el mismo que el ancho de la pantalla del dispositivo (línea 3 del Listado 30). El atributo `initial-scale=1` indica al navegador que no escale la página inicialmente, es decir, que no realice el *zoom out* del comportamiento por defecto.

Ejercicio 43 [website_index_1, 5 pts]

Nota: Para la serie de ejercicios identificados de la forma `website_index_\#`, no cree una carpeta, sino un archivo `index.html` en la raíz de su sitio web personal. Cada ejercicio en esta secuencia debe generar un commit con cambios en este archivo.

Cree un archivo `index.html` en la raíz de su sitio web personal. Su documento debe ser HTML5 con el espacio de nombres XHTML y apegarse a la nomenclatura XML. Escriba el encabezado HTML. Su encabezado debe tener un título significativo y al menos los siguientes metadatos: codificación, autor, el generador (quién hizo el documento: usted mismo/a) y palabras clave. Puede revisar la especificación HTML5 por documentación sobre estos metadatos. Recuerde hacer que su página se adapte al dispositivo del usuario.

4.1.4. Cuerpo del documento web

Después del encabezado (elemento `<head>`) continúa el cuerpo del documento (elemento `<body>`). El cuerpo del documento almacena el contenido del documento. Esto es lo que ve o escucha el usuario una vez que se ha cargado. El elemento `<body>` tiene cerca de 16 atributos que son manejadores de eventos y se estudiarán en la parte de comportamiento de este material. El contenido del elemento `<body>` puede ser texto puro en HTML, pero normalmente es una mezcla de los elementos que se explican en el resto de secciones de este capítulo.

Avance de proyecto 22 [prj_homepage_head, 5 pts]

Agregue la página principal en XHTML5 con nombre `index.xhtml` en la raíz de su proyecto con el contenido de un documento minimalista (Listado 26). Revise la especificación HTML5 para incluir todos los metadatos que tienen sentido para su proyecto. No es necesario agregar hojas de estilo ni programas en JavaScript por ahora. En cuanto al cuerpo del documento, puede dejar el elemento `<body>` vacío.

Asegúrese de que el documento es XHTML5 válido, y agréguelo a control de versiones. Si publica su proyecto en un servidor web, asegúrese de que despache los archivos con extensión `.xhtml` en mensajes de respuesta HTTP con el campo `Content-type: application/xhtml+xml` en sus encabezados.

Experimente introduciendo un error en el cuerpo del documento (por ejemplo, un elemento `<p>` no cerrado). Si el navegador reporta el error, está recibiendo el documento como XHTML, sino como HTML. Una vez que verifique que realmente el reporte ocurre, puede remover el error del documento.

4.2. Estructura del contenido

Un sitio web es una colección intercomunicada de páginas web. Para hacer que la interacción de los visitantes con el sitio sea clara, todas las páginas que componen el sitio deben tener la misma estructura del contenido, presentación y comportamiento. La **estructura del contenido (X)HTML** {*(X)HTML content outline*} de la página se refiere a las grandes partes o secciones de información que la componen. Ejemplos comunes de estas grandes partes en los sitios web son un encabezado del sitio, un pie de página, un menú de navegación, y el contenido propio de cada página. La Figura 26 muestra estas partes rotuladas en verde y dos distribuciones en que se pueden formatear con hojas de estilo.

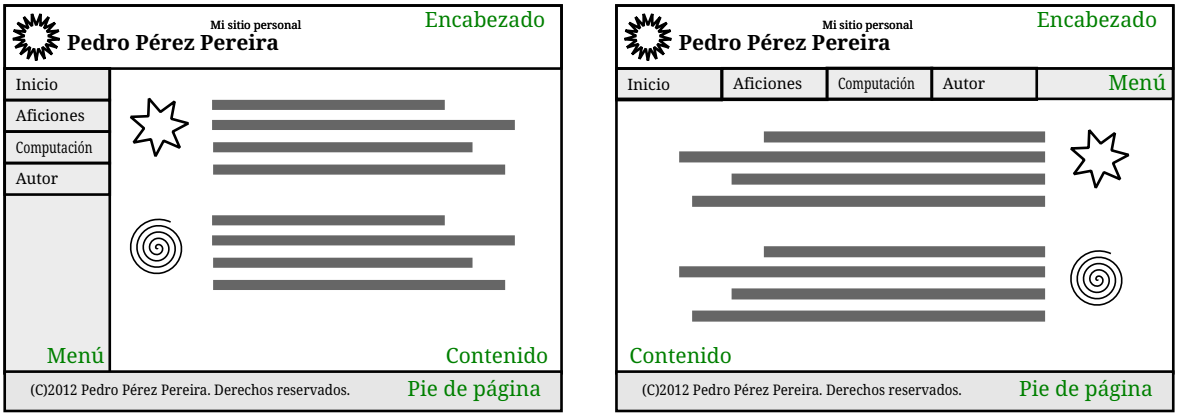


Figura 26. Una estructura del contenido y dos formas de distribuir sus secciones con estilos

Formalmente (X)HTML5 define dos formas no excluyentes para expresar la estructura del contenido: el seccionamiento de contenido y la titulación del contenido. Los autores de versiones previas de (X)HTML utilizaron elementos genéricos (`<div>`), los cuales siguen siendo válidos en (X)HTML5, pero no se recomiendan por ser carentes de semántica. Los tres tipos se explican en los apartados siguientes.

Un sitio web está compuesto de páginas web. Si se quiere que la estructura principal, la presentación y el comportamiento sea uniforme en todo el sitio web, cada página que compone el sitio debe tener la misma estructura del contenido, reglas de estilo y código de JavaScript. Esto lleva a redundancia de código. Los estándares web permiten reutilizar hojas de estilo y archivos de JavaScript a lo largo de todo el sitio, pero no proveen un mecanismo natural para reutilizar contenido (como el encabezado o pie de página del sitio web). Por esto los autores deben recurrir a otras tecnologías para evitar la redundancia de código no controlada. Ejemplos de estas tecnologías son *Server Side Includes*, programación del lado del servidor, documentos anidados (`iframe`), o programación del lado del cliente (JavaScript). En el capítulo sobre programación del lado del servidor se construirá un *web framework* minimalista para resolver este problema.

Ejercicio 44 [student_site_design, 15 pts]

Haga un esquema de página *wireframe* para su sitio web personal como la Figura 26. Dibuje en SVG o en papel a escanear, las *regiones globales* que definen el sitio y cómo quiere que se vean. Ejemplo: encabezado, pie de página, menú principal, contenido. Agregue el esquema de página a la carperta `xhtml/student_site_design/`.

Asegúrese de apegarse a una convención para nombrar carpetas y archivos, como documentos de contenido, imágenes, hojas de estilo y scripts. Escriba su convención de nombres de recursos en una sección de su `README.md` para referencia futura. Agregue las convenciones de identificadores para los diferentes lenguajes que tendrá su sitio: HTML, CSS, y JavaScript. Ejemplos de convenciones son `camelCase`, `snake_case`, o `guio-nes`.

4.2.1. Seccionamiento del contenido

(X)HTML5 define seis elementos para poder dividir o seccionar el contenido de un documento: `<header>`, `<footer>`, `<nav>`, `<aside>`, `<main>`, `<article>` y `<section>`. El Listado 31 muestra un ejemplo de cómo estos elementos pueden usarse para seccionar el contenido de una página.

Listado 31. Ejemplo de seccionamiento del contenido en XHTML5 (sectioning_content.xhtml)

```
<!DOCTYPE html>
<html lang="es" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8"/>
    <meta name="author" content="Usted"/>
    <title>Sectioning content</title>
</head>
<body>
    <header id="site_header">...</header>
    <nav id="site_menu">...</nav>
    <article id="page_content">
        ...
        <section class="content_entry">...</section>
        <section class="content_entry">...</section>
        ...
    </article>
    <footer id="site_footer">...</footer>
</body>
</html>
```

El **elemento `<header>`** sirve para describir el encabezado de página, de un artículo o una sección. En cuanto a un sitio web, el encabezado puede contener el logotipo o imagotipo de la organización o del sitio, el título del sitio, algún lema, un campo de texto para hacer búsquedas en el sitio web, un control de cuenta de usuario, entre otros elementos. En las líneas 9 a 13 del Listado 32 se muestra un encabezado de sitio web. El atributo `id` en los elementos (X)HTML es opcional, pero facilita enormemente la labor de presentación (hojas de estilo) y de comportamiento (JavaScript).

El **elemento `<footer>`** delimita un pie de página. Sirve para indicar información acerca del sitio web o de una sección, como el autor, enlaces hacia documentos relacionados (como términos de servicio), derechos de autor, un menú de navegación secundario, entre otros. Las líneas 62 a 64 del Listado 32 presentan un pie de página, cuyo texto se escribió dentro de un elemento `<small>`, redefinido en (X)HTML5 para marcar textos con restricciones legales y similares.

El **elemento `<nav>`** sirve para indicar un conjunto de enlaces que proveen navegación de la página misma (como una tabla de contenidos), o del sitio web mismo, como se hizo en las líneas 15 a 24 del Listado 32. Enlaces de publicidad, referencias bibliográficas, resultados de búsquedas, y similares, no deben encerrarse dentro de un elemento `<nav>`, dado a que no representan navegación dentro del sitio. El elemento `<nav>` puede contener títulos, listas no ordenadas, párrafos y otros elementos.

El contenido propio de la página web inicia con cualquier elemento que no sea uno de los anteriores. Es apremiante que el autor estructure el contenido utilizando elementos `<main>`, `<article>` y `<section>`.

El **elemento `<main>`** marca el contenido propio o dominante del documento, o la funcionalidad

principal de una aplicación web, como se hizo en las líneas 26 y 60 del Listado 32. Es decir, el contenido que es único en esta página y no está presente en otras páginas del sitio web. Su principal función es ayudar a las *tecnologías de apoyo* {*assistive technology*}, como lectores de pantalla, a saltar directamente al contenido principal de la página. Debe cumplir algunas restricciones. El elemento `<main>` debe ser único en el documento. Es decir, un documento no debe contener dos elementos `<main>` a menos de que su visibilidad sea excluyente con el atributo `hidden`. El elemento `<main>` debe ser hijo del elemento `<body>`, aunque con algunas restricciones podría estar dentro de un formulario web (`<form>`).

De acuerdo a la especificación oficial, el **elemento `<article>`** se utiliza para especificar una obra autocontenida en un documento, página, aplicación o sitio, que se puede distribuir o reutilizar (WHATWG, 2020). Esto implica que un artículo es un trozo de información que puede ser movido de un lugar a otro dentro de la misma página web (a un margen, por ejemplo) y mantener su sentido. Son ejemplos: una entrada de un blog o foro, una noticia, un comentario de usuario, entre otros. Los elementos `<article>` se pueden anidar. En el Listado 32 se indicó que el contenido principal de la página está compuesto de dos artículos. El primero son las secciones del sitio web (líneas 27 a 53). El segundo es una liberación de responsabilidad en la página pricipal del sitio (líneas 55 a 59).

Los artículos se dividen en secciones temáticas. Por ejemplo, un libro se divide en partes, capítulos y secciones. Cada una de estas partes se identifica con un título. El **elemento `<section>`** permite dividir un artículo en partes o secciones, las cuales no son independientes, sino que tienen un orden. Como es de esperar, si una sección de contenido se mueve de lugar dentro de la misma página web (a un margen, por ejemplo), perderá su sentido. En el Listado 32 cada entrada introductoria hacia una parte del sitio web, fue representada como una sección (líneas 34 a 40, y líneas 42 a 49). Las secciones se pueden anidar entre sí. La especificación (X)HTML5 también permite anidar artículos dentro de secciones. Las secciones también pueden tener encabezados, elementos de navegación, y pie de sección.

El **elemento `<aside>`** sirve para indicar contenido que no es considerado parte del contenido principal del documento, por ejemplo, publicidad, noticias, artículos relacionados, etc. Si se quiere, el menú de navegación del sitio (`<nav>`) puede escribirse dentro de un elemento `<aside>`, para indicar que no es parte del contenido principal.

Listado 32. Estructura en XHTML5 de una página principal de un sitio web ficticio (page_outline.xhtml)

```
1  <!DOCTYPE html>
2  <html lang="es" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4     <meta charset="utf-8"/>
5     <meta name="author" content="Jeisson Hidalgo-Céspedes"/>
6     <title>Tres uves dobles</title>
7  </head>
8  <body>
9     <header id="site_header">
10       <img id="site_logo" src="img/3w.svg" alt="Logo del sitio" width="80" height="60"/>
11       <h1 id="site_title">Tres uves dobles</h1>
12       <p id="site_tagline">Trucos y consejos de desarrollo web</p>
13    </header>
14
15    <nav id="site_menu">
```

```
      <ul>
         <li><a href="/">Inicio</a></li>
         <li><a href="/xml/">XML</a></li>
         <li><a href="/html/">(X)HTML</a></li>
         <li><a href="/css/">CSS</a></li>
         <li><a href="/js/">JavaScript</a></li>
         <li><a href="/php/">PHP</a></li>
      </ul>
   </nav>

   <main id="main_content">
      <article id="site_sections">
         <p>Bienvenido(a) a <strong>Tres uves dobles</strong>, un sitio web con consejos para crear
            sitios web. Todo el código aquí alojado puede utilizarlo libremente en sus propios
            proyectos. Además puede colaborar con nuevos trucos que considere útiles para otros
            desarrolladores. Los consejos en este sitio están separados en los siguientes grandes
            temas:</p>

         <section class="content_entry">
            <a href="/xml/"><img src="img/xml.svg" alt="Sección XML" width="80" height="60"/></a>
            <h2><a href="/xml/">Trucos sobre XML</a></h2>
            <p>Trucos para representar información en XML. Escribir y documentar sus propios DTD.
               Hacer transformaciones con XSLT. Formatear documentos con XSL-FO. Ejemplos de API y
               DOM.</p>
         </section><!-- content_entry -->

         <section class="content_entry">
            <a href="/xhtml/"><img src="img/xhtml.svg" alt="Sección (X)HTML" width="80"
               height="60"/></a>
            <h2><a href="/xhtml/">Trucos sobre (X)HTML</a></h2>
            <p>Trucos para representar contenido en HTML 4.01, XHTML 1.0 y (X)HTML5. Tipos de
               documento. Elementos de texto. Elementos de estructura. Elementos multimedia
               (imágenes, audio, vídeo).</p>
         </section><!-- content_entry -->

         <!-- ... -->

      </article><!-- site_sections -->

      <article id="disclaimer">
         <h2>Liberación de responsabilidad</h2>
         <p><small>Los ejemplos en este sitio son ofrecidos con la mejor intención de ser
            útiles y correctos. El uso y sus consecuencias no son responsabilidad...</small></p>
      </article><!-- disclaimer -->
   </main><!-- main_content -->

   <footer id="page_footer">
      <p><small>2020 Jeisson Hidalgo-Céspedes. CC BY-SA 4.0.</small></p>
   </footer>
</body>
</html>
```

Ejercicio 45 [website_index_2, 10 pts]

Identifique las partes o secciones de contenido que su sitio web personal debe tener, a partir del esquema de página que realizó en el ejercicio Ejercicio 44 [student_site_design, 15 pts]. Refleje estas partes o secciones en su índice del sitio web (`index.xhtml`) utilizando los elementos de seccionamiento definidos en (X)HTML5.

El mapa del sitio personal está dado por la estructura de carpetas sugeridas en los ejercicios de este material (`intro`, `xml`, `xhtml`, `css`, y `js`). Refleje esta estructura en el *nivel global de navegación* de su página principal (`index.xhtml`). Para cada carpeta, cree una entrada (`<section>`) en el contenido de la página principal (`index.xhtml`) y provea una imagen como se hizo en el Listado 32. Si usa gráficos permitidos de otra persona, asegúrese de dar crédito.

4.2.2. Web semántico

Los elementos genéricos `<div>` o `<span>` fueron introducidos en HTML 4.01 y XHTML 1.0 para permitir a los autores especificar la estructura de un trozo de información cuando no existe un elemento en el estándar (X)HTML para tal fin. Sin embargo, en (X)HTML5 deben evitarse por carecer de semántica.

De acuerdo a la especificación (X)HTML5 el **elemento `<div>`** carece completamente de significado (WHATWG, 2020). Simplemente se utiliza para agrupar elementos hijos o texto, al cual se le puede dar algún tratamiento con los atributos comunes (Tabla 26) como `id`, `class`, `lang` y `title`. El **atributo `id`** sirve para darle un identificador único en el documento a un elemento. El **atributo `class`** sirve para agrupar uno o más elementos con características comunes, es decir, crea una clase de elementos, de forma similar a las clases de programación orientada a objetos.

Por defecto el elemento `<div>` crea un bloque, de forma similar a los párrafos, títulos, tablas y otros elementos, que al ser colocados unos tras otros son separados visualmente por "un cambio de línea". En algunas circunstancias, el autor necesita encerrar sólo unas letras o un trozo pequeño de texto dentro de un párrafo, un título, u otro elemento de línea, con el fin de darle un tratamiento especial. Este es el objetivo del **elemento `<span>`**, el cual carece de semántica al igual que `<div>` y es normalmente usado con los atributos `class`, `id` o `lang`.

Dado que los elementos `<div>` y `<span>` carecen de significado, el WHATWG se dio a la tarea de definir elementos con semántica que los reemplazaran. La decisión fue apoyada por un estudio que Google realizó en el 2005 en más de mil millones de documentos web llamado *Web Authoring Statistics*. Entre varios análisis, el estudio reveló los valores más frecuentes en los atributos `class` de los elementos `<div>` y `<span>`. Con estos valores se definieron los elementos de seccionamiento descritos en la Sección 4.2.1.

Así como los elementos `<div>` y `<span>` no deben usarse por carecer de significado, tampoco viejos elementos de formato como `<font>`. El autor debe buscar y escoger el elemento (X)HTML5 que mejor represente *por su significado* cada trozo de información que está escribiendo y asegurarse de que sus documentos sean válidos de acuerdo a la especificación (X)HTML. Esta buena práctica se llama **web semántico** {*semantic web*} y tiene varias ventajas como las siguientes (Connolly & Hoar, 2018).

1. **Accesibilidad** {*accessibility*}. El web semántico ofrece información a los navegadores o procesadores (X)HTML para adaptar el contenido a las necesidades de los usuarios. Por ejemplo,

suponga que una persona con baja o ninguna visión accede a una de sus páginas web. Si usted usa web semántico, un software de lectura de pantalla {*screen reader software*} puede saltar el encabezado del sitio (`<header>`) e información alternativa (`<aside>`), y pasar directamente al contenido principal (`<main>`), del cual obtiene las secciones y títulos, hace un resumen auditivo al visitante, quien podrá escoger el trozo de contenido que busca. Si en cambio usted utiliza elementos `<div>` y `<span>`, el lector de pantalla no sabrá cuáles de estos representan contenido principal o secundario y por tanto el visitante tendrá que escuchar desde el encabezado, menú, texto alternativo, y contenido previo antes de encontrar lo que busca, y probablemente sin las pausas naturales que se realizan al pasar de una sección a otra.

2. **Posicionamiento en buscadores** {SEO, *Search engine optimization*}. Si usted crea web semántico, las arañas de los buscadores web podrán omitir información innecesaria y pasar directamente al contenido principal (`<main>`) y su estructura. De esta forma, cuando una persona busca por un contenido que está en su página, el buscador no sólo le proveerá la página completa como resultado, sino que guiará al visitante directamente a la sub-sección de interés. En general, ante dos sitios que son resultado de una misma consulta, los buscadores ubican primero al que es más semántico, dado que es una medida de calidad, y brinda una mejor experiencia de usuario.
3. Otras ventajas. En general las páginas web semánticas son más fáciles de editar, mantener, y requieren menos recursos que las no semánticas. Por tanto, saturan menos a los servidores, se descargan más rápido en los navegadores, y en general ayudan a una mejor experiencia de los usuarios y desarrolladores.

4.2.3. Títulos de secciones

Los documentos en (X)HTML5 son divididos en partes con elementos como `<article>` y `<section>`. Por ejemplo, un libro se divide en capítulos y secciones. Tanto la obra como sus partes son identificadas con **títulos (X)HTML** {*(X)HTML headings*}. En el caso de libro, cada uno de sus capítulos, secciones, y el mismo libro, tienen un título que los identifica y los distingue de las demás.

(X)HTML define 6 niveles de títulos en orden descendente de importancia `<h1>`, `<h2>`, ..., `<h6>`. El navegador u otro software podría usar los títulos del documento, por ejemplo, para construir una tabla de contenidos automáticamente. Es habitual que la primera actividad que realice un autor sea definir la estructura temática de su documento antes de empezar a escribir el contenido de cada sección, en otras palabras, definir los títulos del documento.

El Listado 33 presenta un ejemplo de cómo se pueden usar los títulos de secciones dentro de los elementos `<article>` y `<section>` para estructurar las partes de una tesis en (X)HTML5. En este ejemplo, el título de nivel 1 (`<h1>`) se usa para el título de la tesis, los títulos de nivel 2 (`<h2>`) para los títulos de los capítulos, los títulos de nivel 3 (`<h3>`) para los títulos de las secciones, los títulos de nivel 4 (`<h4>`) para los apartados, y así sucesivamente.

Listado 33. Ejemplo de secciones y títulos en (X)HTML5 (section_headings.xhtml)

```
<!DOCTYPE html>
<html lang="es" xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="utf-8"/>
   <title>Tesis: Representación de letras de canciones en XML</title>
</head>

<body>
<main>
<article id="tesis">
    <h1>Tesis: Representación de letras de canciones en XML</h1>
    <p>Maicol Yaxon &lt;myaxon@garach.edu&gt;</p>

    <section id="abstract">
        <h2>Resumen</h2>
        <p>Dado que no existe un estándar para representar letras[...]</p>
    </section><!-- abstract -->

    <section id="objetivos">
        <h2>Objetivos</h2>

        <section id="objetivo_general">
            <h3>Objetivo general</h3>
            <p>El objetivo general de este trabajo es formular una espec[...]</p>
        </section><!-- objetivo_general -->

        <section id="objetivos_especificos">
            <h3>Objetivos específicos</h3>
            <ol>
                <li>Analizar estadísticamente los componentes típicos [...]</li>
                <li>Proponer una representación XML de los componentes[...]</li>
                <li>Generar tipos de documentos y esquemas de la repre[...]</li>
                <li>[...]</li>
            </ol>
        </section><!-- objetivos_especificos -->

    </section><!-- objetivos -->

    <!-- ... -->

</article><!-- tesis -->
</main>
</body>
</html>
```

(X)HTML permite que los títulos estén fuera de elementos de seccionamiento de contenido. En tal caso, el navegador debe suponer que cada título está dentro de una sección implícita. Es decir, cuando se abre un título de igual o mayor nivel, el navegador supone que la sección anterior se debe cerrar. El

uso de títulos fuera de artículos o secciones, limita el poder de los estilos (CSS) o del comportamiento (JavaScript), y no se recomienda para documentos extensos o complejos.

Los estándares web establecen que el formato *por defecto* de cada elemento visible en (X)HTML debe ser escogido por el navegador, con el fin de que pueda ser adaptado a cada medio específico. Los navegadores emplean tamaños más grandes y mayor peso en las tipografías para que los títulos sean más vistosos. Esto a veces provoca un efecto indebido, ya que algunos autores escogen el nivel del título por la apariencia. Por ejemplo, usar un `<h3>` para el título del sitio dado que su tamaño por defecto es el que mejor se aproxima al alto del logotipo. La práctica recomendada del *web semántico* dicta que los elementos deben ser escogidos por su significado y no por su apariencia. El encabezado `<h1>` se recomienda para el título del sitio web, `<h2>` para el título de cada página web, `<h3>` para los artículos o secciones de la página, y así sucesivamente, sin importar su apariencia. (X)HTML permite al autor controlar el formato utilizando hojas de estilo como se estudiará en un capítulo posterior.

Ejercicio 46 [website_index_3, 5 pts]

En la página principal de su sitio web personal (`index.xhtml`), escriba el título de su sitio web en un título de nivel 1 (`<h1>`). No utilice elementos `<h1>` para ningún otro título del documento. Su título del sitio debe estar en encabezado del sitio, y por tanto, dentro de un elemento `<header>`.

Utilice un título de nivel 2 para el título de la página web dentro del contenido principal (`<main>`). Es probable que el texto de este elemento sea una réplica del elemento `<title>`. No utilice títulos de nivel 2 para otros títulos del documento.

Utilice títulos de nivel 3 (`<h3>`) para las secciones de contenido. Debería tener varios títulos de este nivel en su página web. Recuerde siempre validar sus documentos web tras modificarlos.

Avance de proyecto 23 [prj_homepage_outline, 5 pts]

Seccione el cuerpo (`<body>`) de su `index.xhtml` para reflejar las secciones del esquema de la página principal de su proyecto. Este esquema {*wireframe*} se encuentra documentado en el archivo `design/design.md` el cual realizó en el Avance de proyecto 10 [prj_wireframe_doc, 15 pts].

Asegúrese de utilizar los elementos XHTML5 más significativos para cada sección del esquema de página. Utilice títulos del nivel apropiado. Cada sección única debe tener un identificador (atributo `id`), y cada sección repetible debe tener una clase (atributo `class`). Asegúrese de que su página principal resultante sea válida de acuerdo al estándar XHTML5.

4.3. Elementos de texto

En esta sección se presentarán los elementos más comunes para estructurar el texto del documento web: párrafos, texto preformateado, elementos de frase, y listas de elementos.

4.3.1. Párrafos

Un párrafo en (X)HTML se escribe con el elemento `<p>` como se aprecia en el listado Listado 34.

Aunque en HTML es opcional cerrarlo, es recomendable siempre hacerlo. A través de hojas de estilo se puede controlar el espaciado entre párrafos, la sangría y otros formatos. No escriba párrafos vacíos `<p></p>` para tratar de dejar espacio en blanco en el documento. La especificación (X)HTML recomienda al navegador ignorar estos elementos. Algunos autores, entonces, fuerzan el espaciado con párrafos que consisten de espacio no separable (`<p> </p>`), lo cual es el mal uso de la estructura (los párrafos) con fines de apariencia (el estilo).

Listado 34. Párrafos y texto preformateado en (X)HTML (paragraphs.xhtml)

```
<!DOCTYPE html>
<html lang="es" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8"/>
    <title>Programación en C++</title>
</head>
<body>
<main>
<article id="libro">
    <header>
        <p>Universidad de Costa Rica<br/>
        Escuela de Ciencias de la Computación e Informática<br/>
        CI1201 - Programación II</p>
    </header>
    <h1>Programación en C++</h1>

    <section id="introduccion">
        <h2>Introducción a la programación</h2>
        <p>La computadora es un artefacto electro-matemático que puede ejecutar una secuen[...]</p>

        <section id="hola_mundo">
            <h3>Hola mundo en C++</h3>

            <p>Desde el libro de Kernighan y Ritchie se ha seguido la tradición del primer programa
                a mostrar sea uno simple: escribir la cadena "Hola mundo" en la terminal. La forma
                de decir "Hola mundo en C++" se aprecia en el siguiente código:</p>

            <pre><![CDATA[#include <iostream>

int main()
{
    std::cout << "Hola mundo" << std::endl;
    return 0;
}
]]></pre>

        </section><!-- hola_mundo -->
    </section><!-- introduccion -->
</article><!-- libro -->
</main>
</body>
</html>
```

A veces se quiere insertar un cambio de línea sin iniciar otro párrafo, por ejemplo, en un poema los versos se separan por cambios de línea y las estrofas por párrafos. Un cambio de línea se representa con el elemento vacío `<br/>` (acrónimo de *break return*). Una vieja práctica agregaba un espacio en blanco antes de la barra inclinada en las etiquetas vacías de la forma `<br />`, para que navegadores que no implementan XHTML ignoraran la barra inclinada como si fuese un "dedazo" del autor. Esta práctica ya no es necesaria.

Cuando la longitud de un párrafo excede el ancho de la ventana, el navegador inserta cambios de línea automáticos {*word wrap*}. Cuando se quiere evitar que el navegador por esta característica separe dos palabras, estas deben separarse por un **((espacio no divisible)** {*not breaking space*} obtenible por la referencia de entidad ` `. Por ejemplo, el código `(506) 905-LLAMEYA` genera el texto "(506) 905-LLAMEYA". Si usted redimensiona la ventana del navegador, notará que el número de teléfono no será separado del código de país mientras se ajusta el párrafo automáticamente al tamaño de la ventana (probar en una ventana externa). La entidad ` ` está sólo disponible en HTML. En XHTML se puede usar su equivalente ` `.

El hecho de que (X)HTML ignore los cambios de línea que usted escribe, no es práctico para escribir código fuente u otros tipos de escritos donde el espaciado en blanco es relevante. Para esas situaciones (X)HTML provee el elemento de **texto preformateado** {*preformatted text*} `<pre>`, el cual instruye al navegador mantener los espacios en blanco, usar una fuente mono-espaciada, y desactivar el ajuste automático de línea {*word wrap*}. La especificación recomienda a los navegadores reemplazar tabuladores por 8 espacios en blanco, lo cual muchas veces resulta molesto, por eso, es recomendable utilizar espacios en blanco en lugar de tabuladores en las secciones `<pre>` o hacer modificaciones con programación del lado del servidor o JavaScript.

El navegador intentará interpretar el marcado (X)HTML que escriba dentro de un elemento `<pre>`, por esto debe al menos reemplazar los tres caracteres especiales (`<`, `>` y `&`) con sus referencias de entidad respectivas (`&lt;`, `&gt;` y `&amp;`), o escribir el texto dentro de una sección CDATA como se hizo en las líneas 28 a 35 del Listado 34, lo cual tendrá efecto sólo si su documento es XHTML.

Un **elemento `<hr>`** {*hard return*} crea una línea horizontal que sirve para separar párrafos tratando de hacer una separación más visible entre ellos, por ejemplo, para separar los cambios temáticos entre estrofas de un poema. Es un elemento vacío y su apariencia debe ser estrictamente controlada con estilos.

Ejercicio 47 [html_br_close, 5 pts]

¿Cuál de las siguientes afirmaciones es *falsa*?

1. El elemento `br` siempre debe cerrarse en XHTML de la forma `<br/>` ó `<br></br>`.
2. El elemento `br` nunca debe cerrarse en HTML 4.01.
3. Para permitir que navegadores viejos puedan mostrar código XHTML, algunos autores cierran el elemento br antecediendo un espacio en blanco, de la forma `<br />`. Esto es válido en XHTML pero no en HTML, sin embargo, los navegadores ignoran lo que no entienden y esto es lo que hacen los navegadores viejos cuando ven esta barra inclinada.
4. El estándar HTML 4.01 se modificó para que `<br />` sea una construcción completamente válida.
5. El estándar HTML5 se modificó para que `<br />` sea una construcción completamente válida.
6. Lo anterior también se cumple para los elementos `hr`, `img` y `meta`.

Ejercicio 48 [html_inconsistent_output, 5 pts]

En su nuevo trabajo usted está mejorando un sistema existente que genera varios documentos HTML automáticamente. Uno de ellos es bastante extenso, se comporta diferente entre navegadores, a veces carga por completo y otras sólo un trozo del documento aparece, incluso en el mismo navegador. La información en el documento es importante y el cliente está bastante disconforme. Hojeando rápidamente el código generado usted detecta los siguientes problemas. ¿Cuál de ellos puede estar ocasionando este comportamiento impredecible?

1. Dos elementos `<pre>` están anidados.
2. Algunas etiquetas `<p>` están cerradas y otras no.
3. La indentación es inconsistente.
4. No especifica el tipo de documento, ni el encabezado HTML.
5. Hay caracteres `- -` dentro de comentarios

Ejercicio 49 [blockquote_q, 5 pts]

Los elementos `<blockquote>` y `<q>` sirven para citar ideas de otro autor. Investigue en la especificación:

1. ¿Cuál es la diferencia entre `<blockquote>` y `<q>`?
2. ¿Debe el autor proveer comillas o estos elementos las agregan automáticamente?
3. ¿Estos dos elementos permiten agregar varios párrafos `<p>` en su contenido?
4. ¿En qué afecta indicar el idioma de la cita?
5. ¿Cuáles de estos elementos se pueden anidar?
6. ¿Qué abuso han históricamente cometido los autores de páginas web con el elemento `<blockquote>`

Ejercicio 50 [website_index_4, 5 pts]

Su `index.html` debe hacer referencia a las grandes secciones de contenido de su sitio web (intro, xml, html, css, js, php). Si no lo ha hecho, escriba una sección "entrada de contenido", y por tanto algo como `<section class="content_entry">` para cada una de las secciones de su sitio web. Cada sección de contenido debe tener al menos un título, una imagen y un párrafo. El párrafo debe explicar al visitante de qué se trata esa sección.

4.3.2. Elementos de frase

(X)HTML define los **elementos de frase** {*phrasing elements*} para agregar información a fragmentos de texto. Son elementos que afectan no a un párrafo sino sólo a algunas palabras de un párrafo, por ejemplo, denotan énfasis en una palabra, un trozo de código, una abreviatura, etc. La Tabla 27 lista los elementos de frase definidos en (X)HTML.

Tabla 27. Elementos de frase en (X)HTML

Elemento	Descripción
`<abbr>`	Indica una abreviatura, sigla o palabra reducida, como WWW, HTTP, etc., y PhD.
`<acronym>`	Indica un acrónimo, que es una abreviatura que se lee como si fuera una palabra normal, por ejemplo: sonar, codec, y JSON.
`<cite>`	Indica una cita o referencia a otras fuentes, en especial para resaltar el título de la obra.
`<code>`	Indica un fragmento de código informático.
`<dfn>`	Indica un término que se está definiendo.
`<em>`	Indica énfasis.
`<kbd>`	Indica que el texto debe ser ingresado o tecleado por el usuario.
`<samp>`	Indica salida de programas informáticos.
`<strong>`	Indica mayor énfasis.
`<var>`	Indica una variable de un programa informático.

Los dos elementos de frase de uso más común son `<em>` y `<strong>`. Normalmente los navegadores los despliegan en itálicas y negritas respectivamente, pero se puede alterar con hojas de estilo. Los demás elementos de frase están orientados a documentos técnicos. El listado Listado 35 muestra el uso de algunos de estos elementos.

Listado 35. Elementos de frase en (X)HTML (phrase_elements.xhtml)

```
<!DOCTYPE html>
<html lang="es" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8"/>
    <title>Phrase elements</title>
</head>

<body>
    <p>
        Se dice que <em>las entidades</em> describen la <strong>estructura física</strong>
        y <em>los elementos</em> la <strong>estructura lógica</strong> de los documentos
        <abbr title="Lenguaje de marcado extensible (eXtensible Markup Language)">XML</abbr>.
        La estructura lógica (elementos) y física (entidades) se representa dentro del
        documento <abbr title="Lenguaje de marcado extensible (eXtensible Markup Language)">
        XML</abbr> agregando <strong>el marcado</strong>, el cual
        se delimita de <em>los datos de carácter</em> encerrando la descripción de los
        elementos dentro de paréntesis angulares ("<code>&lt;</code>" y "<code>&gt;</code>"),
        que se conocen como <dfn>etiquetas</dfn> y <em>las referencias a entidades</em> entre
        el signo "<code>&amp;</code>" y el punto y coma ("<code>;</code>"). Es decir, el
        texto encerrado dentro de estos cuatro caracteres se conoce como <dfn>marcado</dfn>,
        lo restante como <dfn>datos de carácter</dfn>. La combinación del <em>marcado</em>
        más los <em>datos de carácter</em> son el <dfn>texto XML</dfn> [Gold99].
    </p>
</body>
</html>
```

Al igual que todos los elementos (X)HTML, los de frase tienen un atributo `title` en el cual se puede escribir información adicional. Esta información aparece cuando hay cierta interacción con el elemento, normalmente un *tooltip* cuando el puntero del ratón pasa sobre ellos. Esto es útil para expandir el significado de las abreviaturas y acrónimos sin tener que incluir estas definiciones explícitamente en el texto cada vez que se quiere usar la abreviatura.

Nótese que el atributo `title` se definió dos veces en el listado Listado 35 para la abreviatura "XML" (líneas 4 y 6), lo que genera redundancia de código. Teóricamente esta redundancia se puede evitar en XHTML definiendo entidades generales en el parámetro interno del tipo de documento, sin embargo, los navegadores las ignoran. Por tanto, se requiere hacer algún tipo de programación para evitar la redundancia. Otra desventaja del atributo `title` es que puede no activarse en ciertos dispositivos, en especial aquellos que no tienen un dispositivo de apuntar.

Los elementos `<sub>` y `<sup>` sirven para declarar subíndices y superíndices respectivamente. Aunque tienen utilidad matemática y en otras notaciones científicas, realmente se mantienen en la recomendación para otros textos, como los ejemplos de la Tabla 28.

Tabla 28. Subíndices y superíndices en (X)HTML

Texto	Código (X)HTML
El 1ro de la clase	`El 1<sup>ro</sup> de la clase`

Texto	Código (X)HTML
May 3rd, 2011	`May 3<sup>rd</sup>, 2011`
$C_6H_{12}O_6$	`C<sub>6</sub>H<sub>12</sub>O<sub>6</sub>`

El **elemento `<time>`** aparecido en (X)HTML5 permite especificar que un trozo de texto representa una fecha, una hora o ambas. Posibilita a los programas el procesamiento de fechas y horas, por ejemplo, para agregar un evento a la agenda del usuario, hacer conversiones de huso horario, guiar a los motores de búsqueda a indexar la fecha correcta, entre otras. El contenido del elemento `<time>` permite escribir la fecha en el formato que prefiera el autor, y el atributo `datetime` en el formato `YYYY-MM-DDThh:mm:ss.sss±HH` entendido por las computadoras. El Listado 36 muestra algunos ejemplos.

Listado 36. Ejemplos de uso del elemento `<time>`

```
Estimado cliente. Su tiquete para el vuelo LA389 del
<time datetime="2012-04-30T18:45">lunes 30 de abril a las 6:45 p.m.</time>
ha sido reservado con éxito. La hora actual de su destino, San Pedro Sula, es
<time datetime="20:29">8:29 p.m.</time> Recuerde registrar este evento en su agenda.

<small>©2012 Aivan Airlines Group. Documento generado el
<time datetime="2012-03-23T00:26:54">23-mar-2012 12:26:54 a.m.</time></small>
```

Ejercicio 51 [xhtml_del_ins, 5 pts]

Investigue los elementos `<del>` e `<ins>` en la especificación (X)HTML5. Escriba un documento de ejemplo, en el cual los usa y describe su utilidad.

Ejercicio 52 [xhtml_i_b_s, 5 pts]

Investigue los elementos `<i>`, `<b>`, `<s>`, `<small>` y `<mark>` en la especificación (X)HTML5.

1. ¿Cuál es la diferencia entre `<em>` e `<i>`?
2. ¿Cuál es la diferencia entre `<strong>` y `<b>`?
3. ¿Cuál es la diferencia entre `<del>` y `<s>`?

Si un documento (X)HTML sólo debe representar contenido y no presentación ni comportamiento ¿por qué el estándar (X)HTML5 los mantiene?

4.3.3. Listas

(X)HTML permite definir listas de ítemes. Las hay en tres tipos: listas ordenadas, listas no ordenadas, y listas de definiciones. Las tres se pueden anidar. El Listado 37 muestra un ejemplo de los tres tipos de listas y su anidamiento:

Listado 37. Listas de elementos en (X)HTML (lists.xhtml)

```
<!DOCTYPE html>
<html lang="es" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8"/>
    <title>Listas de elementos</title>
</head>

<body>
    <h1>Puesto: Desarrollador web</h1>
    <h2>Requisitos:</h2>
    <ul>
        <li>Conocimiento de estándares web</li>
        <li>Programación en PHP o JSP</li>
        <li>DBMS libres: PosgreSQL, MySQL o SQLite</li>
        <li>Afiliado al colegio respectivo</li>
    </ul>

    <h2>Procedimiento de reclutación</h2>
    <ol>
        <li>Llenar la fórmula RE-TI-C-2348 a mano</li>
        <li>Presentarla en las oficinas centrales</li>
        <li>Si es seleccionado:
            <ol>
                <li>Presentarse a realizar el examen RE-TI-E-23</li>
                <li>Presentarse a realizar el examen RE-LG-E-02</li>
                <li>Si es seleccionado: presentarse a trabajar</li>
            </ol>
        </li>
    </ol>

    <h2>Definiciones</h2>
    <dl>
        <dt>DBMS</dt><dd>Database Management System</dd>
        <dt>RE-TI-C-2348</dt><dd>Fórmula de cualidades para TI</dd>
        <dt>RE-TI-E-23</dt><dd>Examen de tecnologías web</dd>
        <dt>RE-LG-E-02</dt><dd>Examen básico de inglés</dd>
    </dl>
</body>
</html>
```

Las **listas ordenadas (X)HTML** {*(X)HTML ordered lists*} se escriben con el elemento `<ol>`. Sirven para indicar que los ítemes en la lista siguen cierto orden, como en una serie de pasos a ejecutar o cuando se quiere simplemente enumerar ítemes. Cada ítem de la lista se escribe dentro del **elemento `<li>`** {*list item*}. Los navegadores les anteponen números arábigos por defecto, pero con estilos estos se pueden cambiar por números romanos, letras, u otras formas. El atributo `start` del elemento `<ol>` en (X)HTML5 permite indicar el número con el cual se inicia el conteo, y si no se incluye se asume que es 1.

Las **listas no ordenadas (X)HTML** {*(X)HTML unordered list*} se escriben con el elemento `<ul>`. Sirven para indicar una lista de ítemes que no siguen un orden inherente. Visualmente se anteceden con una viñeta circular que puede cambiarse con estilos a rectángulos u otras figuras. En los elementos `<nav>` se suelen incluir listas no ordenadas de enlaces para definir la navegación del sitio web.

Las **listas de definiciones (X)HTML** {*(X)HTML definition list*} se escriben con el elemento `<dl>`. Sirven para escribir glosarios u otros tipos de estructuras similares. A diferencia de los tipos de listas anteriores, los ítemes de las listas de definiciones constan de parejas término-definición. El término se escribe con el **elemento** `<dt>` {*definition term*} y su definición con el **término** `<dd>` {*definition description*}.

4.4. Enlaces

Un enlace es un mecanismo que permite acceder a otros recursos desde un documento web. (X)HTML define dos tipos: los enlaces a recursos externos y los hiperenlaces. Los enlaces con el elemento `<link>` provoca que el recurso externo sea importado en el documento web, es decir, forma parte integral de la página web, y ésta se cargará de forma incompleta si el recurso externo no se puede acceder. Los hiperenlaces con el elemento `<a>` provocan un cambio de navegación, es decir, el navegador irá al recurso destino.

4.4.1. Hiperenlaces

Cuando el usuario accede a un **hiperenlace** {*hyperlink*}, el agente de usuario navega al nuevo recurso, lo que provoca un cambio de contexto. El recurso destino puede ser otra sección del mismo documento, otro documento web, un correo electrónico, un archivo (sonido, imagen, video, ejecutables, comprimidos), y en general cualquier recurso que pueda ser identificado por un URI. Los enlaces se escriben con el elemento `<a>` y pueden tener varios atributos (WHATWG, 2020). El Listado 38 muestra su sintaxis más básica.

Listado 38. Sintaxis de un enlace

```
<a href="uri-destino">contenido del hiperenlace</a>
```

El valor del atributo `href` (contracción de *hypertext reference*) es un URI hacia el recurso que se quiere enlazar. El contenido del hiperenlace típicamente es un texto o una imagen. El navegador permitirá al usuario accionar el contenido del enlace, por ejemplo, haciendo clic sobre él. Cuando esto ocurre, el navegador trata de mostrar el recurso apuntado por el atributo `href` en la misma ventana donde está el documento con el enlace, y por tanto, reemplaza a dicho documento. Si el navegador no sabe cómo presentar el recurso, tratará de invocar un programa que sí lo haga o permitirá al usuario guardar el recurso en su sistema de archivos.

El autor puede incidir en el comportamiento por defecto con atributos. El atributo `download="yes"` sugiere al navegador siempre descargar el recurso en lugar de tratar de reemplazar el documento actual por el recurso destino. El atributo `ping="URI"` es útil para informar a un programa identificado por `URI` que el enlace fue accedido. El atributo `type="MIME"` permite informar al navegador el tipo de recurso destino.

Los elementos `<a>` nunca deben anidarse. A modo de sugerencia trate de usar como contenido del enlace texto que es parte de la narrativa original del documento, y no un "click aquí".

Ejercicio 53 [exc_navigation_links, 5 pts]

Actualizar el menú de navegación en la página principal de su sitio web personal. El menú debe constar de una entrada para la página principal (`Inicio`) y una entrada `<li>` para cada subcarpeta temática del curso (`intro`, `xml`, `xhtml`, `css`, `js`). Cada entrada del menú debe ser un enlace que lleve a la subcarpeta correspondiente, incluso aunque esta no exista aún.

Sugerencia. Dado que su sitio web personal está alojado en un servidor web, puede anteponer la raíz `/` al URI de cada entrada del menú. Por ejemplo `<a href="/intro">Introducción</a>`. El uso de la raíz de su sitio web le permitirá reutilizar el menú de navegación en páginas secundarias de su sitio web sin tener que modificarlo. Para la página principal, el URI basta con la raíz misma, por ejemplo `<a href="/">Inicio</a>`.

Pruebe que el menú de navegación le permita recorrer las subcarpetas de su sitio web. Recuerde verificar que su documento XHTML5 sea válido.

Avance de proyecto 24 [prj_navigation_links, 5 pts]

Actualice el menú de navegación en la página principal de su aplicación web, de la misma forma que se sugiere en el ejercicio `exc_navigation_links`.

4.4.2. Anclas

El elemento `<a>` también permite darle nombre a un punto específico del documento, al cual se le puede hacer referencia posteriormente con un URI. A este punto se le llama **ancla** (en inglés, *anchor*, de ahí el nombre del elemento `<a>`). La siguiente notación crea un ancla en el documento

```
<a name="nombre_ancla"/>
```

Para hacer referencia a un ancla, se crea un enlace y se le indica al navegador que su destino es el nombre del ancla pero antecedido por un símbolo de número (`#`). De esta forma un enlace puede apuntar hacia un documento web, y con el símbolo de número hacia una sección específica de dicho documento, como se aprecia en el Listado 39.

Listado 39. Ejemplo de creación y acceso a anclas en un documento

```
<!-- Define un ancla en este documento -->
<a name="Introduccion"/><h1>Introducción</h1>

<!-- Apunta hacia un ancla en este mismo documento -->
<a href="#Introduccion">Introducción</a>

<!-- Apunta hacia un ancla en otro documento -->
<a href="glosario.html#dtd">DTD</a>
```

La practicidad de las anclas ha caído desde que el W3C permitió que cualquier elemento pueda ser identificado de forma única en el documento con el atributo `id`. El elemento puede ser enlazado utilizando la misma notación de las anclas (`#`), como se ve en el .

Listado 40. Ejemplo de anclas con el atributo `id`

```
<h2>Tabla de contenidos</h2>
<p>
    <a href="#introduccion">Introducción</a><br />
    <a href="#arq_web">Arquitectura web</a><br />
    <!-- ... -->
</p>

<article id="introduccion">
    <h2>Introducción</h2>
    <!-- ... -->
</article>

<h2 id="arq_web">Arquitectura web</h2>
```

4.4.3. Ventana objetivo

Cuando el usuario hace click en un enlace, el navegador reemplaza el documento con el recurso referido. A veces este comportamiento no es el deseado. El atributo `target` del elemento `<a>` permite especificar el nombre de una ventana, pestaña, o marco {*frame*}. Si no existe ninguna con el nombre especificado, el navegador creará una y mostrará en ella todos los recursos destinados a ese nombre. Si se usa `"_blank"` como valor del atributo `target`, el recurso será siempre cargado en una ventana o pestaña nueva del navegador.

```
<a href="destino.url" target="ventana_objetivo">contenido del enlace</a>
```

El atributo `target` está disponible en todas las variantes de (X)HTML, excepto *XHTML 1.x strict*. Esta decisión se debió a que especifica comportamiento, lo cual es responsabilidad de JavaScript. En XHTML5 el atributo puede usarse con confianza.

4.4.4. Acceso con el teclado

Los navegadores permiten acceder a los enlaces, y en general a cualquier elemento, con el teclado, lo cual es importante para aquellos en modo texto o para personas con necesidades especiales. Esto además habilita otras opciones que el desarrollador web debe tener presente.

Se puede asignar atajos de teclado {*keyboard shortcuts*} a los enlaces más frecuentes de su sitio web con el atributo `accesskey`, el cual recibe sólo un carácter, por ejemplo, `accesskey="K"`. El navegador habilitará alguna combinación de teclas, como <kbd>Ctrl+K</kbd>, <kbd>Alt+K</kbd>, o <kbd>Cmd+K</kbd>, que permitirá activar el enlace o al menos seleccionarlo. Es normal que el autor provea en el texto del documento una indicación de que el enlace tiene un atajo de teclado, ejemplo:

```
<a href="#xhtml_keyboard_shortcuts" accesskey="k">Acceso con el teclado (Ctrl+K)</a>
```

Desdichadamente el comportamiento de los atajos del teclado son dependientes del navegador. En algunos agentes de usuario los atajos pueden no funcionar o reemplazar otras operaciones preestablecidas del navegador.

La mayoría de navegadores permiten recorrer los enlaces, controles de formularios y otros elementos con la tecla tabulador. El autor puede alterar este orden con el atributo `tabindex="n"`, donde `n` es la posición en el orden que se quiere para ese enlace o elemento. Si se le da un número negativo se sacará de la secuencia. Si dos elementos tienen el mismo valor para este atributo, el navegador seguirá el orden de aparición en el documento.

4.5. Imágenes

Una característica notable del hipertexto es el soporte de varios medios como imágenes, sonidos, y vídeos. Aunque se pueden escribir de forma codificada dentro del documento, las imágenes usualmente se almacenan en recursos externos (archivos) y se enlazan desde el documento web. Para incluir una imagen se utiliza el elemento vacío `<img>`, cuyo atributo `src` debe tener el URI que identifica el archivo con la imagen, y puede ser absoluto o relativo al documento. La sintaxis más básica del elemento img es como sigue.

```
<img src="uri-imagen" alt="descripción de la imagen"/>
```

Por motivos de accesibilidad, en especial para personas ciegas, toda imagen en (X)HTML debe tener un texto alternativo en el atributo `alt`. Un software de lectura de pantalla podrá convertir a voz este texto, pero no el contenido de la imagen. El texto alternativo también se usa en otras circunstancias, como cuando la imagen no es cargada por un error de conexión, porque la imagen dejó de existir en el servidor, o porque el usuario ha deshabilitado las imágenes en su navegador.

Algunos navegadores muestran el texto alternativo como una descripción emergente {*tooltip*} cuando el puntero del ratón pasa sobre ellas. Sin embargo, esta función es realmente responsabilidad del atributo global `title` (Sección 4.3.2). Si la imagen se va a usar como una decoración y no se quieren descripciones emergente, puede asignar valores vacíos de la forma `<img alt="" title="" ⋯ />` y asegurarse de que la imagen no sea parte del contenido principal de la página (elemento `<main>`). La

apariencia del texto alternativo se puede ajustar con hojas de estilo.

4.5.1. Dimensiones de imagen

Cuando el navegador está cargando un documento web y encuentra un elemento `<img src="uri-imagen" alt="algún texto"/>` con esos atributos, solicita al servidor web que le envíe una copia de la imagen. Mientras ésta llega, el navegador sigue cargando (*rendering*) el resto del documento (X)HTML. Es probable que termine de mostrar el documento antes de que la imagen llegue, y antes que el visitante comience a leer el documento. Cuando finalmente la imagen es recibida, el navegador la insertará en el lugar donde encontró el elemento `<img>` correspondiente, desplazando el texto que se encuentre debajo. Esto es molesto para el lector, en especial si el documento tiene un considerable número de imágenes, o son de gran tamaño (en bytes), o ambas.

El problema anterior se soluciona especificando las dimensiones de la imagen en el documento (X)HTML, de tal forma que el navegador pueda reservar espacio para la imagen antes de que esta sea recibida. Simplemente especifique las dimensiones en pixeles con los atributos `width` y `hight` como el ejemplo en el Listado 41.

Listado 41. Ejemplo de dimensiones de imagen en (X)HTML

```
<img src="uri-imagen" alt="texto alternativo" width="320" height="280"/>
```

Si el valor indicado en los atributos ancho o alto de `<img>` no coincide con el tamaño real de la imagen, el navegador la escalará. También se puede especificar en ellos un porcentaje de la ventana, algo como `width="75%"`. El ancho y alto de una o varias imágenes y otros detalles se pueden controlar con estilos, lo cual es muy útil para un conjunto grande de imágenes que se quieren comparta las mismas dimensiones.

4.5.2. Formatos de imágenes

Aunque existen muchos formatos de imágenes, sólo unos pocos se pueden utilizar en la web. La Tabla 29 muestra una comparación de los formatos más soportados entre los navegadores actuales. Es tentador publicar imágenes de alta calidad, pero su tamaño será considerable y tardarán en cargarse en proporción inversa al ancho de banda del visitante. Si una o muchas imágenes hacen lento el cargado de una página web, es una invitación al lector para abandonarla. El autor debe hacer un balance entre la calidad de la imagen y su tamaño en bytes. Por esto es importante que conozca la diferencia entre los formatos disponibles y cómo ajustar este balance.

Tabla 29. Formatos de imagen soportados por navegadores

Formato	Tipo	Colores	Transparencia	Pérdida	Utilidad
PNG	Escalar	256 / 16M	Sí	No	Ilustraciones
JPEG	Escalar	16M	No	Sí	Fotografías
GIF	Escalar	256	Sí	No	Animaciones sencillas
SVG	Vectorial	16M	Sí	No	Ilustraciones

Cuando un autor quiere representar alguna pieza de información en forma gráfica, debe decidir si utilizar una fotografía, una ilustración, o una animación. Esta decisión ayuda en gran medida a delimitar el formato a escoger.

Una **fotografía** tiene millones de colores o grises. Se obtienen por cámaras digitales, escáneres, satélites, o fuentes afines. Normalmente las fotografías no tienen información de transparencia. El formato más recomendable para fotografías es el JPEG, creado en 1992 por el *Joint Photographic Experts Group*. Es un algoritmo de pérdida de calidad que trata de descartar detalles que el ojo humano no percibe con el fin de ahorrar espacio. El formato PNG (*Portable Network Graphics*) también puede almacenar fotografías, pero sin pérdida de calidad. Es recomendable sólo si el visitante web espera esta calidad y está anuente al impacto en la velocidad de descarga de los recursos.

Una **ilustración** es una imagen normalmente elaborada por computadora a partir de figuras geométricas básicas y textos. Son ejemplos de ilustraciones los diagramas ideados a complementar algún texto. La cantidad de colores de una ilustración suele ser reducida, en contraposición a las fotografías obtenidas cámaras o escáneres. El formato PNG o SVG (*Scalar Vector Graphics*) son aptos para ilustraciones web. La diferencia entre ambos radica en si la imagen es escalar o vectorial respectivamente.

- Una **imagen escalar** es un mapa de bits, es decir, una matriz de tamaño definido donde cada celda o pixel almacena un color codificado en forma numérica. Dimensionar a menor tamaño una imagen escalar provoca que ésta se deforme en especial si no se mantiene su proporción. Si se agranda una imagen escalar provocará que los pixeles se propaguen a las celdas adyacentes, generando zonas de colores poco agradables a la vista humana.
- Una **imagen vectorial** está compuesta por figuras geométricas como puntos, líneas, curvas o polígonos. Sus atributos como posición y tamaño son los que se almacenan en el archivo. Dimensionar una imagen vectorial a cualquier tamaño significa reajustar las posiciones de las figuras geométricas y volverlas a pintar, lo que provoca que la imagen siempre se vea agradable a la vista humana. En la web, las imágenes vectoriales se escriben en formato SVG, que es un estándar abrigado por el Consorcio Web.

Si el autor encuentra que una *animación* es el mejor medio de comunicar algo, se pueden crear animaciones escalares o vectoriales complejas con aceleración gráfica a través de JavaScript. Existen bibliotecas y herramientas que facilitan este proceso. Otra alternativa es utilizar videos, que además pueden tener sonido. Un viejo formato que fue popular en los inicios de la web fueron las imágenes GIF (*Graphics Interchange Format*), por permitir animaciones escalares sencillas, limitadas a un máximo de 256 colores. Sin embargo, en la actualidad son poco llamativas por sus limitaciones y su licencia de uso.

Indiferentemente de si se utilizan imágenes escalares o vectoriales, es apremiante reducir el tamaño tanto como se pueda sin que la imagen se vea desagradable a la vista humana. El autor necesita que el editor de imágenes lo apoye en esta labor. Normalmente estos programas permiten cambiar parámetros y ver su efecto en tiempo real sobre la calidad y tamaño de la imagen destino. Es importante guardar los medios originales (imágenes, sonidos, videos) en un formato de mayor calidad o sin pérdida, aunque no en el repositorio de control de versiones.

Ejercicio 54 [exc_homepage_images, 10 pts]

Elabore las imágenes de su página principal. Necesitará una imagen para el encabezado de su sitio web (logotipo o imagotipo). Cada sección (`intro`, `xml`, `xhtml`, ...) debe tener su propia imagen. Las imágenes puede obtenerlas mediante fotografía o ilustración. Si obtiene imágenes que no son de su autoría, asegúrese de que cumple los derechos de uso y sus requisitos, como dar crédito al artista. Recuerde que su sitio web será publicado y no debe exponerse a una denuncia legal.

Asegúrese de que todas las imágenes tienen las mismas dimensiones, y que estas coincidan con las dimensiones que necesita en su página principal. Edite las imágenes para reducir su tamaño sin perder la calidad visual. Entre menor tamaño de archivo, más rápido cargará su página principal, en especial en dispositivos móviles. Almacene las imágenes en una carpeta `img/` en la raíz de su repositorio personal. Recuerde respetar la convención de nombres (archivo `README.md`). Finalmente, agregue las imágenes a control de versiones.

Avance de proyecto 25 [prj_images, 15 pts]

Elabore las imágenes de su aplicación web a nivel alto de fidelidad. Cuando creó su prototipo en papel, es probable que haya reutilizado las imágenes de sus esquemas de página {*wireframes*} de nivel medio de fidelidad. Si es necesario, edite las imágenes para que alcancen el nivel alto de fidelidad. Si su equipo cuenta con ayuda de diseño gráfico, solicite esta labor. Si obtiene imágenes de otra fuente, asegúrese de cumplir los derechos de uso y sus requisitos.

Extraiga cada imagen a un archivo independiente en la carpeta `img/` en la raíz de su proyecto. Deberá obtener un archivo por cada imagen que forme parte de su prototipo o esquemas de página. Si una imagen se usa repetidamente en distintos lugares, bastará almacenar sólo una copia. Si es necesario, edite las imágenes para que ocupen el menor espacio que no sacrifiquen su calidad visual. Asegúrese de respetar la convención de nombres de archivos de su proyecto. Agregue las imágenes a control de versiones.

4.5.3. Figuras

Cuando se inserta una imagen, vídeo, un listado de código u otro objeto en un documento, es muy común agregar algún texto que dé una explicación corta, los derechos de copia, u otros detalles del objeto. Este texto, a veces conocido como leyenda, rótulo, o título del objeto (*caption*), forma junto con el objeto una unidad, es decir, deben estar juntos en el documento. Por ejemplo, si la imagen se mueve de lugar en el documento, el rótulo debe viajar con ella, de lo contrario pierde su sentido.

(X)HTML5 provee el **elemento `<figure>`** para unir un objeto como imagen con su rótulo. El rótulo se indica con el **elemento `<figcaption>`** como se muestra en el extracto del Listado 42. El ejemplo completo compara imágenes escalares y vectoriales.

Listado 42. Agrupar una imagen con una leyenda en (X)HTML5 (figure.xhtml)

```
1 <figure id="imagen_escalar_4x">
2    <img src="flag_cr.png" width="344" height="360"/>
3    <figcaption>Una imagen escalar incrementada 4x</figcaption>
4 </figure>
```

En caso de proveerse una leyenda con el elemento `<figcaption>`, se convierte en opcional el proveer un texto alternativo con el atributo `alt` de `img`, como ocurrió en la línea 2 del Listado 42. Es importante reiterar que el elemento `<figure>` no está limitado sólo a imágenes, si no a cualquier objeto o trozo de texto que necesite una leyenda.

Ejercicio 55 [exc_image_linking, 5 pts]

Modifique la página principal de su sitio web personal para incluir las imágenes. Si éstas tienen un título, asegúrese de usar elementos `<figure>`. Revise el resultado en un navegador. No se ocupe de la apariencia por ahora, sino de la semántica y que su documento sea válido.

Avance de proyecto 26 [prj_homepage_images, 5 pts]

Edite la página principal de su proyecto. Enlace las imágenes que la componen. Asegúrese de usar web semántico, y por tanto, proveer textos alternativos o elementos `<figure>` si las imágenes tienen títulos. No se ocupe de la apariencia sino de la semántica y validez del documento ante los estándares web.

4.5.4. Icono de favoritos

Se ha convertido en una práctica común proveer un pequeño icono que ayuda al visitante a identificar la página web. A este icono se le suele llamar **icono de favoritos**, **icono de página** ó simplemente **favicon** (contracción de *favorite icon*), porque el navegador lo almacena junto a los favoritos, y lo despliega en la barra de direcciones, o en la pestaña donde está cargada la página. Usualmente el mismo icono se utiliza en todas las páginas de un sitio web, lo que le da identidad al sitio entero y por ende también se le llama **icono de sitio web** {*website icon*}.

El *favicon* es un archivo PNG, o un icono con extensión `.ico` que puede contener imágenes de varios tamaños. Más recientemente los navegadores permiten una imagen vectorial SVG. Normalmente son imágenes cuadradas de una potencia de 2, por ejemplo 16x16 pixeles, y utilizan fondo transparente.

El documento web debe indicar el *favicon* en su encabezado (X)HTML (`<head>`) utilizando el elemento vacío `<link>`, el tipo de imagen y su URI. El **elemento `<link>`** incrusta (enlaza) un recurso externo en el documento. El Listado 43 muestra tres métodos comunes para los tipos de imágenes descritos. Los métodos son excluyentes, por tanto, el autor sólo debe escoger la inclusión de acuerdo al formato de imagen escogida.

Listado 43. Varios métodos para incluir un icono de favoritos en una página web

```
<link rel="icon" type="image/png" href="path/to/favicon.png"/>
<link rel="icon" type="image/x-icon" href="path/to/favicon.ico"/>
<link rel="icon" type="image/svg+xml" href="path/to/favicon.svg"/>
```

Ejercicio 56 [exc_personal_favicon, 5 pts]

Elabore un icono de favoritos para su página principal y agréguelo a la carpeta `img/` de su repositorio personal. Enlace el icono en su página principal. Pruebe el resultado en un navegador. Revise que su página principal sea válida.

Avance de proyecto 27 [prj_favicon, 5 pts]

Elabore un icono de favoritos para su aplicación web. Si trabaja en equipo, discutan el diseño y escojan un formato de imagen permitido. Agregue la imagen a la carpeta `img/` de su repositorio de control de versiones. Enlace el icono en su página principal. Pruebe el resultado en un navegador. Revise que su página principal sea válida.

4.6. Tablas

(X)HTML permite organizar datos en forma tabular con el **elemento** `<table>`. Una tabla es una secuencia de filas declaradas con el **elemento** `<tr>` {*table record*}. Cada fila se compone de celdas declaradas con el **elemento** `<td>` {*table data*}. El Listado 44 muestra una tabla sencilla compuesta de cuatro filas y tres columnas.

Listado 44. Una tabla sencilla en (X)HTML compuesta de 4 filas y 3 columnas (table_min.xhtml)

```
1 <table>
2    <tr> <td>(1,1)</td> <td>(1,2)</td> <td>(1,3)</td> </tr>
3    <tr> <td>(2,1)</td> <td>(2,2)</td> <td>(2,3)</td> </tr>
4    <tr> <td>(3,1)</td> <td>(3,2)</td> <td>(3,3)</td> </tr>
5    <tr> <td>(4,1)</td> <td>(4,2)</td> <td>(4,3)</td> </tr>
6 </table>
```

En (X)HTML una tabla es una secuencia de filas `<tr>`, y éstas se pueden clasificar en filas de encabezado, de cuerpo, y de pie de la tabla. Las filas `<tr>` que conforman el **encabezado de la tabla (X)HTML** {*(X)HTML table header*} se escriben dentro del **elemento** `<thead>` como se hizo en las líneas 3 a 5 del Listado 45. El navegador les puede dar un tratamiento especial, como resaltar sus valores, y repetir el encabezado cuando al imprimir el documento la tabla se extiende por varias páginas.

En un encabezado se pueden tener valores normales dentro del elemento `<td>`, o encabezados de columnas con el **elemento** `<th>` {*table header*}. El autor puede proveer un valor abreviado en el

atributo `abbr` de una celda encabezado `<th>`. En caso de que el navegador no tenga suficiente espacio para desplegar la tabla completa, utilizará la abreviatura indicada.

Las filas `<tr>` que forman el **cuerpo de la tabla (X)HTML** {*(X)HTML table body*} se escriben dentro del **elemento `<tbody>`** como se hizo de la línea 6 a 17 en el Listado 45. Si el autor no especifica elementos `<thead>`, `<tbody>`, ni `<tfoot>` en una tabla `<table>`, el navegador debe suponer que todos los registros `<tr>` pertenecen a un cuerpo `<tbody>` implícito de la tabla. Si una columna, normalmente la primera, tiene encabezados de fila, se escribirán con el elemento `<th>`.

Las filas `<tr>` que forman el **pie de tabla (X)HTML** {*(X)HTML table footer*} se escriben dentro del **elemento `<tfoot>`**, como se hizo en las líneas 18 a 20 del Listado 45. Los pies de tabla se usan principalmente para mostrar resúmenes de los datos en el cuerpo de la tabla.

Listado 45. Encabezado, cuerpo y pie de tabla en (X)HTML (extracto de table_tiobe.xhtml)

```
1 <table>
2    <caption>Lenguajes de programación más populares según TIOBE</caption>
3    <thead>
4       <tr> <th>#</th>  <th>Lenguaje</th>     <th>Popularidad</th> </tr>
5    </thead>
6    <tbody>
7       <tr> <th>1</th>  <td>Java</td>         <td>17.11%</td> </tr>
8       <tr> <th>2</th>  <td>C</td>            <td>17.09%</td> </tr>
9       <tr> <th>3</th>  <td>C#</td>           <td> 8.24%</td> </tr>
10      <tr> <th>4</th>  <td>C++</td>          <td> 8.05%</td> </tr>
11      <tr> <th>5</th>  <td>Objective-C</td>  <td> 7.74%</td> </tr>
12      <tr> <th>6</th>  <td>PHP</td>          <td> 5.56%</td> </tr>
13      <tr> <th>7</th>  <td>Visual Basic</td> <td> 4.37%</td> </tr>
14      <tr> <th>8</th>  <td>JavaScript</td>   <td> 3.39%</td> </tr>
15      <tr> <th>9</th>  <td>Python</td>       <td> 3.29%</td> </tr>
16      <tr> <th>10</th> <td>Perl</td>         <td> 2.70%</td> </tr>
17   </tbody>
18   <tfoot>
19      <tr> <th colspan="2">Total</th>         <td>77.54%</td> </tr>
20   </tfoot>
21 </table>
```

Es válido tener en una tabla celdas sin contenido. También una celda (de datos o encabezado) puede ocupar dos o más columnas, dos o más filas, o ambas. La cantidad de columnas que ocupa una celda se indica con el atributo `colspan` y la cantidad de filas que ocupa una celda se indica con el atributo `rowspan`. El valor por defecto para ambos atributos es 1. En la línea 19 del Listado 45 se indica que la celda con el valor "Total" ocupa dos columnas: la columna `#` y la columna `Lenguaje`.

Es conveniente que el autor provea una leyenda explicando al lector el objetivo de la tabla, con el elemento `<caption>` como se hizo en la línea 2 del Listado 45. Con estilos se puede controlar si la leyenda debe aparecer arriba o debajo de la tabla. Una forma alternativa en (X)HTML5 es introducir tanto la tabla como su leyenda en un elemento `<figure>` como se explica en la Sección 4.5.3.

Por defecto el navegador no dibuja bordes en las celdas ni en la tabla. Esto puede ser confuso para el autor, pero es una responsabilidad de las hojas de estilo (CSS) y no de (X)HTML. Si se requiere de

bordes, el autor puede rápidamente agregar en el encabezado de su documento (X)HTML reglas de estilo como las líneas 7 y 8 del Listado 46. La primera regla dibuja un borde en cada celda de la tabla y la tabla misma. La regla de la línea 8 fusiona los bordes {*collapse*} para que las celdas no parezcan rectángulos aislados. Estas reglas se aplicaron al ejemplo completo) del Listado 45.

Listado 46. Dos reglas de estilo CSS para agregar bordes en las tablas de un documento web.

```
1  <!DOCTYPE html>
2  <html lang="es" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4     <meta charset="utf-8"/>
5     <title>...</title>
6     <style type="text/css">
7        table, th, td { border: 1px solid black; }
8        table { border-collapse: collapse; }
9     </style>
10 </head>
11 <body>
12 ...
13 </body>
14 </html>
```

Ejercicio 57 [exc_secondary_pages, 20 pts]

Escriba las páginas secundarias de su sitio web personal. En cada subcarpeta de contenido de su repositorio (`intro/`, `xml/`, `xhtml/`), cree una página web de índice. Puede copiar la página principal de su sitio web (`index.xhtml`) a cada carpeta (por ejemplo, `intro/index.xhtml`). Esta copia tiene la ventaja de mantener las regiones globales del sitio web (encabezado de sitio, navegación del sitio, pie del sitio, ...). Sin embargo, introduce redundancia no controlada, que luego será corregida con programación del lado del servidor.

Ajuste el título de cada página secundaria. Adapte el contenido para listar los ejercicios que haya resuelto en cada subcarpeta. Cada ejercicio debería producir un enlace hacia la carpeta donde se encuentra. Idealmente sólo debería tener que modificar el contenido del elemento `<main>`.

Su código XHTML debe ser lo más semántico posible. Revise que pueda navegar a lo largo del sitio usando sus páginas índice. Verifique que cada página es XHTML5 válida.

Avance de proyecto 28 [prj_secondary_pages, 25 pts]

Elabore las páginas secundarias de su aplicación web en XHTML5. Utilice el mapa del sitio en su `design.md` como guía para saber cuáles páginas conforman su aplicación web. Para cada una de ellas puede duplicar la página principal con el fin de mantener las regiones globales del sitio web.

Utilice los esquemas de página en su `design.md` como guía para escribir el contenido de cada página. En teoría sólo debería modificar la sección `<main>` para cada página secundaria. Debe prestar atención a los contenidos de los esquemas de página y no a su apariencia. Es decir, a los recursos que componen cada página del sitio, y no a los estilos ni comportamientos.

Por ejemplo, para las páginas de ayuda de la aplicación, el contenido a escribir en el elemento `<main>` será el texto de ayuda, las imágenes explicativas, o los enlaces a los vídeos. Para las páginas más dinámicas, el contenido son los recursos que componen la aplicación. Las fotografías o videos de las pruebas de usabilidad o del protocolo de paso de mensajes pueden resultar especialmente útiles para identificar estos contenidos. Los recursos (imágenes, textos) que tenía sobre el escritorio de pruebas antes de iniciar la sesión representan los recursos que deben conformar el contenido de la página web. Estos contenidos serán manipulados posteriormente por los programas en JavaScript.

Su código XHTML debe ser lo más semántico posible. Cada vez que termine una página del sitio, recuerde probar su navegación y su validez ante el estándar XHTML5.

4.7. Objetos multimedia

Disponible en la edición completa de este material.

4.8. Formularios web

Disponible en la edición completa de este material.

Parte III. Presentación

Un sitio web se compone de la tripleta contenido, presentación, y comportamiento. El contenido es preparado por expertos en contenido, como escritores, arquitectos de la información, filólogos, y en especial los expertos en el dominio o temática del sitio web. El contenido se escribe finalmente en documentos (X)HTML.

La presentación es típicamente asociada con la estética de las páginas web, compuesta de imágenes y estilos preparados por diseñadores gráficos. Sin embargo, la presentación tiene un propósito más general, de ayudar a los usuarios a encontrar y comprender el contenido del sitio web de forma eficiente. La presentación web se realiza con hojas de estilo en cascada.

El comportamiento tiene el mismo propósito que la presentación, de asistir a los usuarios a encontrar y comprender el contenido del sitio web, pero a través de programas de computadora, normalmente preparado por expertos en computación.

Capítulo 5. Hojas de estilo en cascada CSS

Un documento (X)HTML sólo debe tener contenido, es decir, los datos que el autor quiere publicar y su estructura. La apariencia o presentación de un documento web debe especificarse en un archivo externo llamado *hoja de estilos* que indica cómo debe formatearse la información en la pantalla, en papel u otro medio de salida.

La separación del contenido y la presentación tiene grandes ventajas. Una hoja de estilos puede reutilizarse en todas las páginas de su sitio web, lo que les da uniformidad, facilita el mantenimiento, y ahorra ancho de banda. Una misma o varias páginas web pueden ajustarse a diferentes necesidades simplemente cambiando la hoja de estilos. Así el autor puede tener un juego de hojas de estilo en su sitio web, una para cada necesidad: un diseño avanzado para el navegador, grandes contrastes para personas con necesidades especiales, estilos amigables para impresión en papel {*printer friendly*}, entre otros. De esta forma, una misma página web puede verse en formas distintas sin cambiar la información que contiene.

El contenido de este capítulo cambiará significativamente. El texto aquí provisto es una réplica de la edición anterior de este material.

5.1. Generalidades de las hojas de estilo

Una hoja de estilos es un archivo de texto que contiene reglas de formateo de elementos utilizando un lenguaje especial conocido como **hojas de estilo en cascada** {CSS, *Cascading Style Sheets*}, el cual es estándar y definido por el W3C. Por ejemplo, la siguiente regla hace que todos los títulos de un documento web sean de color verde:

```
h1, h2, h3, h4, h5, h6 { color: green; }
```

En general una **regla** CSS tiene la siguiente sintaxis:

```
selector { declaration }
```

El **selector CSS** {*CSS selector*} determina cuáles elementos serán afectados por los estilos definidos en la declaración, y puede ser uno o varios elementos separados por comas. La **declaración CSS** {*CSS declaration*} es una lista de parejas `propiedad: valor` que serán aplicadas a los elementos listados en el selector. Dichas parejas se separan por el símbolo punto y coma (`;`). Es opcional terminar en punto y coma la última pareja. Con esto la sintaxis se expande a:

```
element1, element2, ..., elementN
{
   property1: value1;
   property2: value2;
   ...
   propertyN: valueN;
}
```

Es muy conveniente que escriba comentarios explicando la intención de cada regla, o al menos de aquellas no triviales. Los comentarios utilizan la misma notación del lenguaje de programación C:

```
/* Las definiciones deben estar en negritas y no en itálicas como ocurre en ciertos
navegadores. Globalmente se usa verde para definiciones y títulos, azul para enlaces. */
dfn
{
   font-weight: bold;
   font-style: normal;
   color: #004444;
}
```

5.1.1. Ubicación de las reglas de estilo

La recomendación (X)HTML permite declarar reglas de estilo en los siguientes cinco lugares, ordenados descendentemente por prioridad. Es decir, si dos reglas están en conflicto, las que tienen números más bajos serán consideradas.

1. En una hoja de estilos externa provista por el visitante (no por el autor).
2. En el elemento mismo, con el atributo `style`.
3. En el encabezado del documento web, con el elemento `<style>`.
4. En un archivo `.css` externo el cual se liga al documento con el elemento `<link>`.
5. En la implementación del navegador web.

De los cinco lugares para especificar estilos en la lista anterior, el Consorcio Web (W3C) desalienta el uso del segundo y tercero, ya que el documento web debe almacenar contenido únicamente y no estilos. El primer y último lugar listado anteriormente están fuera del control del autor. Por eso, se aconseja al autor declarar las reglas de estilo siempre en una o varias hojas de estilos `.css` externas al documento web (lugar 4).

El Listado 47 muestra un ejemplo que una página web HTML5 que declara estilos en los tres lugares donde puede un autor. La primera regla de estilo se declara en la línea 7, que indica al navegador pintar los párrafos de rojo y centrar sus textos en la página. Las líneas 6 a 8 declaran un elemento `<style>` dentro del encabezado (`<head>`) del documento (X)HTML, que corresponde al segundo lugar de acuerdo a la lista anterior. El encabezado de documento puede tener cero o más elementos `<style>`.

Listado 47. Ejemplo de ubicación de reglas: documento HTML (rules_location.html)

```
1  <!DOCTYPE html>
2  <html lang="en">
3     <head>
4        <meta charset="utf-8">
5        <title>CSS rules locations</title>
6        <style>
7           p { color: red; text-align: center; }
8        </style>
9        <link rel="stylesheet" media="screen" href="rules_loc_screen.css"/>
10       <link rel="stylesheet" media="print" href="rules_loc_print.css"/>
11    </head>
12    <body>
13       <p style="color: blue">L'amour est bleu...
14       <p>...even if you print it.
15    </body>
16 </html>
```

Listado 48. Ejemplo de ubicación de reglas externas para pantalla (rules_loc_screen.css)

```
1 p { color: green; font-style: italic; }
```

Listado 49. Ejemplo de ubicación de reglas externas para impresión (rules_loc_print.css)

```
1 p { color: gray; font-size: 3em; }
```

La línea 9 del Listado 47 incrusta una hoja de estilos externa con el elemento `<link>`. El atributo `rel` indica la relación entre el recurso incrustado y la página web. El valor `rel="stylesheet"` indica que el recurso incrustado es una hoja de estilos en cascada. El atributo `href` contiene un URI hacia la hoja de estilos. Una hoja de estilos es un archivo de texto con extensión `.css` que contiene reglas.

La línea 9 del Listado 47 incrusta las reglas del archivo `rules_loc_screen.css` ubicado en la misma carpeta que el archivo HTML en el documento. El Listado 48 muestra el contenido de la hoja de estilos `rules_loc_screen.css` que se compone de sólo una regla, la cual pinta a los párrafos de verde y su tipografía es itálica.

La línea 10 del Listado 47 incluye una segunda hoja de estilos, con una diferencia a la línea 9. El atributo `media` del elemento `<link>` indica si las reglas de estilo aplican a un medio específico, como la pantalla (`screen`), la impresora (`print`), programas de lectura sintetizada (`aural`), computadoras de mano (`handheld`), televisores (`tv`) y otros. En la línea 9 del Listado 47, un navegador incrustará la hoja `rules_loc_screen.css` y los aplicará cuando el documento web es visualizado en una pantalla. Si el usuario decide imprimir el documento, el navegador solicitará el recurso `rules_loc_print.css` (línea 10) y los aplicará en la copia que enviará a la impresora.

El atributo `media` es opcional, y si no se especifica en un elemento `<link>`, se asume que esa hoja de estilos aplicará a todos los medios, lo que equivale a darle el valor `media="all"`. El atributo `media` puede

contener varios valores separados por comas, de esta forma, una hoja de estilos puede aplicarse a varios medios simultáneamente.

Nótese que el encabezado `<head>` admite combinar hojas de estilo externas y elementos `<style>` en cualquier orden. La especificación CSS indica que el navegador debe respetar este orden, de tal forma que las reglas de estilo que se encuentran en un archivo `.css`, tienen el mismo efecto que si se escribiesen directamente en un elemento `<style>` del encabezado.

En otras palabras, cuando un navegador incrusta una hoja de estilos, obtiene el archivo `.css`, carga su contenido, y reemplaza el elemento `<link>` por un elemento `<style>` cuyo contenido es el contenido del archivo `.css`. Finalmente, todos los elementos `<style>` en el encabezado son "concatenados" en el orden en que fueron declarados, y el resultado final es como si todas las reglas de estilo hubieren sido escritas en un único elemento `<style>`. El Listado 50 muestra el resultado tras aplicar este procedimiento al Listado 47.

Listado 50. Documento HTML tras incrustar hojas de estilo (rules_location_resolv.html)

```
1  <!DOCTYPE html>
2  <html lang="en">
3     <head>
4        <meta charset="utf-8">
5        <title>CSS rules locations</title>
6        <style>
7           @media all
8           {
9              p { color: red; text-align: center; }
10          }
11          @media screen /* from rules_loc_screen.css */
12          {
13             p { color: green; font-style: italic; }
14          }
15          @media print /* from rules_loc_print.css */
16          {
17             p { color: gray; font-size: 3em; }
18          }
19       </style>
20    </head>
21    <body>
22       <p style="color: blue">L'amour est bleu...
23       <p>...even if you print it.
24    </body>
25 </html>
```

La línea 13 del Listado 47 (o línea 22 del Listado 50) declara una regla de estilo en el *atributo* `style` del elemento al que se quiere aplicar. En este caso, todas las propiedades aplicarán únicamente a dicho elemento por lo que no es necesario especificar un selector, ni agrupar las propiedades dentro de llaves `{ }`. Corresponde al primer lugar donde se pueden especificar reglas de la lista de ubicaciones al inicio de este apartado.

Tras incrustar las reglas en el Listado 50 puede verse que hay conflictos por los colores de los párrafos.

El lector puede tratar de predecir cuáles colores y estilos serán visualizados en una pantalla o al imprimir, y corroborar sus resultados al realizar estas operaciones con un navegador. Los conflictos entre las reglas son resueltos por el principio de cascada de las hojas de estilo.

5.1.2. El principio de cascada

En vista de que hay cinco lugares opcionales donde se especifican estilos, puede ocurrir que para una propiedad no hayan reglas, o bien apliquen varias. El navegador necesita un algoritmo que especifique cómo actuar en estos casos, al cual se le llama el *principio de cascada* y consta de otros tres principios.

1. Si dos o más reglas compiten para darle estilo a una misma propiedad, recibirá mayor prioridad aquella que tenga mayor especificidad en su selector. A esto se le llama **principio de especificidad** {*specificity principle*}.
2. Si dos reglas compitiendo por la misma propiedad tienen igual especificidad, se tomará la que aparezca de último, es decir, tendrá más peso aquella que se encuentre más cerca del elemento. A esto se le llama el **principio de ubicación** {*location principle*}.
3. Cuando por el contrario, no se especifican reglas para una propiedad, el navegador debe seguir un principio más, el **principio de herencia** {*inheritance principle*}, el cual indica que ante la ausencia de una regla de estilo, el navegador debe aplicar el estilo del elemento padre. No todos las propiedades son heredables por defecto. Por ejemplo, la fuente lo es pero no el margen. De esta forma, al asignar la fuente al elemento `<body>`, todos los párrafos la heredan, lo cual es deseable, pero si se asigna un margen grande al `<body>` para mantener una separación visual con los bordes de la ventana del navegador, sería indeseable si este borde se aplicara entre párrafo y párrafo.

El uso de los tres principios: ubicación, especificidad y herencia para determinar cuál regla debe aplicar a una propiedad de estilo de un elemento se conoce como **principio de cascada** {*cascading principle*}. Los navegadores implementan este principio dándole puntos o pesos a cada regla de estilo siguiendo un cálculo propuesto en la especificación CSS, que cualitativamente se puede resumir en lo siguiente:

Ante la ausencia de una regla, el estilo se hereda del elemento padre si la propiedad es heredable, si no, se usa el estilo por defecto del navegador. Si dos o más reglas compiten por darle estilo a una misma propiedad, se tomará aquella con mayor especificidad de su selector, independientemente de dónde esté ubicada. Con dos reglas de igual especificidad, ganará la que aparezca de último.

A modo de ejemplo, el listado Listado 51 muestra un documento web que tiene dos párrafos y estilos definidos en tres lugares: una hoja de estilos externa (`cascading1.css`) enlazada en la línea 4; un elemento `<style>` en el encabezado (líneas 5 a 7), y un atributo `style` en el segundo párrafo (línea 18). Se estudiará el efecto de estos estilos en tres propiedades: el color, la fuente y el margen.

Listado 51. Ejemplo de estilos en cascada (cascading.xhtml)

```
1  <!DOCTYPE html>
2  <html xml:lang="es" xmlns="http://www.w3.org/1999/xhtml">
3  <head>
4      <meta charset="utf-8"/>
5      <title>Estilos en conflicto: cascada</title>
6      <link rel="stylesheet" type="text/css" href="cascading1.css"/>
7      <style type="text/css">
8          p { color: teal; }
9      </style>
10 </head>
11
12 <body>
13     <p>Este es el primer párrafo. Ninguna regla de estilo para <code>p</code> especifica la fuente,
   por lo que <em>hereda</em> la fuente de su elemento padre <code>body</code>. Aunque hay una hoja de
   estilos externa que especifica que el color del texto de los párrafos debe ser rojo, éste es <
   strong>verde azulado</strong> porque hay una regla en el encabezado del documento, que está más cerca
   de este párrafo.</p>
14
15     <p style="color: green;">Este es el segundo párrafo. Tiene un atributo de estilo que sobrescribe
   el color del texto a <strong>verde</strong>, ya que por <em>especificidad</em> está más cerca del
   elemento. Ninguna regla de estilo formatea la separación vertical entre párrafos, por lo que se
   aplica la que el navegador tenga implementada internamente.</p>
16 </body>
17 </html>
```

Listado 52. Estilos incrustados en el listado anterior (cascading1.css)

```
body
{
    font-family: Verdana, Geneva, Arial, Helvetica, sans-serif;
    margin: 60px;
    color: blue;
}

p { color: red; }
```

La fuente es una propiedad heredable de acuerdo a la especificación CSS. La mayoría de navegadores asumen una fuente con *serifas*, como `Times New Roman`. Al cargar la hoja de estilos el navegador procesa la regla que indica que la fuente del cuerpo (`<body>`) del documento debe mostrarse en `Verdana`, que es por el contrario una fuente sin serifas (`sans-serif`); cualquier regla del autor tiene más prioridad que el valor por defecto del navegador y por ende el navegador debe usar `Verdana` para formatear el texto del `<body>`, el cual es vacío (`<body>` no tiene texto directamente, sino dos elementos `<p>` hijos).

Al desplegar el primer párrafo (línea 11), el navegador no encuentra una regla de fuente para dicho elemento de párrafo. Como es una propiedad heredable, el navegador toma la fuente del padre del párrafo que es el elemento `<body>`, y por ende despliega el texto del párrafo en `Verdana`. Esto mismo ocurre recursivamente para los elementos hijos del párrafo, como `<em>`, `<strong>` y `<code>`; y luego para

el párrafo siguiente. De esta forma, todo el documento aparece en `Verdana` como el autor podría esperar.

La propiedad de margen (`margin`) no es heredable de acuerdo a la especificación CSS. El navegador web asume un margen nulo o minúsculo para el `<body>`, por lo que el texto se suele pegar con los extremos de la ventana del navegador. Al encontrar la regla `margin: 2.5cm`, el navegador la aplica al elemento `<body>`, haciendo que el margen del documento sea de 2.5 centímetros. Sin embargo, al no ser una propiedad heredable, el navegador no aplica un margen de 2.5 centímetros a los párrafos ni a los elementos `<em>`, `<strong>` y `<code>`, sino que asume el margen por defecto, que es de unos cuantos pixeles para párrafos y de 0 pixeles para los otros elementos citados.

El color es una propiedad heredable según la especificación CSS. El navegador no encuentra una regla para el color de `<body>` y asume el por defecto del navegador compite por la misma propiedad: establecer el color de los párrafos en verde azulado (`teal`). Estas dos reglas están en conflicto, y ganará la que tenga mayor especificidad en el selector, sin embargo ambas tienen el mismo selector; y en tal caso el navegador deberá aplicar la que aparezca de último de acuerdo al *principio de ubicación*, y por ende, los parrafos aparecerán en verde azulado.

Al pintar el segundo párrafo que inicia en la línea 18 del listado Listado 51, el navegador encuentra una regla para la propiedad `color` en su atributo `style`. De acuerdo al *principio de especificidad* esta regla tiene la mayor prioridad y por ende, el segundo párrafo aparecerá en verde simple en lugar de verde azulado.

Ejercicio 58 [exc_css_link, 5 pts]

Cree una carpeta `css/` en la raíz de su repositorio personal de control de versiones. Cree en ella un archivo `screen.css`. Agregue algunas reglas de estilo de prueba en cada uno de los archivos.

Para cada página XHTML de su sitio personal, enlace la hoja de estilos. Puede usar direcciones absolutas de la forma `<link href="/css/screen.css"···/>`, la cual es idéntica para todas las páginas del sitio, pero dependen de un servidor web. Verifique que al incrustar la hoja de estilos, éstos sean automáticamente aplicados a la página. Asegúrese de que las páginas y estilos sean válidos ante los estándares web.

Avance de proyecto 29 [prj_css_link, 5 pts]

Cree una carpeta `css/` en la raíz del repositorio de su proyecto. Cree en ella dos archivos: `global.css` y otra con el nombre de su aplicación y extensión `.css`. Agregue algunas reglas para el elemento `<body>`, por ejemplo, la tipografía sea sin serifas, o darle un color distinto de negro.

Para cada página XHTML de su mapa del sitio, enlace la hoja de estilos global. Para las páginas XHTML de su aplicación web, enlace la hoja de estilos homónima luego de la global. Puede usar direcciones absolutas de la forma `<link href="/css/global.css"···/>`, la cual es idéntica para todas las páginas del sitio, pero dependen de un servidor web. Verifique que al incrustar la hoja de estilos, éstos sean automáticamente aplicados a cada página. Asegúrese de que las páginas y estilos sean válidos ante los estándares web.

5.1.3. El operador @

Disponible en la edición completa de este material.

5.2. Selectores

Selectors Level 4 W3C Recommendation.

Disponible en la edición completa de este material.

5.3. Valores de propiedades

Los valores que pueden tomar las propiedades son de muy diversa naturaleza. Sin embargo, existen algunos dominios comunes que se resumen en la Tabla 30 y a los cuales se les hará referencia luego en este documento.

Tabla 30. Dominios comunes de ciertas propiedades CSS

Dominio	Descripción	Ejemplos
`inherit`	Se le da el valor `inherit` a una propiedad cuando explícitamente se quiere especificar que esa propiedad tome el mismo valor que la del elemento padre.	`p { margin: inherit; }`

`<length>`	Son magnitudes compuestas siempre de un valor y una unidad, como `2.5cm` las cuales no deben estar separadas por espacios. La unidad se puede omitir sólo cuando la magnitud es `0`. Hay dos tipos de unidades:
	• **Absolutas**: centímetros (`cm`), milímetros (`mm`), pulgadas (`in`), puntos (`pt`) y picas (`pc`).
	• **Relativas**: tamaño de fuente (`em`), tamaño de la letra "x" en la fuente actual (`ex`), pixeles (`px`), y porcentajes del valor de la propiedad del elemento padre (`%`).
		`body { margin: 2.5cm; }`
		
		`h1`
		`{`
		`  margin-top: 2em;`
		`  margin-bottom: 0;`
		`}`
		
		`pre { font-size: 85%; }`

`<number>`	Son números sin unidades.	`div.content { z-index: 2; }`

`<url>`	URL hacia otro recurso, como una imagen, sonido, etc. Si el URL es relativo, lo será con respecto a la hoja de estilos y no al documento (X)HTML. Encerrar el URL entre comillas es opcional. No debe separarse la palabra `url` del paréntesis que abre.	`body`
		`{`
		`  background: url(water.png) repeat;`
		`}`

Dominio	Descripción	Ejemplos

`<color>`	Hay tres formas de especificar colores en CSS:
	• **Nombres predefinidos**. Hay 17 colores predefinidos en CSS y 140 en CSS3 como se explica más adelante. Los nombres no son sensitivos a mayúsculas y no se deben especificar dentro de comillas.
	• **Notación rgb()**. Permite construir un color con la cantidad de rojo, verde y azul, sea con valores entre 0 y 255 (o su correspondiente hexadecimal), o porcentajes. No se debe separar la palabra `rgb` de los paréntesis que abren.
	• **Notación #**. Se especifica las cantidades de rojo, verde y azul en notación hexadecimal antecedidos por un símbolo de número, de la forma `#rrggbb`, o `#rgb` si los dígitos de cada componente se repiten. Las letras hexadecimales pueden estar en mayúsculas o minúsculas o ambas.
		`body { background: black; }`
		`h1 { color: rgb(245, 255, 250); }`
		`h2 { color: rgb(100%, 75%, 50%); }`
		`pre`
		`{`
		`  border: solid 1px;`
		`  border-color: #80c0a0 #000 #000 #80c0a0;`
		`  background: #EFE;`
		`}`

Disponible en la edición completa de este material.

5.4. Módulos y propiedades

- CSS Snapshot
- CSS Working Group Editor Drafts
- CSS CheatSheet

Core:

1. CSS Syntax Module Level 3
2. Selectors Level 3
3. CSS Cascading and Inheritance Level 3
4. CSS Values and Units Module Level 3
5. CSS Custom Properties for Cascading Variables
6. CSS Namespaces Module Level 3

Box model:

1. CSS Box Model Module Level 3. Margin. Padding. Border.
2. CSS Backgrounds and Borders Module Level 3. Border styles. Background.

Positioning:

1. CSS Positioned Layout Module Level 3. Coordinates(top, left, bottom, right), relative positioning, absolute positioning.
2. CSS Box Alignment Module Level 3. Align content. Flex layout. Grid layout.

3. CSS Overflow Module Level 3
4. CSS Display Module Level 3
5. CSS Inline Layout Module Level 3

Color:

1. CSS Color Module Level 3

Text:

1. CSS Text Decoration Module Level 3
2. CSS Text Module Level 3
3. CSS Font Loading Module Level 3
4. CSS Fonts Module Level 3

Element:

1. CSS Images Module Level 3
2. CSS Table Module Level 3
3. CSS Lists Module Level 3
4. CSS Counter Styles Level 3

Other:

1. CSS Basic User Interface Module Level 3 (CSS3 UI)
2. CSS Conditional Rules Module Level 3
3. CSS Fragmentation Module Level 3
4. CSS Generated Content Module Level 3
5. CSS Intrinsic & Extrinsic Sizing Module Level 3
6. CSS Paged Media Module Level 3
7. CSS Writing Modes Level 3

5.4.1. El modelo de caja

CSS Box Model Module

5.4.2. El modelo de fuente

Disponible en la edición completa de este material.

5.4.3. El modelo de color

Disponible en la edición completa de este material.

5.4.4. El modelo de visualización

Disponible en la edición completa de este material.

Ejercicio 59 [css_personal_website_style, 25 pts]

Haga que las páginas de su sitio web personal reflejen el diseño elaborado en los esquemas de página {*wireframes*}. Escriba sus reglas en un archivo `.css` para pantalla y asegúrese de que todas las páginas de su sitio web lo enlacen. Asegúrese de que su sitio web personal se adapte a los dispositivos de los visitantes {*responsive design*}.

Avance de proyecto 30 [prj_project_style, 25 pts]

Haga que las pantallas de su proyecto reflejen el diseño elaborado en los esquemas de página {*wireframes*}. Escriba sus reglas en uno o varios archivos `.css` y asegúrese de que todas las páginas de su aplicación web lo enlacen. Asegúrese de que su aplicación se adapte a los dispositivos de los visitantes {*responsive design*}.

Parte IV.

Comportamiento

Los dos siguientes capítulos usan los paradigmas de programación para resolver problemas de la web. El capítulo 10 crea soluciones del lado del cliente, conocido como *frontend Javascript* o *client-side Javascript*. El capítulo 11 crea soluciones del lado del servidor, conocido como *backend Javascript* o *server-side Javascript*. El capítulo 12 crea soluciones en ambos lados, a lo que se le llamará *Javascript distribuido* {*distributed Javascript*}.

Capítulo 6. Comportamiento con JavaScript

JavaScript es un lenguaje interpretado por el navegador web que permite al autor manipular dinámicamente el documento, sus estilos y la ventana misma del navegador, para hacer la experiencia del lector más natural y amena. JavaScript fue creado en 1995 por Brendan Eich cuando trabajaba para Netscape. Un año después Microsoft produjo su propia versión llamada JScript. También en 1996, Netscape sometió JavaScript a la {*European Computer Manufacturer's Association*} para consideración como estándar de la industria, el resultado fue ECMAScript.

ECMAScript es un estándar internacional de un lenguaje genérico de "scripting" para extender la funcionalidad de un programa cualquiera, no sólo navegadores web. Hay un número creciente de implementaciones basadas en ECMAScript que además extienden su funcionalidad, como el inicial JavaScript de Netscape, JScript de Microsoft, ActionScript de Macromedia (adquirida por Adobe), SpiderMonkey y Rhino de Mozilla, etc. Sin embargo, el nombre ECMAScript no tomó popularidad, y cuando la mayoría de la gente dice "JavaScript" está haciendo referencia al lenguaje en forma general, no a la implementación de Netscape, y así se hará en este documento.

Cualquier software que quiera permitir al usuario automatizar tareas propias, en lugar de crear un nuevo lenguaje de guiones {*scripting*}, puede echar mano de JavaScript. De esta forma, JavaScript es por naturaleza un lenguaje genérico. Un desarrollador que conozca este lenguaje, puede aprovechar su conocimiento para automatizar variedad de aplicaciones, como ocurre en la actualidad en programación de dispositivos móviles, acceso a bases de datos orientadas a documentos, animación digital y otros. Pero su uso más difundido ha sido históricamente la web, en la programación en el lado del cliente y más recientemente en el servidor web con la tecnología Node.js.

6.1. Ambientes de ejecución

Netscape creó JavaScript imitando características de Java para aprovechar la promoción que Sun Microsystems realizaba de éste último hacia 1995. Por consecuencia, la sintaxis de JavaScript es familiar para programadores de C/C++ y Java. Es sensitivo a mayúsculas y minúsculas, por lo que resulta más consistente con XHTML que con HTML. Aunque no es obligatorio que cada instrucción en JavaScript termine en punto y coma, se considera buena práctica siempre hacerlo. Los comentarios pueden ser de una o varias líneas como el Listado 53.

Listado 53. Modo estricto y comentarios de JavaScript

```
1 'use strict';
2 // comentario hasta el final de línea
3 /* comentario que puede
4    extenderse varias líneas */
```

El código JavaScript es un guión. Es decir, el intérprete ejecutará las líneas una a una en el orden en que aparezcan. La instrucción `'use strict';` (con comillas simples o dobles) indica al intérprete de JavaScript que sólo permita un subconjunto del lenguaje conocido como **modo estricto** {*strict mode*}, y deshabilite extensiones de compatibilidad hacia atrás que permiten ejecutar código con prácticas que posteriormente se han considerado indebidas. En ES6 el modo estricto es obligatorio e implícito si el código está en un archivo `.js` (llamado módulo JavaScript), por tanto sólo es necesario incluir la instrucción `'use strict';` para código dentro del documento (X)HTML. El código JavaScript puede ejecutarse en varios contextos relacionados con la tecnología web:

1. Con el pseudoprotocolo `javascript:`.
2. En un evento intrínseco del documento web.
3. En el elemento `<script>` de un documento web.
4. La consola web del navegador.
5. En un archivo `.js` externo asociado al documento web.
6. La consola web de Node.js (REPL).
7. En un archivo `.js` ejecutado por Node.js.

Dos contextos anteriores corren código JavaScript en el ambiente Node.js, los restantes cinco corren en el ambiente provisto por el navegador, el cual ofrece acceso restringido al documento y otros recursos de la ventana donde éste se muestra. Node.js permite al código JavaScript correr en el ambiente del sistema operativo, como lo hace un programa de C/C tradicional. El intérprete de Node.js es un programa implementado en C que ofrece funcionalidades del sistema operativo al código JavaScript, por tanto puede correr en una computadora de escritorio tradicional o en un servidor web. A continuación se presentan brevemente cada contexto de ejecución.

6.1.1. El pseudoprotocolo `javascript:`

Se puede ejecutar código JavaScript en la barra de direcciones del navegador, en el destino de un enlace, o cualquier otro lugar donde se pueda escribir un URI. Se utiliza el pseudoprotocolo `javascript:` seguido por una o varias instrucciones JavaScript separadas por punto y coma, como los ejemplos del Listado 54. El resultado de invocar estas instrucciones, si lo hay, se toma como un texto que es desplegado en la ventana del navegador. Sin embargo, algunos navegadores como Chrome y Safari, ejecutan el código en el pseudoprotocolo pero no reemplazan el documento actual por el resultado de la expresión, lo que produce la sensación de ser ignorados.

Listado 54. Ejemplos del pseudoprocolo JavaScript (pseudoprotocol.xhtml)

```
1 javascript:11%254
2 javascript:document.getElementById('advertising').remove();
3 javascript:function contrast(){for (let el of document.querySelectorAll('*')) {
  el.style.backgroundColor='black'; el.style.color='white'; el.style.borderColor='white'; } }
  contrast();
```

El ejemplo de la línea 1 del Listado 54 muestra que el pseudoprotocolo debe ser un URI válido, por lo que caracteres especiales (como el operador módulo) deben ser codificados. Una utilidad pseudoprotocolo `javascript:` es que el código se puede agregar a los favoritos del navegador y

ejecutarlos sobre cualquier página, incluso de terceros. Por ejemplo, la última línea del Listado 54 cambia todos los elementos del documento a blanco sobre fondo negro.

6.1.2. Código en los eventos intrínsecos

Se puede especificar código JavaScript en los eventos intrínsecos de ciertos elementos, tales como `onload`, `onmouseover` y `onclick`, como se ilustra en el Listado 55. Este código JavaScript es ejecutado únicamente si el evento es accionado. El código para los eventos intrínsecos no se debe incluir en el documento (X)HTML, sino que debe ser asignados dinámicamente desde un archivo `.js`, como se estudiará en un capítulo posterior.

Listado 55. Código ejecutado cuando un evento ocurre como presionar un botón (onclick_button.xhtml).

```
1 <body>
2    <h1>Eventos en JavaScript</h1>
3    <button onclick="alert('No me toque');">Un botón!</button>
4 </body>
```

6.1.3. El elemento `<script>` y la consola web

Se puede escribir código JavaScript en el contenido del elemento `<script>`, el cual debe ser hijo directo de `<head>` o `<body>`. Al correr Listado 56 muestra los cuadrados de los primeros 20 números naturales en el documento resultante.

Listado 56. Un elemento `<script>` en el cuerpo del documento (squares_list1.html)

```
1 <body>
2    <h1>Cuadrados naturales</h1>
3    <script type="text/javascript">
4       'use strict';
5       document.write('<ul>\n');
6       for ( let n = 1; n <= 20; ++n )
7          document.write('. ', n, '<sup>2</sup> = ', (n * n), '\n');
8       document.write('</ul>\n');
9    </script>
10 </body>
```

Como se puede deducir del Listado 56, JavaScript interpreta el texto entre apóstrofes (`''`) y entre comillas (`""`) como cadenas de caracteres. Las variables no se declaran precedidas por su tipo de datos, sino por la palabra reservada `let`. El ciclo `for` tiene la misma sintaxis de C/C++/Java.

Mientras el navegador está cargando un documento web, va mostrando sus elementos (títulos, párrafos, imágenes, tablas, etc.) a medida que los encuentra. Lo mismo pasa con los elementos `<script>`. Inmediatamente que el navegador encuentra un elemento `<script>`, ejecuta el código que se encuentra en su contenido, línea a línea. El atributo `type` indica el *MIME type* del lenguaje de programación utilizado por el *script*. Si se omite, en (X)HTML5 se asumirá `"text/javascript"` y se considera una mala práctica hacer explícito este valor por defecto como se hizo en la línea 3 del

Listado 56, por lo que no se volverá a incluir en el resto del material.

En el ejemplo del Listado 56, se invoca al método `write()` del objeto `document` que representa al documento web cargado en el navegador. El método `document.write()` recibe una lista de valores separados por coma y su salida es insertada como código HTML inmediatamente después del elemento `<script>` que lo invoca, es decir, su invocación altera la estructura del documento. Una vez que el `<script>` ha terminado su ejecución, el navegador continúa procesando el resto del documento como de costumbre, y procesará tanto el código provisto por el autor en el documento original como el código insertado con `document.write()` tras ejecutar el `<script>`. Los elementos autogenerados pueden verse al inspeccionar el árbol de elementos del documento con las herramientas de desarrollador del navegador.

El uso del método `document.write()` se considera una pobre práctica de programación. De acuerdo al W3C, en XHTML el objeto `document` no dispone de este método, ya que XML prohíbe modificar el documento mientras éste se esté analizando. El efecto del método `document.write()` puede obtenerse mediante manipulación del árbol de nodos del documento (DOM, *Document Object Model*) que se estudiará en el próximo capítulo.

Para rastrear el efecto de un programa puede usarse la **consola web** {*web console*} diseñada para este fin y está disponible en el ambiente del navegador y Node.js. La consola del navegador se accede con las herramientas de desarrollador del navegador, normalmente con las teclas `Ctrl` + `Shift` + `I`, `Cmd` + `Alt` + `I`, o `F12`. El objeto `console` dispone de métodos como `console.log()` y `console.dir()` que reciben una lista de objetos separados por comas y los imprimen en notación (X)HTML o JSON (*JavaScript Object Notation*) respectivamente si son elementos, o en formato de texto si son otro tipo de objetos. El Listado 57 muestra el resultado de las potencias en la consola del navegador.

Listado 57. Reporta cuadrados naturales en la consola web (squares_list1.xhtml)

```
<body>
    <h1>Cuadrados naturales</h1>
    <p>Vea el resultado en la consola del navegador.</p>
    <script><![CDATA[
        for ( let n = 1; n <= 20; ++n )
            console.log('%d^2 = %d', n, n * n);
    ]]></script>
</body>
```

Dado que el código JavaScript en el Listado 57 está incrustado en un documento XHTML, los caracteres especiales para definir el marcado como menor que (`<`), mayor que (`>`), y ampersand (`&`) deben ser reemplazados por sus correspondientes entidades, o encerrar el código JavaScript en una sección `CDATA`. En todo caso, el documento (X)HTML debería especificar únicamente contenido, sin estilos ni comportamiento. Por tanto el código JavaScript debe estar en un recurso externo (archivo `.js`) como se verá luego.

El objeto `console` dispone de varios métodos convenientes para el reporte de diagnósticos. Por ejemplo, los métodos `console.info()`, `console.warn()`, `console.error()`, y `console.debug()` despliegan una lista de objetos separadas por comas acompañadas de iconos, colores, y otros formatos acordes a su tipo. La consola puede ser limpiada, reportar tiempos transcurridos, probar supuestos ({*asserts*}), rastrear la pila de invocaciones, entre otras funcionalidades.

Ejercicio 60 [multiplication_tables, 10 pts]

En un documento XHTML5, modifique el ejemplo del Listado 57 para desplegar en formato tabular en la consola del navegador, las tablas de multiplicar del 1 al 20. Sugerencia: indague en el objeto `console` un método apropiado para reportar este tipo de información. La tabla debe tener una fila de encabezado, una columna de encabezado y 20x20 = 400 celdas de datos, como se ilustra a continuación. Asegúrese de que su documento sea válido ante los estándares.

```
 * |  1   2   3  ..  20
---+------------------
 1 |  1   2   3  ..  20
 2 |  2   4   5  ..  40
 3 |  3   6   9  ..  60
.. | ..  ..  ..  ..  ..
20 | 20  40  60  .. 400
```

Nota: la cantidad de valores y el formato de salida depende del navegador. El siguiente código de ejemplo puede adaptarse con ciclos para crear tablas numéricas en JavaScript.

```
let matrix = [];
let row1 = [];
row1[0] = 1.1;
row1[1] = 1.2;
row1[2] = 1.3;
matrix[0] = row1;
let row2 = [];
row2[0] = 2.1;
row2[1] = 2.2;
row2[2] = 2.3;
matrix[1] = row2;
...
```

Ejercicio 61 [math_power, 5 pts]

En un documento XHTML5 escriba una tabla con las primeras 10 potencias naturales de los números 1 a 20. La tabla tendrá una fila de encabezado, una columna de encabezado y 20x10 celdas de datos. Indague sobre el método `Math.pow()`. Recuerde validar su documento.

6.1.4. La consola web del navegador

La consola del navegador se accede con las herramientas de desarrollador del navegador, normalmente con las teclas `Ctrl` + `Shift` + `I` o `F12` en Windows/Linux, y `Cmd` + `Alt` + `I` en macOS. La consola es un intérprete interactivo de JavaScript que permite escribir expresiones y obtener el

resultado de la expresión, instrucciones y ver su efecto inmediato. Esta funcionalidad es muy útil para el desarrollo en el lado del cliente {*front-end javascript*}, porque permite a la persona desarrolladora probar en tiempo real el efecto de sus instrucciones, incluso en el documento o la ventana del navegador, como se hizo en la Figura 27 que rellena una tabla XHTML usando la propiedad `innerHTML`.

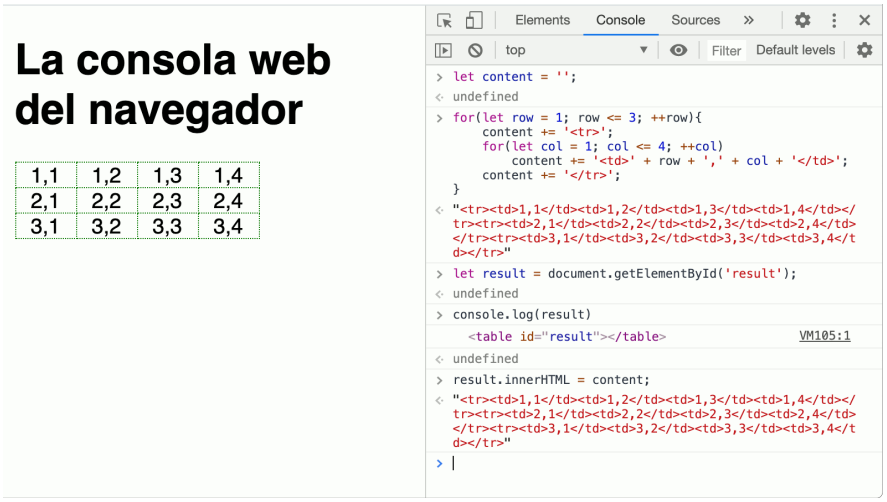


Figura 27. Una captura de pantalla de la consola web del navegador Chrome

La consola web del navegador actúa de la misma forma que una terminal del sistema operativo combinada con algunas funcionalidades de un IDE. Por ejemplo, almacena el historial de comandos que pueden ser recuperados con las teclas de flecha y autocompleta instrucciones. Si se quieren preservar las instrucciones que realizan un trabajo pretendido, deben ser trasladas a un archivo de JavaScript.

6.1.5. Código JavaScript en un archivo externo

En un documento (X)HTML, el elemento `<script>` permite escribir código JavaScript en su contenido, pero el Consorcio Web recomienda sacar el comportamiento del documento web a un recurso reutilizable, por convención, un archivo con extensión `.js`. El elemento `<script>` permite hacer la inclusión del archivo externo con el atributo `src`. El ejemplo del Listado 58 incluye en la línea 4 el archivo externo cuyo contenido se muestra en el Listado 59, y tiene el mismo efecto que el ejemplo del Listado 57. Es importante advertir que un `<script>` debe tener contenido o un recurso externo en el atributo `src`, pero no ambos, de lo contrario generará un documento inválido en (X)HTML5.

Listado 58. Importar código JavaScript en un documento (X)HTML (squares_list2.xhtml)

```
1 <body>
2     <h1>Cuadrados naturales</h1>
3     <p>Vea el resultado en la consola del navegador.</p>
4     <script src="squares_list2.js"></script>
5 </body>
```

Listado 59. El archivo JavaScript `squares_list2.js` referido en el ejemplo anterior

```
for ( let n = 1; n <= 20; ++n )
    console.log('%d^2 = %d', n, n * n);
```

Ejercicio 62 [external_file, 5 pts]

En los ejercicios sobre las tablas de multiplicar y las tablas de potencias, extraiga el código JavaScript que escribió a uno o varios archivos `.js` externos. Asocie el archivo `.js` en su documento (X)HTML. Asegúrese de que ambas tablas sean desplegadas correctamente en un lugar adecuado del documento.

En HTML la etiqueta de cierre `</script>` es siempre requerida, aunque el contenido del *script* sea vacío. Es decir, siempre se deben escribir de la forma `<script src="archivo.js"></script>`. En XHTML se puede tener una etiqueta vacía de la forma `<script src="archivo.js"/>`.

Cuando el código de un *script* se encuentra en un archivo externo, el atributo `charset` permite indicar la codificación empleada del archivo `.js` si ésta difiere de la codificación del documento (X)HTML. Los atributos `defer` y `async` permiten controlar en qué momento se debe ejecutar el guión, como se indica en la Tabla 31.

Tabla 31. Los atributos `defer` y `async` permiten controlar el momento en que el navegador ejecuta un *script*

defer	async	Comportamiento	Ejemplo
—	—	Si se omiten ambos atributos, el *script* es ejecutado inmediatamente, y antes de continuar procesando (*parsing*) el resto del documento. Dado que el código JavaScript está en un recurso externo, el navegador debe esperar a que éste se haya descargado completamente antes de continuar desplegando el documento.	`<script src="file.js"/>`
`defer`	—	El *script* es ejecutado inmediatamente después de que el documento y sus requisitos (imágenes, hojas de estilo, etc.) se hayan terminado de cargar por completo. Este es el momento idóneo para ejecutar código JavaScript en la mayoría de casos.	`<script src="file.js" defer="defer"/>`
—	`async`	Indica al navegador continuar cargando el documento y solicitar al servidor web el archivo `.js`, y cuando lo reciba, ejecutarlo simultáneamente con el cargado del documento. Sólo está disponible en HTML5 y no es soportado por Internet Explorer.	`<script src="file.js" async="async"/>`
`defer`	`async`	Carece de sentido. El navegador actuará como si sólo se hubiese especificado `async`.	—

Los atributos `defer` y `async` se pueden especificar únicamente cuando el *script* externo *no* haga uso de `document.write()`. Una invocación a este método obliga al navegador detener el cargado del

documento, esperar a que el *script* termine de generar contenido, analizarlo {*parse*} y luego continuar con el resto del documento.

6.1.6. La consola web de Node.js (REPL)

Node.js es un ambiente de ejecución que permite ejecutar código JavaScript sobre el sistema operativo en lugar del navegador. Por tanto, los programas en JavaScript pueden acceder al sistema de archivos, bases de datos, y otros recursos que no están disponibles en un navegador. El comando `node` incluido con la instalación de Node.js es un intérprete de expresiones e instrucciones de JavaScript, de la misma forma que la consola web del navegador. Se le conoce comúnmente como REPL porque implementa un ciclo de lectura de instrucciones, evaluación de las mismas, e impresión de resultados {REPL, *read-eval-print loop*}. Para ejecutar el intérprete, simplemente invoque el comando `node` sin argumentos.

Listado 60. Ejemplo de una sesión interactiva con Node.js

```
$ node
Welcome to Node.js v14.4.0.
Type ".help" for more information.

> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the repl
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file

Press ^C to abort current expression, ^D to exit the repl

> '2^64 = ' + 2**64
'2^64 = 18446744073709552000'

> console.log('2^64 = ' + 2n ** 64n)
2^64 = 18446744073709551616
undefined

> .exit
```

Al igual que la consola web del navegador, la consola web de Node.js permite probar instrucciones y luego trasladarlas a un archivo externo `.js`. El archivo puede ser ejecutado en lote como se explica en el siguiente apartado.

6.1.7. archivo `.js` ejecutado por Node.js.

Un guión de JavaScript es una secuencia de instrucciones que son interpretadas una tras otra. El comando `node` que viene en la instalación de Node.js puede interpretar en lote instrucciones en un

archivo `.js`. Simplemente se invoca el comando y se le da como argumento la ruta y nombre del archivo. El Listado 61 muestra un extracto de ejecutar el ejemplo del Listado 59.

Listado 61. Ejemplo de una sesión en lote de Node.js

```
$ node squares_list2.js
1^2 = 1
2^2 = 4
3^2 = 9
...
19^2 = 361
20^2 = 400
```

El archivo `squares_list2.js` fue asociado al documento web del Listado 58, lo que muestra una forma en que el código JavaScript puede ser reutilizado tanto al lado del cliente como del servidor. Siguiendo la recomendación del Consorcio Web, el comportamiento debería ubicarse siempre en un archivo `.js` que puede ser enlazado a un documento web, o interpretado con Node.js, o ambos. En los ejemplos de este capítulo se asumirá el ambiente de ejecución en lote de Node.js.

6.1.8. Análisis estático de código (*lint*)

Se llama en inglés ***lint*** (literalmente, "quitar pelusas") al proceso de análisis estático de código para detectar potenciales errores, violaciones a una convención de estilos, malas prácticas, código ineficiente, u otras deficiencias en el código fuente. Existen varios programas *linter* para JavaScript. Uno de los más populares al momento de escribir es ESLint {*ECMAScript Linter*}, el cual depende de Node.js. ESLint trabaja a nivel de proyecto, y requiere un archivo de configuración para saber cuáles reglas aplicar o ignorar de acuerdo a las preferencias del proyecto. El Listado 62 muestra algunos pasos para configurarlo en el directorio `$HOME` de un sistema operativo basado en Unix, de tal forma que pueda ser invocado en cualquier archivo fuente `.js`.

Listado 62. Caption (file.xhtml)

```
# Install node.js from https://nodejs.org/
# Install ESLint from https://eslint.org/

# Go to your project or home folder
cd

# Create a node.js project, answer the questions
npm init

# Create a ESLint configuration file, answer the questions
eslint --init

# Edit the ESLint configuration file to enable rules
nano .eslintrc.js

# Lint your JavaScript code
eslint /path/to/your/file.js
```

El comando `eslint --init` permite escoger el formato del archivo de configuración, entre JavaScript, JSON, y YAML. El Listado 63 muestra algunas reglas en notación JavaScript. La elección de las reglas y los niveles de aplicación dependen de las políticas de cada organización. La lista de reglas disponibles se encuentra en la documentación de ESLint.

Listado 63. Ejemplo de un archivo de configuración de ESLint (eslintrc.js)

```
module.exports = {
    'env': {
        'browser': true,
        'es2020': true,
        'node': true
    },
    // Check the default rules, see https://eslint.org/docs/rules/
    'extends': 'eslint:recommended',
    'parserOptions': {
        'ecmaVersion': 11,
        'sourceType': 'module'
    },
    'rules': {
        // Prefer single quotes
        'quotes': ['warn', 'single', { 'avoidEscape': true }],
        // All instructions must end with semicolons
        'semi': ['error', 'always'],
        // Indent with tabs
        'indent': ['warn', 'tab'],
        // Force Unix new-lines
        'linebreak-style': ['warn', 'unix'],
        // Force strict mode
        'strict': ['error', 'safe'],
        // Force curly-braces style
```

```
        //'curly': ['warn', 'multi-or-nest'],
        // Default case always must be last one
        'default-case-last': 'error',
        // Use === or !== instead of == or !=
        'eqeqeq': ['error', 'smart'],
        // Allow only one class per file
        'max-classes-per-file': 'warn',
        // No empty functions, at least a comment is needed
        'no-empty-function': 'warn',
        // Eval function is dangerous and slow
        'no-eval': 'error',
        // Avoid extend native objects
        'no-extend-native': 'error',
        // Avoid vars in loops that returns functions
        'no-loop-func': 'error',
        // Avoid multiline strings with backslash
        'no-multi-str': 'error',
        // Avoid javascript: pseudoprotocol within .js files
        'no-script-url': 'error',
        // Avoid comparing a variable with itself
        'no-self-compare': 'error',
        // Avoid loops that do not change condition
        'no-unmodified-loop-condition': 'warn',
        // Avoid unused expressions
        'no-unused-expressions': ['warn', { 'allowShortCircuit': true, 'allowTernary': true }],
        // Throw error in promises instead of reject
        'prefer-promise-reject-errors': ['error', {'allowEmptyReject': true}],
        // Do not allow var declarations, use let or const instead
        'no-var': 'error',
        // Prefer const if variable is not changed within the program
        'prefer-const': 'warn'
    }
};
```

Se considera una excelente práctica de programación revisar sus programas con un *linter*, y atender los diagnósticos que reporta. Pueda que su ambiente de desarrollo incorpore convenientemente el proceso de *lint*. Por ejemplo, el IDE JetBrains WebStorm puede configurarse para "quitar pelusas" a los programas JavaScript conforme los escribe. Este IDE es además gratuito si dispone de una cuenta académica.

Ejercicio 63 [exc_eslint, 5 pts]

Instale y configure una herramienta de *linting*. Configure las reglas como en el Listado 63. Asegúrese de que todas sus soluciones al los ejercicios previos que requieren archivos `.js` cumplen con las reglas y convenciones de estilo. Mantenga la práctica de "quitar pelusas" en todos los ejercicios de este y siguientes capítulos. Esta es una práctica valiosa a mantener en la industria y es obligatoria en empresas consolidadas de desarrollo web que definen sus propias reglas de *linting*.

6.2. Tipos de datos primitivos

JavaScript define varios tipos de datos que pueden clasificarse en primitivos y compuestos. Los tipos de datos primitivos son booleanos, números enteros y flotantes, cadenas de caracteres (*strings*), expresiones regulares, y valores especiales. Los tipos de datos compuestos son funciones, objetos, y arreglos. En esta sección se exploran los tipos de datos primitivos.

6.2.1. Números

JavaScript no hace diferencia entre números enteros y de punto flotante. Por defecto son representados internamente como punto flotante de 64 bits (IEEE 754). En expresiones donde se requieren enteros, por ejemplo para indexar un arreglo o con operadores de bits, son convertidos a complemento a dos de 32 bits. La Tabla 32 muestra algunos ejemplos de constantes literales numéricas. El guión bajo (_) dentro de una constante literal es ignorado por JavaScript, lo que permite agrupar los dígitos para hacerlos más legibles. Por ejemplo, es útil como separador de miles, o como separador de bytes en expresiones hexadecimales.

Tabla 32. Constantes literales numéricas en JavaScript

Tipo de constante literal	Ejemplos
Enteros en base 10	`0, 16777216, -3_358`
Enteros hexadecimales	`0xFF_EF, -0xff`
Enteros en octal	`0o0, 0o644, -0o012_345_670`
Enteros en binario	`0b0, 0b0001_0000, -0o11111111_11110000`
Flotantes en notación fija	`-3.141592, .1`
Flotantes en notación exponencial	`6.67e-11, .1E3`
Enteros de precisión arbitraria	`0n`, `-123_456_789_123_456_789_123_456_789n`, `0b1101n`

Cuando un valor flotante llega a no ser representable (por ejemplo, una división por cero), JavaScript no genera una excepción ni detiene la ejecución del guión, sino que se almacena con el valor especial `Infinity` o su opuesto negativo. Cuando se hace una operación indefinida, se genera un número especial `NaN` {*not-a-number*}, el cual nunca es igual a nada, incluso ni a él mismo, por eso debe usarse la función especial `isNaN()`. Otra función práctica es `isFinite()`, que prueba si un número *no* es `Infinite` y *no* es `NaN`. A continuación una lista de constantes numéricas especiales:

1. `Infinity`. Valor especial usado en lugar de un número gigante que no cabe en un `double` de 64 bits, por ejemplo el resultado de la expresión `17/0`.
2. `NaN`. Acrónimo de "not-a-number". Es un valor especial usado cuando una expresión no genera un número válido, como la división `0/0` o tratar de convertir un *string* no apto a un número como `parseInt("cinco")`.
3. `Number.MAX_VALUE`. El número más grande representable en un `double` de 64 bits.
4. `Number.MIN_VALUE`. El número decimal más pequeño (cercano a cero) representable en un `double` de 64 bits.
5. `Number.NaN`. Igual a `NaN`.

6. `Number.POSITIVE_INFINITY`. Igual a `+Infinity`.
7. `Number.NEGATIVE_INFINITY`. Igual a `-Infinity`.

Ejercicio 64 [js_constants_1, 5 pts]

Escriba un *script* que genere una tabla con las siguientes columnas: el nombre de la constante, el valor real de dicha constante, el valor que precede a la constante, el valor sucesor de la constante, el resultado de invocar `isNaN()` con la constante, y el resultado de invocar `isFinite()` con la constante. En una columna encabezado escriba cada una de las constantes listadas anteriormente. Llene las celdas de datos de la tabla con el resultado de evaluar cada expresión o función en la columna con la constante de la fila. La tabla tendrá una estructura similar a la siguiente.

Constante	Valor	Predecesor	Sucesor	isNaN()	isFinite()
Infinity	Infinity	Infinity	Infinity	false	false
NaN	...	...	...	...	...
Number.MAX_VALUE	...	...	...	...	...
...	...	...	...	...	...
Number.NEGATIVE_INFINITY	-Infinity	-Infinity	-Infinity	false	false

Ejercicio 65 [js_constants_2, 5 pts]

Agregue dos filas más a la tabla del ejercicio anterior. La primera con una expresión aritmética que genera un valor real, y la segunda que genere un valor indefinido.

6.2.2. Cadenas de caracteres (textos)

Las cadenas literales en JavaScript se encierran entre comillas dobles (`"`), `simples` (`'`), o *backticks* (`` \` ``) y se pueden anidar estos separadores dentro de la cadena. En algunos contextos (por ejemplo, *frameworks*) donde es común escribir código HTML dentro de JavaScript y viceversa, es conveniente adquirir el hábito de uniformar las comillas para cada lenguaje. El siguiente ejemplo utiliza comillas dobles para valores de atributos de (X)HTML y comillas simples para strings en JavaScript:

```
<a href="" onclick="alert('You\'re welcome')">Thanks</a>
```

Igual que en C/C++/Java, JavaScript emplea secuencias de escape iniciadas con barra invertida {*backslash*} (`\`) para anular el efecto de un carácter especial del lenguaje de programación, tales como `\n` para el cambio de línea, `\r` para el retorno de carro, `\"` para comillas dobles, `\'` para apóstrofe o comilla simple, `\0` para el carácter nulo y `\\` para la barra invertida. JavaScript almacena

internamente las cadenas de caracteres en UTF-16, lo que permite escribir caracteres especiales en el código fuente mientras éste se almacene en un archivo Unicode. Si se necesita que el código fuente esté en ASCII, se pueden usar las secuencias de escape `\xHH`, `\uHHHH`, y `\u{H···}` donde `H` se reemplaza por dígitos hexadecimales.

Las cadenas dentro de comillas dobles o apóstrofes deben terminar en la misma línea donde se abren. Sin embargo, se puede utilizar el *backslash* frente a un cambio de línea para continuar una cadena en varias líneas como ocurre con `str1` en el Listado 64. Este mismo efecto puede conseguirse con el operador de concatenación (`+`) como se hizo con `str2`.

Listado 64. Continuar un texto en varias líneas con el de escape (\) o concatenación (newline_strings.html)

```
<body><main>
    <pre id="output"/>
    <script>
        'use strict';
        let str1 = 'Cada una de estas líneas\n\
            tiene dos cambios de línea\n\
            el primero es parte del string,\n\
            el segundo es ignorado por JavaScript.\n'

        const str2 = 'Cada una de estas líneas\n' +
            'también tiene cambios de línea\n' +
            'pero están concatenadas por el operador+\n' +
            'se leen mejor y no hay que eliminar los \\t\n';

        let str3 = `Los backticks pueden ser multi-línea
            automáticamente consideran el cambio de línea
            y no requieren anular el cambio de línea final.
            Además pueden hacer interpolación de variables.`;

        str1 = str1.replace(/\t/g, '.');
        str3 = str3.replace(/\t/g, '.');
        const output = document.getElementById('output');
        console.assert(output !== null);
        output.innerHTML += str1 + '\n' + str2 + '\n' + str3;
    </script>
</main></body>
```

La línea 20 del Listado 64 emplea expresiones regulares para convertir los tabuladores en puntos con el fin de hacerlos visibles, y el resultado es asignado a la referencia misma. Por esta razón `str1` se declaró como una referencia mutable con la palabra reservada `let`. La segunda referencia `str2` no requiere ser modificada, por lo que se declaró con la palabra reservada `const`. Es una práctica heredada del paradigma de programación funcional preferir declarar como constantes todas aquellos valores nombrados que no cambian durante la ejecución del programa, y evitar el uso de variables mutables.

Las líneas 15 a 18 del Listado 64 utilizan *backticks* que pueden extenderse por varias líneas y conservan el espacio en blanco en forma literal. Además los *backticks* permiten **interpolación de variables** {*variable interpolation*} con la notación `${exp}` donde `exp` es una expresión que se evalúa

como cualquier código JavaScript, normalmente el nombre de una variable, y el resultado de la evaluación es convertido a texto. Por ejemplo, la línea 24 pudo haberse escrito con interpolación de variables de la siguiente forma:

```
output.innerHTML += `${str1}\n${str2}\n${str3}`;
```

Aunque JavaScript los caracteres de una cadena como enteros de 16 bits en Unicode, las cadenas de caracteres {*strings*} son tratadas como un tipo de datos primitivo inmutable y no un arreglo de elementos. JavaScript no tiene el concepto de carácter (`char`) como sí ocurre en otros lenguajes de programación. Un carácter se representa como un texto de longitud 1. El operador `str[i]` o el método `str.charAt(i)` permiten obtener un *string* con el carácter que está en la posición `i` de `str`, donde el índice `i` está basado en cero, es decir, 0 representa el primer carácter de la cadena. El siguiente código muestra varias operaciones definidas para cadenas de caracteres en JavaScript.

```
const visitorName = 'Chema';
const str = 'Bienvenido ' + visitorName;       // concatenación de textos (Bienvenido Chema)
const len = str.length;                        // longitud de cadena (16)
const lastChar = str.charAt(str.length - 1);   // obtener un carácter de la cadena ('a')
const sub = str.substring(6, 10);              // obtener los caracteres 6 a 10 ('nido')
const firstE = str.indexOf('e');               // la posición de la primera letra 'e' en str (2)
const lastE = str.lastIndexOf('e');            // posición de la última letra 'e' en str (13)
const lastK = str.lastIndexOf('k');            // posición de la última letra 'k' en str (-1)
const dbg = 'lastE = ' + lastE;                // concatenación con conversión, genera 'i = 13'
const upName = visitorName.toUpperCase() + '!';// convertir a mayusculas (CHEMA!)
const str2 = str.replace('e', ' ');            // Reemplaza la primera 'e' (Bi nvenido Chema)
const arr = str2.split(' ');                   // Divide el texto (['Bi', 'nvenido', 'Chema'])
const rep = upName.repeat(3);                  // Repite un texto (CHEMA!CHEMA!CHEMA!)
```

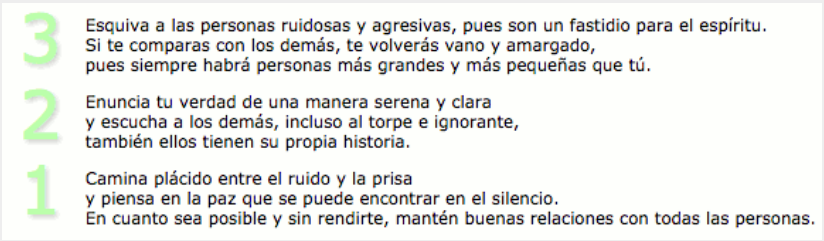
El operador de suma (+) cuando alguno de sus operandos es una cadena, hace concatenación de forma similar a Java. Debe tenerse claro que las operaciones con textos retornan un nuevo *string* como resultado, por lo que el original no se puede modificar. El método `str.split(sep)` recibe un texto separador `sep` o una expresión regular, y en cada ocurrencia de este texto, el `str` es divido. Los trozos resultantes son retornados en un arreglo de textos. Al igual que los *string*, los arreglos tienen la propiedad `length` que indica la cantidad de elementos y el operador corchetes que permiten acceder a cada elemento con la expresión `arr[i]`.

Ejercicio 66 [inverted_poem, 15 pts]

Escriba un poema de su agrado (o cualquier otro texto con varios párrafos) en una variable string de JavaScript. Separe cada verso por un cambio de línea (`'\n'`) y cada estrofa por dos cambios de línea (`'\n\n'`).

Imprima el poema dos veces. La primera vez en orden natural, reemplazando los cambios de línea por elementos `<br>` y las estrofas por párrafos (`<p>`).

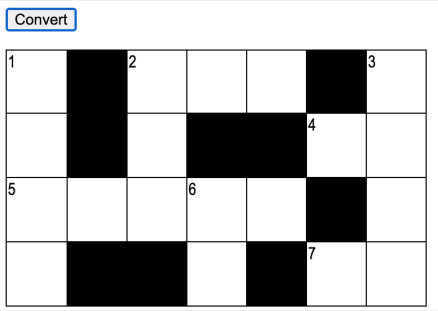
La segunda vez imprima las estrofas (no las líneas que lo componen) en orden inverso en el documento. Para hacer notorio al lector que las estrofas están en orden, o en orden inverso, imprima el número de la estrofa dentro de un elemento `<span>`. Si gusta puede escribir el texto de la estrofa dentro de otro elemento `<span>`, ambos dentro del párrafo `<p>`. Con estilos CSS haga a este número visiblemente grande y ubicado a la izquierda o derecha de la estrofa y deseablemente detrás del resto del texto (propiedad `z-index`) si hay intersección entre ambos. La siguiente imagen muestra una captura de pantalla de un ejemplo de poema invertido.



Ejercicio 67 [exc_crossword_formatter, 15 pts]

Implemente un programa que formatee crucigramas. Tome como punto de partida el documento crossword_formatter.xhtml. Extraiga el código CSS a un archivo aparte, el código JavaScript a un archivo externo, y enlace ambos al documento `crossword_formatter.xhtml`.

Modifique la hoja de estilos y el programa JavaScript para que cuando se presione el botón, genere dentro del documento una tabla que corresponde al crucigrama ingresado en el área de texto. Las celdas de la tabla deben tener únicamente números o texto vacío como contenido. Con el atributo `class`, controle las celdas rellenas y las celdas vacías. Con estilos haga que las celdas rellenas sean negras, las celdas vacías sean blancas. La siguiente figura muestra un ejemplo de un resultado.



Su solución debe ajustarse a las dimensiones del crucigrama automáticamente. Puede usar estos casos de prueba y se recomienda crear los propios. Sugerencia: Puede usar el método `str.split(sep)` que retorna un arreglo de textos resultado de separar la cadena `str` por el separador `sep`. Los arreglos tienen la propiedad `arr.length`, y puede usar el ciclo por contenedor `for(let value of arr)`. Recuerde verificar que su solución JavaScript no tenga "pelusas" con una herramienta de *linting*.

Ejercicio 68 [exc_question_field, 5 pts]

Indague sobre los métodos `padStart` y `padEnd` de los *string* de JavaScript. Use los métodos para implementar una función que recibe un ancho de página, el texto de pregunta, y el ancho del campo numérico. La función debe retornar un texto formateado con la pregunta, el campo numérico, y estos separados por puntos en el ancho dado. Una vez que haya implementado la función, el siguiente código debería correr en Node.js sin fallar los supuestos (*asserts*).

Listado 65. Formatear el texto de una pregunta con campo (question_field.js)

```
function formatQuestion(width, text, digits)
{
    return '';
}

console.assert(formatQuestion(60, 'Extension en km^2 de Costa Rica', 5) ===
'Extension en km^2 de Costa Rica.............|__|__|__|__|__|');
console.assert(formatQuestion(60, 'Covid-XX', 2) ===
'Covid-XX.....................................|__|__|');
console.assert(formatQuestion(20, 'Mes', 2) ===
'Mes..........|__|__|');
console.assert(formatQuestion(20, 'Año', 4) ===
'Año....|__|__|__|__|');
console.assert(formatQuestion(20, 'TipoSangre', 2) ===
'TipoSangre...|__|__|');
```

Sugerencia: El cuerpo de la subrutina puede implementarse en un par de líneas. Recuerde asegurarse de que su código no "tiene pelusas" con su herramienta de *linting*.

6.2.3. Conversiones de tipos

El sistema de conversión de tipos de JavaScript es muy intrincado (Flanagan, 2020), y en este material se cubrirá de forma básica. Cuando un número aparece en un contexto donde se requiere un texto, JavaScript lo convierte automáticamente. Por ejemplo, si uno de los operandos del operador de suma (`+`) es una cadena y el otro es un número, el operador `+` actuará como el operador de concatenación y convertirá el número en una cadena, como se hace en la línea 3 del Listado 66.

Listado 66. La concatenación convierte números a cadenas automáticamente (correr este ejemplo)

```
1 const hdd_gb = 500; // 500 GB
2 const hdd_gib = hdd_gb * Math.pow(10, 9) / Math.pow(2, 30);
3 const text = 'Un disco duro de ' + hdd_gb + 'GB equivale a ' + hdd_gib + 'GiB';
4 console.log(text);
```

Las conversiones explícitas se pueden hacer con el constructor `String(numero)`, y varios métodos de la clase `Number` como `numero.toString(base)`, `numero.toFixed(decimales)`, `numero.toExponential(decimales)`

y `numero.toPrecision(decimales)`. Ejemplos:

Listado 67. Conversión explícita de números a *string* (number_to_string.js)

```
const hdd_gib = 500 * Math.pow(10, 9) / Math.pow(2, 30);

hdd_gib + '';              // "465.66128730773926"
String(hdd_gib);           // "465.66128730773926"

hdd_gib.toString();        // "465.66128730773926"
hdd_gib.toString(2);       // "111010001.10101001010010010001"
hdd_gib.toString(16);      // "1d1.a94a2"

hdd_gib.toFixed(0);        // "466"
hdd_gib.toFixed(2);        // "465.66"

hdd_gib.toExponential(0);  // "5e+2"
hdd_gib.toExponential(3);  // "4.657e+2"

hdd_gib.toPrecision(4);    // "465.7"
hdd_gib.toPrecision(7);    // "465.6613"
```

Ejercicio 69 [javascript_console, 5 pts]

Pruebe todas las expresiones que aparecen en el Listado 66 y Listado 67 en la consola web de de Node.js o de su navegador, donde puede introducir sentencias que serán evaluadas "en vivo". Sugerencia cognitiva: escriba cada expresión manualmente, en lugar de copiarla y pegarla. Al final, copie los resultados que haya obtenido en la consola en un archivo de texto.

Esta consola además reporta errores gramaticales o de ejecución en el código fuente, por lo cual es importante revisarla constantemente mientras se está desarrollando en JavaScript. Mantenga el hábito de probar cada ejemplo de este capítulo en dicha consola.

En la dirección opuesta, cuando un *string* se utiliza en un contexto donde se requiere un número, será traducido automáticamente por JavaScript, por ejemplo:

```
let product = "21" * "2";  // product == 42
```

Con la notación anterior no se podrá sumar un texto a un número con el operador +, ya que será interpretado como concatenación. El constructor `Number(str)` convierte el texto `str` en un número, siempre que `str` tenga formato de número en base 10 y no inicie con espacios en blanco. Las funciones globales `parseInt(str,base)` y `parseFloat(str,base)` son más flexibles. Estas funciones suponen que `str` está en la base dada por el parámetro `base` ó 10 si se omite, y lo convierten en un número entero o real respectivamente. Las bases permitidas están entre 2 y 36. Ejemplos:

Listado 68. Conversión explícita de *string* a números

```
const height = ' 101.11cm';
const copyright = '(C)2020';

parseInt(height);                       // 101
parseInt(height,2);                     // 5
parseInt(height,16);                    // 257
parseInt('0xFF');                       // 255
parseInt('0o77');                       // 255

parseFloat(height);                     // 101.11
parseFloat(height.replace('.', 'e'));   // 10100000000000
parseFloat(height.replace('.', '_'));   // 101

parseInt(copyright);                    // NaN
parseFloat(copyright);                  // NaN
```

Si la cadena a convertir inicia con `"0x"` se interpreta que está en hexadecimal. Si inicia con 0 su resultado es indefinido, ya que algunas implementaciones interpretan octal y otras decimal, por lo que es conveniente siempre especificar la base como segundo parámetro de `parseInt()`. Si no se puede convertir a un número, estas funciones retornan `NaN`.

6.2.4. Booleanos

Los valores booleanos sólo pueden contener los valores literales `false` o `true`. Cuando se usa un booleano en un contexto numérico, se convierten automáticamente a los valores respectivos 0 y 1. Si se usan en un contexto *string*, JavaScript los convierte implícitamente a las cadenas `"false"` y `"true"`. Los valores especiales `NaN`, `null`, `undefined` y la cadena vacía (`""`) siempre se convierten a `false`; todos los demás a `true` (como `Infinity`). Para hacer explícita la conversión, es recomendable emplear la función constructora `Boolean()`:

```
let xAsBoolean = Boolean(x);
```

Ejercicio 70 [boolean_expressions, 5 pts]

¿Qué valor y tipo de datos generan las siguientes expresiones en JavaScript? Escriba sus repuestas en un documento de texto o una hoja de cálculo. Después de responder la columna "Predicción", utilice una consola de JavaScript para evaluar cada expresión y anotar en la columna "Resultado" la respuesta que obtuvo. ¿Qué porcentaje de acierto obtuvo?.

	Expresión	Predicción	Resultado
1	`const a = 0/0 == NaN;`		
2	`const b = a == false;`		
3	`const c = 0 === false;`		
4	`const d = Number(a);`		
5	`const e = Number(b);`		
6	`const f = String(a);`		
7	`const g = String(b);`		
8	`const h = Boolean('true');`		
9	`const i = Boolean(' True ');`		
10	`const j = Boolean('false');`		
11	`const k = Boolean('0');`		
12	`const l = Boolean(0);`		
13	`const m = Boolean(-1);`		
14	`const n = Boolean(null);`		
15	`const o = Boolean('');`		
16	`const p = Boolean(' ');`		

6.3. Tipos de datos compuestos

Los tipos de datos compuestos se distinguen de los primitivos porque pueden tener propiedades, y sus variables son realmente referencias hacia los valores. Los tipos de datos compuestos de JavaScript son funciones, objetos, y arreglos.

6.3.1. Funciones

El programador puede declarar sus propias funciones con la palabra reservada `function`, el nombre opcional de la función, los parámetros sin tipos entre paréntesis `( )`, y el cuerpo de la función entre llaves `{ }`. Siguiendo el paradigma de programación funcional, las funciones en JavaScript son un tipo de datos más, por ende, una función es un valor. De esta forma, las funciones se pueden almacenar en

variables, miembros de objetos, elementos de arreglos, y se pueden pasar por parámetro o retornar de una función.

Cuando una función se asigna a un objeto como un miembro, recibe el nombre especial de **método**. En caso de asignarse a una variable, puede omitirse el nombre de la función, lo que en JavaScript se llama **función literal** o **función lambda** en homenaje al lenguaje Lisp que fue uno de los primeros en permitir funciones sin nombre. El Listado 69 muestra varias formas de declarar una función en JavaScript.

Listado 69. Varias formas de declarar una función en JavaScript

```
// Una función nombrada
function square1(x) { return x * x; }

// Una función literal o lambda
const square2 = function(x) { return x * x; }

// Una función construida a partir de textos
const square3 = new Function("x", "return x * x;");

// Otra función lambda o "función flecha"
const square4 = (x) => { return x * x; }

// Invocar las funciones
square1(2.0); // 4
square2(3.0); // 9
square3(4.0); // 16
square4(5.0); // 25
```

La función constructora `Function()` recibe dos textos. El primero corresponde a los parámetros de la función. El segundo corresponde al cuerpo de la misma. El intérprete debe analizar los textos datos por parámetro y construir una función, lo cual consume recursos y se considera la variante menos eficiente para declarar una función.

La **función flecha** {*arrow function*} es una notación compacta para crear funciones sin nombre no recursivas. Se usa una flecha (⇒) que separa los parámetros de la función anónima del cuerpo de la misma. Los parámetros se escriben dentro de paréntesis redondos, pero estos paréntesis pueden omitirse si la función tiene exactamente un único parámetro. Las instrucciones que conforman el cuerpo de la función se escriben dentro de llaves ({}). Estas llaves pueden omitirse si el cuerpo consta de una única instrucción `return`. En tal caso la palabra reservada `return` debe omitirse también. A modo de ejemplo, las declaraciones del Listado 70 son equivalentes.

Listado 70. Formas de simplificar una función flecha

```
const square4 = (x) => { return x * x; }
const square5 = x => { return x * x; }
const square6 = x => x * x;

const randIn1 = (a,b) => { return a + Math.random() * (b - a); }
const randIn2 = (a,b) => a + Math.random() * (b - a);
```

Las funciones son valores compuestos. Por tanto, una variable o constante no almacena el valor sino una referencia a la función. A bajo nivel las referencias son direcciones de memoria de la computadora. Asignar una variable a otra no copia la función, sino la dirección de memoria, de tal forma que ambas variables quedan apuntando a la misma función, y por tanto, la comparten. Si se comparan dos funciones, por ejemplo, con el operador de igualdad (`==`) o identidad (`===`), no se estará comparando los parámetros o el cuerpo de las mismas, sino sus direcciones de memoria.

Para invocar una función se usa el operador paréntesis `()`, indiferentemente de si se hace en una función directamente o a través de una variable o constante. El nombre de una función sin el operador paréntesis `()` se evaluará por su dirección de memoria.

Ejercicio 71 [js_factorial, 5 pts]

Programe la función factorial recursivamente. Imprima en una lista no ordenada de (X)HTML el factorial de los primeros 20 naturales. ¿Qué sucede si llama su función con un número muy grande o con parámetro no numérico?

Ejercicio 72 [js_palindrome_arrow, 5 pts]

Almacene en una constante `isPalindrome` una función flecha que recibe un texto por parámetro y retorna un booleano que indica si el texto es palíndromo o no. La declaración de la función debería ocupar una única línea. Sugerencia: indague métodos de la pseudoclase String que le ayuden a realizar este trabajo. Su implementación debería pasar casos de prueba como los siguientes.

Listado 71. Casos de prueba para el ejercicio (palindrome_arrow.js)

```
console.assert( isPalindrome('anona') === true );
console.assert( isPalindrome('Amor a Roma') === false );
console.assert( isPalindrome('2020') === false );
console.assert( isPalindrome('20202') === true );
console.assert( isPalindrome('AMA ANA AMA') === true );
console.assert( isPalindrome('palindromo') === false );
```

Una función en JavaScript puede invocarse con igual, menos, o más argumentos que la cantidad de parámetros esperada. Si no se envía un valor para un parámetro obligatorio, recibirá el valor `undefined`. La persona programadora puede indicar **argumentos por defecto** {*default arguments*}, que

son valores que los parámetros tomarán si el invocador los omite. Al igual que C++ se usa el operador de asignación (`=`) seguido del valor por defecto al declarar los parámetros de la función. Los parámetros con argumentos por defecto deben ser los últimos de la declaración.

El Listado 72 muestra dos funciones con argumentos por defecto. La función `randBetween(a,b)` retorna un número entero pseudoaleatorio en el rango `{a, a+1,..., b-1}`. La invocación `randBetween(10)` recibe 10 en el parámetro `a`, asume `100` en el parámetro `b`, y retorna un entero de dos dígitos. La invocación `randBetween()` asume `a=0`, `b=100`, y retorna un entero entre 0 y 99 inclusive. Los argumentos por defecto también se pueden usar en funciones flecha, como ocurre con `randProbability(p)` del Listado 72, la cual recibe una probabilidad `p` entre 0 y 100%, genera un número pseudoaleatorio en este rango, y retorna verdadero si el número es menor que la probabilidad porcentual `p` dada.

Listado 72. Ejemplos de funciones con argumentos por defecto (default_arguments.js)

```
function randBetween(a = 0, b = 100)
{
    return Math.trunc(a + Math.random() * (b - a));
}

const randProbability = (percent = 50) => {
    return randBetween() <= percent; };

console.log( randProbability() ? 'escudo' : 'corona' );

if ( randProbability(10) )
    console.log('Felicidades, eres del 10% de gente con más suerte!');
```

Ejercicio 73 [js_palindrome_default, 5 pts]

Los palíndromos en el idioma tienden a ser oraciones donde se ignora la puntuación. Por ejemplo "Anita, lava la tina!" es un palíndromo. Copie su implementación de `isPalindrome` y adáptela con los siguientes cambios:

1. No sea función flecha.
2. Recibe un texto por parámetro obligatorio, si no debe hace fallar un supuesto (`console.assert`).
3. Recibe un segundo parámetro que indica si debe ignorar mayúsculas y minúsculas. Si no se provee, asume `false`.
4. Recibe un tercer parámetro que indica si debe ignorar puntuación. Si no se provee asume `false`.

Asegúrese de que nueva implementación pase los casos de prueba del ejercicio `js_palindrome_arrow`. Agregue los siguientes casos de prueba. Implemente su solución para ignorar cualquier puntuación y no sólo la incluida en los casos de prueba.

Listado 73. Casos de prueba para el ejercicio (palindrome_default.js)

```
console.assert( isPalindrome('A-no-na', true) === false );
console.assert( isPalindrome('A-no-na', true, true) === true );
console.assert( isPalindrome('Amor a Roma', true) === true );
console.assert( isPalindrome('Amor a Roma', true, true) === true );
console.assert( isPalindrome('2020', true) === false );
console.assert( isPalindrome('2020', true, true) === false );
console.assert( isPalindrome('pa-lin dro-mo', true, true) === false );
console.assert( isPalindrome('amar da drama', false, false) === false );
console.assert( isPalindrome('amar da drama', false, true) === true );
console.assert( isPalindrome('Amar da drama', false, true) === false );
console.assert( isPalindrome('Amar da drama', true, true) === true );
console.assert( isPalindrome('Anita, lava la tina!') === false );
console.assert( isPalindrome('Anita, lava la tina!', true) === false );
console.assert( isPalindrome('Anita, lava la tina!', true, true) === true );
```

Sugerencia: Indague métodos de la pseudoclase String que le ayuden a realizar este trabajo en pocas líneas. En especial use expresiones regulares para buscar y eliminar la puntuación del texto.

En JavaScript las funciones son tipos de datos compuestos porque en realidad son objetos, y por ende tienen propiedades. Una función puede ser invocada con igual, menos o más argumentos de los esperados por la función, lo cual no genera un error de sintaxis. Por el contrario, JavaScript permite al programador determinar la cantidad real de parámetros enviados, y por tanto, es fácil implementar funciones que trabajan adecuadamente con cantidades arbitrarias de argumentos. El Listado 74 es un ejemplo de invocación de una función tradicional que podría beneficiarse de esta característica.

Listado 74. Cómo determinar la cantidad de parámetros con que se invoca una función

```
// Sea una función que espera dos parámetros, ignora los demás
function min(a,b) { return a < b ? a : b; }

// En JavaScript es completamente valido invocarla con menos o más parámetros
min(25, 0.5, -80);   // retorna 0.5 en lugar de -80
```

Como se ve en el Listado 74, la invocación de `min()` con tres parámetros falla el resultado esperado por el llamador. Si una función tiene sentido que trabaje con una cantidad arbitraria de argumentos, se puede hacer que reciba un arreglo de argumentos. Este arreglo se declara como el último parámetro de la lista y se le antecede con tres puntos en su nombre. Cuando la función se invoca, los argumentos son mapeados a los parámetros normales en el mismo orden como de costumbre. Cuando todos los parámetros normales han recibido su valor, el arreglo tomará los argumentos restantes. El Listado 75 muestra un ejemplo.

Listado 75. Una función que recibe un número arbitrario de argumentos (varargs_min.js)

```
function min(value1, ...rest)
{
    let result = value1;
    for (let index = 0; index < rest.length; ++index)
        if ( rest[index] < result )
            result = rest[index];

    return result;
}

min();              // undefined
min(25);            // 25
min(25, 0.5);       // 0.5
min(25, 0.5, -80);  // -80
```

Las funciones que pueden recibir una cantidad arbitraria de argumentos como `min` en Listado 75 se les suele llamar en inglés *variadic functions* o *vararg functions* por herencia de C. Antes de ECMAScript6, los programadores accedían a los argumentos con el *pseudoarreglo* {*array-like object*} `arguments` que toda invocación a función recibe. La longitud de ese arreglo, dada por `arguments.length`, indica la cantidad de argumentos con que se invocó la función y la expresión `arguments[i]` accede al argumento `i + 1` en la lista. Este pseudoarreglo podría usarse para forzar una cantidad exacta de argumentos como el Listado 76, que compara la cantidad de parámetros declarada en la función (`min.length`) con los argumentos con que se invocó (`arguments.length`). Sin embargo, a partir de ECMAScript6 no se recomienda el uso del pseudoarrglo `arguments`.

Listado 76. Una función que exige un número estricto de argumentos.

```
// Esta versión de min sólo trabaja con dos argumentos
function min(a,b)
{
   // Si el numero dado de argumentos es diferente de los declarados
   if ( arguments.length != min.length )
      throw new Error(`min(): expected ${min.length} args, sent ${arguments.length}`);

   // Solo dos argumentos fueron provistos como se esperaba
   return a < b ? a : b;
}

try
{
   min();              // Lanza excepción
   min(25);            // Lanza excepción
   min(25, 0.5);       // Retorna 0.5
   min(25, 0.5, -80);  // Lanza excepción
}
catch(exc)
{
   console.error(exc);
}
```

Ejercicio 74 [average_vararg, 5 pts]

Escriba una función que puede recibir una cantidad arbitraria de parámetros y calcula su promedio. Si es invocada sin parámetros retorna `undefined`. Llame su función con cero, uno, dos y más argumentos e imprima el resultado en la consola. Compruebe que los resultados sean correctos.

Ya que una función es un objeto, puede tener propiedades, como `length` usada en el Listado 76. El programador también puede crear sus propias propiedades utilizando el operador punto (`.`). Las propiedades que el programador defina tendrán el efecto de ser "variables estáticas" de la función. El Listado 77 muestra una función que reporta el número de veces que ha sido invocada.

Listado 77. Simular una variable estática dentro de una función (call_count.js)

```
// Crea una "variable estática" como propiedad de la función. JavaScript analiza el código
// antes de ejecutarlo, por lo que el nombre de la función puede usarse antes de ser declarada
countCalls.count = 0;

function countCalls()
{
   // Usar la propiedad como si fuese una variable estática
   console.log(`countCalls() ha sido llamada ${++countCalls.count} veces`);
}

countCalls();  // Imprime "countCalls() ha sido llamada 1 veces"
countCalls();  // Imprime "countCalls() ha sido llamada 2 veces"
countCalls();  // Imprime "countCalls() ha sido llamada 3 veces"
```

6.3.2. Arreglos

En JavaScript un arreglo es una colección de datos enumerados. Los arreglos pueden tener datos heterogéneos, de cualquiera de los tipos de datos de JavaScript. Así un elemento del arreglo puede contener otro arreglo, y el elemento que le sigue puede contener una función. Los arreglos se pueden crear con el constructor `Array()`o con el arreglo literal `[]`. Los elementos se pueden agregar simplemente asignándolos a sus índices o bien, como parámetros del constructor `Array()`. Sin embargo, si se pasa un único entero al constructor, de la forma `Array(N)`, se creará un arreglo con `N` elementos indefinidos. Los elementos se acceden con un índice entero basado en 0, escrito entre corchetes tras el nombre del arreglo. La propiedad `length` permite acceder al número de elementos en el arreglo. El [lst_array_create_access] muestra la creación de arreglos y acceso a sus elementos.

```
let a = new Array();                // Arreglo vacío, lo mismo que: let a = [];
a[0] = 1.2;                         // Un elemento es insertado en la posición 0
a[1] = "JavaScript";                // Un elemento es insertado en la posición 1
a[2] = true;
a[4] = { x:1, y:3 };                // La posición 3 tiene un elemento con el valor undefined
a[5] = (x) => { return x * x; };    // a[5](7) retornará 49
a[6] = [];                          // Elemento a[6] almacena un arreglo vacío
a[6][0] = -Infinity;                // Inserta un elemento en el arreglo que está en a[6]
console.log(a.length);              // Imprime la cantidad de elementos en el arreglo: 7
delete a[6];                        // Elimina el último, pero a.length se mantiene en 7

let b = new Array(1.2, "ES6", true, {x:1, y:3});
console.log(b.length);              // 4 elementos

let c = new Array(10);              // Arrreglo de 10 elementos indefinidos
console.log(c.length);              // Imprime 10 y no 1
```

JavaScript permite crear arreglos literales, preferiblemente utilizados para inicialización, y son una lista de valores separados por comas dentro de corchetes, que se asignan secuencialmente empezando

en 0. Los elementos también pueden ser indefinidos lo cual se logra omitiendo el valor entre comas:

```
let b = [ 1.2, "JavaScript", true, { x:1, y:3 } ];

let matrix = [[1,2,3], [4,5,6], [7,8,9]];           // matrix[2][1] == 8

let base = 1024;
let table = [base, base+1, base+2, base+3];

let sparseArray = [1,,,,5];
```

El Listado 78 muestra varios métodos para recorrer los elementos de un arreglo. El **ciclo por contador** {*counter loop*} utiliza una variable entera que en cada iteración toma un índice del arreglo, como ocurre en las líneas 5 y 6 del Listado 78. El **ciclo por contenedor** {*container loop*}, también conocido como *for-each loop*, recorre un elemento a la vez del contenedor. En JavaScript el ciclo por contenedor es implementado por la cláusula `for/of` como ocurre en las líneas 10 y 11 del Listado 78. En cada iteración, la variable o constante a la izquierda del operador `of` toma un valor del contenedor. La decisión entre declarar una variable (`let`) o constante (`const`) depende de si el valor se va a modificar dentro del ciclo o no.

Listado 78. Recorrido por un arreglo (array_traverse.js)

```
const values = [-5, 6, 4, 5, -7];

// Counter loop
let sum1 = 0;
for ( let index = 0; index < values.length; ++index )
    sum1 += values[index];

// Container loop
let sum2 = 0;
for ( const value of values )
    sum2 += value;

// Non-destructive recursion
function sum(array)
{
    if ( array.length === 0 )
        return 0;
    const [first, ...last] = array;
    return first + sum(last);
}

const sum3 = sum(values);

console.log('values =', values);
console.log('sum1 =', sum1);
console.log('sum2 =', sum2);
console.log('sum3 =', sum3);
```

Los arreglos también pueden recorrerse por recursión. Las invocaciones de la función recursiva podrían recibir el índice por parámetro. Sin embargo, las líneas 14 a 20 del Listado 78 usan "destructuración" del arreglo. En cada invocación la línea 18 crea dos constantes: `first` que obtiene el primer valor del arreglo, y `last` que obtiene un arreglo con los restantes valores. Por ejemplo, en la primera inovación `first` obtendrá el valor `-5` y `last` el arreglo `[6,4,5,-7]`. En la segunda invocación `first` obtendrá el valor `4` y `last` el arreglo `[4,5,-7]`. En la penúltima invocación `first` obtendrá el valor `-7` y `last` el arreglo vacío `[]`, que disparará la condición de parada en la líneas 16 y 17.

La **destructuración** {*destructuring*} es una característica de ECMAScript 6 que permite declarar o modificar variables a la izquierda de una asignación, con un objeto o arreglo al lado derecho. Las variables o constantes del lado izquierdo obtendrán los valores de los campos o elementos del lado derecho en el mismo orden y estructura. El Listado 79 muestra como aprovechar la "destructuración" para intercambiar dos variables en la línea 4. Simplemente las variables `a` y `b` obtienen los valores del arreglo de la derecha en el mismo orden, por tanto, `a` adquirirá el valor de `b` y `b` el valor de `a`. Este intercambio parece propenso a fallar, dado que una vez que `a` tome el valor de `b` afectará la asignación siguiente de `b`. Sin embargo, este fallo no ocurre, dado que a expresión de la derecha se evalúa primero, y en este caso es la creación de un nuevo arreglo temporal anónimo `[b, a]` que copia los valores de `a` y `b` antes de realizar la asignación.

Listado 79. Intercambio de variables por destructuración (destructuring_swap.js)

```
1  function randBetween(a = 0, b = 100)
2  {
3      if ( a > b )
4          [a, b] = [b, a]; // swap
5
6      return Math.trunc(a + Math.random() * (b - a));
7  }
8
9  // Dice roll: Both calls print an integer between [1, 7[
10 console.log( randBetween(1, 7) );
11 console.log( randBetween(7, 1) );
```

Ejercicio 75 [exc_crossword_autonum, 15 pts]

Copie su solución al ejercicio Ejercicio 67 [exc_crossword_formatter, 15 pts]. Para el usuario sería de mucha ayuda que la computadora sea quien calcule los números donde inician las palabras en el crucigrama. Por tanto, la entrada sólo constaría de los símbolos numeral (`#`) para las celdas rellenas, y punto (`.`) para las celdas vacías. Puede suponer que las celdas se separan por espacios en blanco, como ocurre en estos casos de prueba y se recomienda crear los propios.

Un crucigrama se puede numerar por filas o por columnas. Incluya un control a la izquierda al botón "Generar" que le permita al usuario escoger si el crucigrama se va a auto-numerar por filas o por columnas. El control puede ser una casilla de verificación, o dos botones de radio).

La numeración por filas o por columnas sólo indica que el orden en que el recorrido se realiza por el crucigrama. Por filas es de izquierda a derecha en cada fila iniciando con la superior, como se ve la izquierda de la Figura 28. Por columnas es de arriba a abajo en cada columna iniciando con la del extremo izquierdo, como se ve a la derecha de la Figura 28. En cada celda se debe determinar si en ella inicia una palabra de al menos dos letras, sea en forma horizontal o vertical o ambas. Una palabra siempre inicia después de una celda rellena o un borde hacia la derecha o hacia abajo. Si en la celda inicia una palabra, se escribe un número, de lo contrario se deja vacía.

Por filas

1	■	2	3		■	4
5	6	■		■	7	
8		9		10		■
■	11		■	12		13
14		■	15	■	16	
17		18		19	■	
■	20					■

Por columnas

1	■	9	12		■	19
2	6	■		■	17	
3		10		14		■
■	7		■	15		20
4		■	13	■	18	
5		11		16	■	
■	8					■

Figura 28. Ejemplo de numeración por filas y por columnas de un crucigrama

Sugerencia: construya a partir de los textos ingresados una matriz (un arreglo de arreglos) que le permita determinar las celdas que llevan números.

Al igual que en Java, los arreglos en JavaScript son objetos, y por tanto, tipos de datos compuestos. La propiedad más natural de un arreglo es `length` que indica la cantidad de elementos que hay almacenados en él. De hecho, esta propiedad es la principal diferencia entre un arreglo y un objeto tradicional. La Tabla 33 muestra una lista de métodos que cada arreglo hereda de la pseudoclase `Array`. Es importante resaltar que algunos de estos métodos modifican el arreglo directamente.

Tabla 33. Métodos de la pseudoclase `Array`

Método	Detalles
`push(elems)`	Inserta los elementos enviados por parámetro al final del arreglo. Retorna la cantidad de elementos que quedan en el arreglo. `[1,2,3].push(9,null) // retorna 5, deja [1,2,3,9,null]`

Método	Detalles
`pop()`	Elimina y retorna el último elemento del arreglo. Si está vacío, retorna `undefined`. `[1,2,3].pop() // deja [1,2]`
`unshift(elems)`	Inserta los elementos enviados por parámetro al inicio del arreglo. Retorna la cantidad de elementos que quedan en el arreglo. `[1,2,3].unshift(9,null) // retorna 5, deja [9,null,1,2,3]`
`shift()`	Elimina y retorna el primer elemento del arreglo `[1,2,3].shift() // deja [2,3]`

`join(sep)`	Retorna un string resultado de concatenar todos los elementos en el arreglo, separados por la cadena `sep`, o comas si se omite.
	`[1,2,3].join()     // genera "1,2,3"`
	`[1,2,3].join('; ') // genera "1; 2; 3"`

`toString()`	Está sobrescrito para hacer lo mismo que `join()`. `[1,[8,9],2].toString() // retorna "1,8,9,2"`
`reverse()`	Invierte el orden de los elementos en el arreglo. `[1,2,3].reverse() // genera [3,2,1]`

Método	Detalles

`sort(comp)`	Ordena los elementos en el arreglo de acuerdo a la función comparadora `comp()`, sino los ordena alfabéticamente invocando `toString()` a cada uno de sus elementos.
	`[7, 200, 81].sort() // genera [200,7,81]`
	`[7, 200, 81].sort( (a,b) => a - b ) // genera [7,81,200]`

`slice(i,j)`	Retorna un nuevo subarreglo con los elementos encontrados desde la posición `i` hasta `j - 1`, es decir, el que está en `j` no se incluye. `[1,2,3,4,5].slice(1,3) // genera [2,3]`

`forEach(f)`	El método `forEach` invoca la función `f` con cada elemento del arreglo. La función será invocada con tres argumentos: el elemento, su índice, y una referencia al arreglo. Los dos últimos permiten modificar el elemento, pero si por el contrario no se usan se pueden omitir en la definición de la función. Es común utilizar funciones flecha por su notación compacta. Si el arreglo es disperso, la función no se invoca en los valores no inicializados.
	`let arr = [3,-8,,5], min = Infinity;`
	`arr.forEach( (x) => { min = x < min ? x : min; } ) // min == -8`

`map(f)`	El método `map` invoca la función `f` con cada elemento del arreglo, la cual debe producir un valor de retorno. Este valor será almacenado en un nuevo arreglo de elementos resultantes que será el valor de retorno de `map`. Por lo tanto, el trabajo de `map()` es producir un arreglo resultado de "mapear" los elementos del arreglo original (dominio) a un arreglo resultante (codominio) invocando una función `f`. La función `f` será invocada con tres argumentos: el elemento, su índice, y una referencia al arreglo. Si no se usan, los dos últimos se pueden omitir en la definición de la función. Es común utilizar funciones flecha por su notación compacta. Si el arreglo es disperso, la función no se invoca en los valores no inicializados, sino que estos serán copiados al arreglo resultante.
	`let names = ['ana', 'jAIME', , 'IRIS']; // We want: ['Ana', 'Jaime', , 'Iris']`
	`names.map( str => str.slice(0,1).toUpperCase() + str.slice(1).toLowerCase() );`

Conviene estudiar la pseudoclase Array dado que versiones recientes de JavaScript han agregado una considerable cantidad de métodos útiles.

Ejercicio 76 [exc_statistics, 10 pts]

Escriba un programa para Node.js que lee números de la entrada estándar, uno por línea, e imprime estadísticas: la cantidad de valores, el valor mínimo, el valor máximo, el promedio, la desviación estándar, y la mediana. Puede adaptar el código fuente del Listado 80. Se necesita que el programa produzca una salida como la siguiente. Su solución *NO* debe agregar ciclos ni funciones recursivas al código fuente dado. Indague sobre métodos de la pseudoclase Array, en especial sobre el método `reduce` que le ayude a hacer este trabajo siguiendo el paradigma de programación funcional.

Entrada de ejemplo:

```
75
90
100
45
```

Salida de ejemplo:

```
count  : 4
minimum: 45
maximum: 100
average: 77.5
stdev  : 23.979157616563597
median : 82.5
```

Listado 80. Código inicial para el ejercicio: lee líneas de la entrada estándar (statistics.js)

```
process.stdin.setEncoding('utf8');

let values = [];

process.stdin.on('readable', () =>
{
    let line = null;
    while ((line = process.stdin.read()) !== null) {
        process.stdout.write(`[${line}]`);
    }
});

process.stdin.on('end', () =>
{
    process.stdout.write(`count: ${values.length}\n`);
});
```

Para invocar su programa puede redireccionar la entrada estándar al archivo que contiene los datos, por ejemplo `node statistics.js < statistics01.txt`. Nota: los estadísticos no están definidos para conjuntos vacíos. Si se invoca sin datos, por ejemplo `node statistics.js < /dev/null`, debería

indicarlo en la salida, como:

```
count  : 0
minimum: undefined
maximum: undefined
average: undefined
stdev  : undefined
median : undefined
```

6.4. Objetos

Un **objeto JavaScript** {*JavaScript object*} es una colección de valores nombrados, que usualmente se les refiere como propiedades, campos o miembros del objeto, y se les accede utilizando el operador punto. Por ejemplo:

```
image.width
image.height
document.myform.button
console.log('log es un método: una propiedad cuyo tipo de datos es una función');
```

El operador punto permite acceder a las propiedades utilizando identificadores. Pero JavaScript permite también usar cadenas de caracteres para acceder a las propiedades, con el operador corchetes `[]`. Esta segunda notación es mucho más flexible ya las cadenas de caracteres pueden ser el resultado de evaluar una expresión. De esta forma, el operador corchetes permite emplear los *objetos* de JavaScript como *arreglos asociativos*, también llamados *mapas maps* o *tablas de dispersión hash*. Ejemplos:

```
image['width']
image['height']
document['myform']['button']
console['log']('log es un método: una propiedad cuyo tipo de datos es una función');
```

Los objetos se crean llamando funciones constructoras con el operador `new`, después de lo cual se usan como de costumbre. Estos objetos son almacenados en memoria dinámica por el navegador, el cual incorpora un recolector de basura {*garbage collector*}, de tal forma que ahorra al programador la responsabilidad de liberar la memoria de cada objeto creado.

Listado 81. Los objetos se crean con el operador `new` y funciones constructoras

```
1 let now = new Date();
2 let pattern = new RegExp("\\s(\\w+)\\s", "i");
3
4 let point = new Object();    // Lo mismo que: let point = {};
5 point.x = 2.3;
6 point.y = -1.2;
```

En la línea 4 del Listado 81 el objeto `point` se creó como un objeto vacío, y sus propiedades se fueron agregando luego con el operador de asignación (líneas 5 y 6). Existe una notación para definir objetos literales, con o sin propiedades, útil para inicializaciones:

Listado 82. Notación para escribir objetos literales en JavaScript

```
let obj =
{
    property1 : value1,
   "property2": value2,
    ...,
   'propertyN': valueN
};
```

Las llaves `{}` en una sección de declaración de variables (`let`) de JavaScript indican la creación de un **objeto literal** {*literal object*}. Los nombres de las propiedades pueden declararse como identificadores (como se hizo con `property1` en Listado 82) o como *strings* (como se hizo con `"property2"` en Listado 82). Los valores de cada propiedad pueden ser de cualquier tipo de datos de JavaScript (booleano, numérico, texto, función, arreglo, u objeto), sea como valores literales o como resultado de una expresión aritmética. Cada propiedad debe separarse por una coma, y no por punto y coma. Las declaraciones de objetos literales se pueden anidar como se hizo con `rectangle` en el Listado 83.

Listado 83. Notación de objetos de JavaScript

```
let point = { x:2.3, y:-1.2 };

let activo1 =
{
   "tipo": "disco_duro_externo",
   "precio": 30000, // colones
   'tamaño': 500 * Math.pow(10, 9) / Math.pow(2, 30), // GiB
   'interfaces': ['eSATA', 'USB3']
};

let rectangle =
{
   color : "#a4f0ca",

   background:
   {
      color: "lightgray",
      image: "img/bricks.png",
      repeat: "repeat"
   },

   geometry:
   {
      topLeft: { x: 45, y: 10 },
      extend: { width: 21.93, height: 38.34 }
   }
};
```

La notación presentada en el Listado 83 para escribir objetos literales, es importante, y recibe el nombre de **JavaScript Object Notation** (JSON). Es ampliamente usada para representar datos y funciones que deben ser intercambiados entre el servidor web y el navegador, en bases de datos orientadas a documentos, y un número creciente de tecnologías. Si se escribe en un archivo con extensión `.json`, todos los nombres de los miembros deben estar dentro de comillas dobles, y no se permiten comentarios. Si el objeto JSON está en un archivo de JavaScript (`.js`) estas restricciones no aplican.

Cada vez que JavaScript evalúa un objeto literal, se crea un objeto diferente, debido a que los valores de las propiedades pueden ser expresiones que cambian de una evaluación a otra, por ejemplo, en un ciclo. El Listado 84 muestra un programa JavaScript que construye una tabla (X)HTML con una lista de personas ficticias generadas pseudo-aleatoriamente a partir de un arreglo de nombres y apellidos.

El programa del Listado 84 consta de dos declaraciones de variables globales, cuatro declaraciones de funciones y una invocación de función (línea 37). Al ser invocada, la función `imprimirPersonas()` recibe el número deseado de personas y por cada una de ellas invoca a `crearPersona()`, la cual crea un objeto literal (líneas 13 a 17), lo almacena en una variable local, y luego retorna una referencia a dicho objeto. La función `imprimirPersonas()` recibe el objeto creado (línea 33) y lo pasa por parámetro a `imprimirPersona()` quien genera la línea correspondiente en la tabla. Es importante notar que en cada

iteración del ciclo la evaluación del objeto literal (líneas 13 a 17) genera una persona distinta, y no el mismo objeto, incluso aunque el valor de sus propiedades no cambiaran.

Listado 84. Cada vez que se evalúa un objeto literal, JavaScript crea un nuevo objeto. Correr ejemplo completo

```
1  const nombres = ['Ana', 'Braulio', 'Carlos', 'Dalia', 'Ernesto', 'Hilda', 'Laura'];
2  const apellidos = ['Abarca', 'Castillo', 'Fernández', 'Mora', 'Pérez', 'Trejos', 'Vega'];
3
4  // Escoge un valor del arreglo aleatoriamente y lo retorna
5  function valorAleatorio(arreglo)
6  {
7     return arreglo[ Math.trunc( Math.random() * arreglo.length ) ];
8  }
9
10 // Crea y retorna un objeto persona diferente cada vez que se invoca
11 function crearPersona()
12 {
13    const nuevaPersona = {
14       nombre: valorAleatorio(nombres) + ' ' + valorAleatorio(apellidos),
15       edad: Math.round(Math.random() * 110),
16       ingreso: Math.floor(Math.random() * 1000) + ',000'
17    };
18    return nuevaPersona;
19 }
20
21 // Imprime un objeto persona en un registro de una tabla (X)HTML
22 function imprimirPersona(persona, numero)
23 {
24    console.log(`${numero}\t${persona.nombre}\t${persona.edad}\t${persona.ingreso}`);
25 }
26
27 // Crea e imprime en una tabla (X)HTML la cantidad solicitada de personas aleatorias
28 function imprimirPersonas(cantidad)
29 {
30    console.log('#\tNOMBRE APELLIDOS\tEDAD\tINGRESO');
31    for ( let index = 1; index <= cantidad; ++index )
32       imprimirPersona( crearPersona(), index );
33 }
34
35 imprimirPersonas(300);
```

Si un objeto se emplea en un contexto booleano se traduce siempre a `true`, a menos de que sea una referencia `null`. Si un objeto aparece en un contexto *string* se invocará su método `toString()` para hacer la conversión. Si un objeto aparece en un contexto numérico, JavaScript invocará `valueOf()` para hacer la conversión. El desarrollador puede sobrescribir estos métodos en sus propios objetos, lo cual es una práctica recomendada de programación.

Ejercicio 77 [geometric_figures_1, 10 pts]

Escriba un objeto `Círculo`, `Triángulo` y `Rectángulo`. En cada uno de ellos almacene su posición y dimensiones. Escoja valores arbitrarios para cada uno de ellos. Provea dos métodos en cada objeto, uno para calcular el perímetro y otro para el área. Haga un programa en JavaScript que imprima para cada una de las tres figuras: su tipo, posición, dimensiones, perímetro y área. Para este ejercicio no tiene que dibujar las figuras geométricas, sino imprimir sus coordenadas y resultados numéricos en una tabla como la de abajo.

#	Figura	Posición/Dimensiones	Perímetro	Área
1	Círculo	centro(..., ...) radio(...)	...	...
2	Triángulo	a(..., ...) b(..., ...) c(..., ...)	...	...
3	Rectángulo	a(..., ...) b(..., ...)	...	...

Nota: si `A`, `B` y `C` son los vértices de un triángulo, su área y perímetro se pueden obtener mediante:

$$area = \left| \frac{A_x\left(B_y - C_y\right) + B_x\left(C_y - A_y\right) + C_x\left(A_y - B_y\right)}{2} \right|$$

$$perimeter = \sqrt{\left(A_x - B_x\right)^2 + \left(A_y - B_y\right)^2} + \sqrt{\left(A_x - C_x\right)^2 + \left(A_y - C_y\right)^2} + \sqrt{\left(B_x - C_x\right)^2 + \left(B_y - C_y\right)^2}$$

Ejercicio 77 [geometric_figures_2, 5 pts]

Agregue otro triángulo a la colección de figuras hechas en el ejercicio anterior. Intente reutilizar el código de los métodos del primer triángulo en lugar de duplicarlo (consulte la Sección 6.3.1). Haga que su nuevo triángulo aparezca en la cuarta línea de la tabla. Sugerencia: puede verificar el área de sus triángulos contra esta aplicación.

6.4.1. Objetos tipo arreglo

El operador corchetes `[]` se utiliza en JavaScript tanto para acceder a los elementos de un arreglo, como a las propiedades de un objeto. De esta forma, un objeto podría utilizarse para simular un arreglo, almacenando los elementos en propiedades con nombres numéricos (`'1'`, `'2'`, `'3'`, ...), más una propiedad `length` asignada de manera acorde a la cantidad de elementos alojados en el objeto. Por ejemplo:

Listado 85. Objetos tipo arreglo. Correr este ejemplo

```
1 // Retorna un objeto-arreglo de nombres aleatorios
2 function generarNombres()
3 {
4    let nombres = ['Ana', 'Beto', 'Denis', 'Fabiola', 'Horacio', 'Ingrid', 'Marta'];
5    let resultado = {};
6    let total = Math.floor(Math.random() * 100);
7
8    for ( let i = 0; i < total; ++i )
9       resultado[i] = nombres[ Math.floor(Math.random() * nombres.length) ];
10
11   resultado.length = total;
12   return resultado;
13 }
14
15 // Generar un arreglo de nombres aleatorios e imprimirlo
16 let nombresAleatorios = generarNombres();
17 for ( let i = 0; i < nombresAleatorios.length; ++i )
18    console.log('\n', i + 1, ': ', nombresAleatorios[i]);
19
20 console.log('\n\nTotal ', nombresAleatorios.length, ' nombres generados.');
```

Cada vez que la función `generarNombres()` del Listado 85 es invocada, creará un objeto vacío `resultado` (línea 5), el cual llenará un número pseudo-aleatorio de propiedades cuyo nombre son precisamente números secuenciales. En la línea 11 le crea una propiedad `length` y retorna una referencia al objeto. Las líneas 16 a 20 crean uno de estos objetos y lo utilizan como si fuese un arreglo normal. Sin embargo no lo es, hay algunas diferencias importantes, en especial que carece de los métodos heredados de la pseudoclase `Array`, y por ende el **objeto tipo arreglo** {*array-like object*} tiene poca funcionalidad para ser modificado. Es decir, se ha creado un arreglo de "sólo lectura", lo cual es una cualidad importante en JavaScript y ampliamente explotada por los navegadores para acceder al modelo de objetos del documento {DOM, *Document Object Model*} como se verá adelante.

Ejercicio 78 [geometric_figures_3, 5 pts]

Almacene las figuras que haya creado en los ejercicios Ejercicio 77 [geometric_figures_1, 10 pts] y Ejercicio 77 [geometric_figures_2, 5 pts] en un arreglo. Con un ciclo recorra el arreglo imprimiendo cada figura, su perímetro y su área, para generar la tabla resultado.

Ejercicio 79 [geometric_figures_4, 30 pts]

Escriba una función que crea y retorna un objeto figura geométrica pseudo-aleatoriamente cada vez que se invoca. Llene un arreglo de 20 figuras aleatorias e imprímalas en la tabla como hizo en el ejercicio anterior. Puede apoyarse en los métodos `Math.random()` y `Math.trunc()` para generar números aleatorios.

Ejercicio 80 [inventory_json, 40 pts]

Represente el inventario de la Figura 23 como una jerarquía de activos en *JavaScript Object Notation* (JSON). Programe métodos para imprimir cada activo e imprima el inventario en un documento HTML. Haga que cada activo genere un `<div>` en el documento, de tal forma que los `<div>` mantengan el mismo anidamiento que los objetos de JavaScript. Con estilos CSS agregue bordes y colores a los `<div>` para que sea visualmente clara la relación entre estos. La Figura 29 muestra un ejemplo potencial.

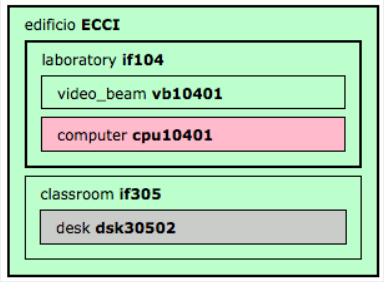


Figura 29. Potencial representación del inventario

6.4.2. La pseudoclase `Object`

Todos los objetos de JavaScript heredan de la pseudoclase `Object`, y por ende, todos los objetos de JavaScript tienen las propiedades que esta pseudoclase defina. En lo siguiente se presentarán algunas de ellas. Cuando se crea un objeto siempre se guarda una referencia a la función que se utilizó para inicializar el objeto, en la propiedad `constructor`. Por ejemplo:

```
let d1 = new Date();
d1.constructor == Date    // Se evalúa como true
```

La propiedad `constructor` sirve para saber si un objeto es instancia directa de una psudoclase particular, mientras que el operador `instanceof` sirve para determinar si el objeto es descendiente de una pseudoclase dada. Por ejemplo:

```
let d1 = new Date();
d1.constructor == Date    // true
d1.constructor == Object  // false
d1 instanceof Date        // true
d1 instanceof Object      // true, todo objeto es descendiente de Object
d1 instanceof Array       // false

let o1 = new Object();
let o2 = {};
o1.constructor == Object  // true
o2.constructor == Object  // true
o1 instanceof Object      // true
o2 instanceof Object      // true

let a1 = new Array();
let a2 = [];
a1.constructor == Array   // true
a2.constructor == Array   // true
a1.constructor == Object  // false
a2.constructor == Object  // false
a1 instanceof Array       // true
a2 instanceof Array       // true
a1 instanceof Object      // true
a2 instanceof Object      // true

function f1() {};
let f2 = function() {};
let f3 = new Function('', ';');
fx.constructor == Function // true, fx=f1,f2,f3
fx.constructor == Object   // false
fx instanceof Function     // true
fx instanceof Object       // true
```

El método `toString()` heredado de la pseudoclase `Object` es invocado automáticamente por JavaScript cuando necesita convertir el objeto en un *string*. La implementación por defecto retorna la cadena `"[object Object]"` que es poco significativa. Por ende, este método debería ser sobrescrito por clases descendientes.

El método `valueOf()` es invocado automáticamente por JavaScript cuando el objeto se utiliza en un contexto numérico. El programador también debería sobrescribirlo para clases propias, si su objeto puede convertirse automáticamente a un número.

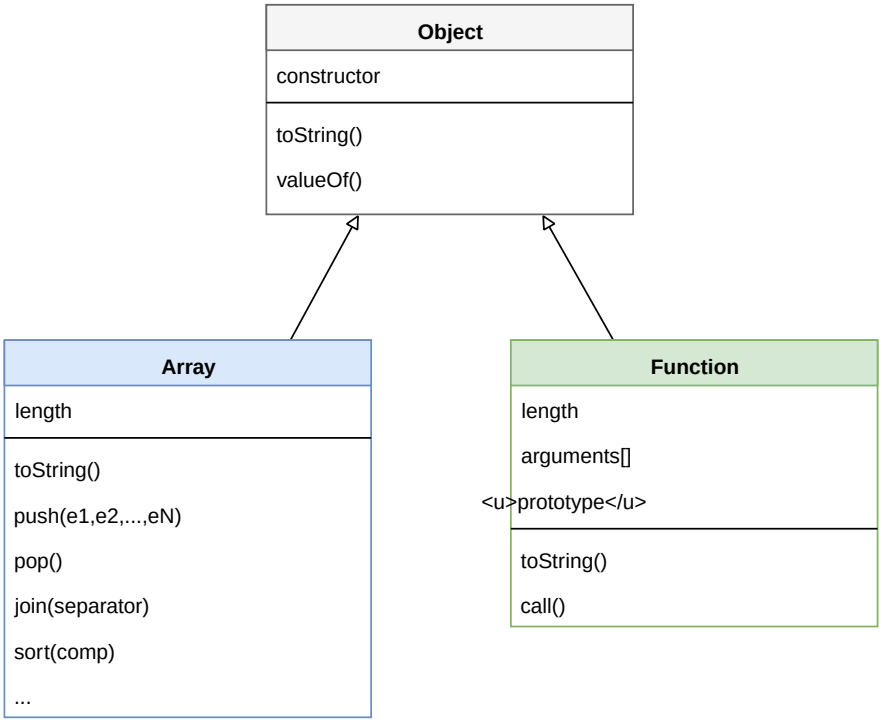


Figura 30. Algunas propiedades y métodos que JavaScript provee por defecto en los objetos, arreglos y funciones.

6.5. Pseudoclases

JavaScript no tiene un concepto fuerte de *clase* como ocurre en C++ o Java, donde una clase es necesaria para poder construir objetos. En JavaScript, las clases y la herencia se simulan con funciones inicializadoras, objetos, y prototipos de objetos. Una **función constructora** o **función inicializadora** es una función cualquiera que recibe un objeto por parámetro `this` y le crea propiedades y métodos. Por ejemplo:

Listado 86. Una función constructora para simular una clase Rectángulo

```
1 // Construye un rectángulo. Recibe un objeto en el parámetro oculto this
2 function Rectangle(width, height)
3 {
4    // Declara propiedades y las inicializa
5    this.width = width;
6    this.height = height;
7
8    // También puede crear métodos
9    this.area = function() { return this.width * this.height; }
10 }
```

La función `Rectangle()` del Listado 86 es una función global como cualquier otra. Si se define fuera de

un módulo, será una propiedad del objeto global (`window`, en caso de los navegadores web, o `global` en caso de Node.js). Si se invoca directamente, el valor de `this` será el objeto global (`window` o `global`) y las líneas 5 y 6 crearán en él dos nuevas propiedades `width` y `height` (si no existen ya), y la línea 9 crea otra propiedad más que almacena una función. Como se puede deducir, invocar una función constructora directamente carece de sentido.

Las funciones constructoras deben ser invocadas con el operador `new`. Este operador hace lo siguiente. Crea un objeto vacío, sin propiedades. Invoca a la función constructora pasándole el objeto vacío por parámetro para que lo incialice. La función constructora recibe entonces el nuevo objeto en su parámetro `this` y crea en él las propiedades esperadas. Una vez que la función constructora haya terminado su ejecución, el operador `new` retorna la dirección de memoria del nuevo objeto inicializado.

```
let rect1 = new Rectangle(2, 4);    // rect1 = { width:2, height:4 };
let rect2 = new Rectangle(8.5, 11); // rect2 = { width:8.5, height:11 };
console.log("Area of rect2: " + rect2.area() ); // Imprime "Area of rect2: 93.5"
```

Todos los objetos que sean inicializados con la función `Rectangle()` tendrán un `width` y un `heigth`, el desarrollador puede utilizar con confianza estas propiedades. La función constructora no debe retornar un valor, ya que reemplazará el resultado de la expresión `new`. Nótese que nunca hubo una clase, sino una función constructora y objetos tradicionales que son inicializados con ella. Por esto se les llama **pseudoclases** {*pseudoclass*} y la función constructora es quien da nombre a la pseudoclase.

Como es de esperarse, todos los objetos inicializados con `Rectangle()` tienen su propia copia independiente de `width` y `height`. Esto implica que en el ejemplo anterior, si a `rect2.width` se le asigna otro valor, no afectará al valor de `rect1.width`, como podría esperarse. Sin embargo, esto mismo ocurre con los métodos, y por tanto, cada objeto inicializado con la función constructura `Rectangle()` de la Listado 86 tendrá su propia copia independiente del método `area()`, lo cual es ineficiente. Esta redundancia se elimina con los *objetos prototipo*.

Todas las funciones que el programador defina, tendrán automáticamente otra propiedad llamada `prototype` creada por `Function`, de la misma forma que `length` y `arguments`. La propiedad `prototype` es un objeto, definido como una "variable estática" y por tanto, todas las invocaciones a la función tendrán acceso al mismo `prototype`.

La especificación ECMAScript exige a cada intérprete de JavaScript implementar el siguiente comportamiento. Cuando se intenta acceder a una propiedad `prop` en el objeto `obj`, de la forma `obj.prop` u `obj['prop']`, el intérprete buscará si `obj` tiene una propiedad `prop` declarada en él, y en tal caso la usará. De lo contrario, el intérprete buscará la propiedad `prop` en el objeto prototipo de `obj`, es decir, en `obj.prototype.prop`; si está declarada la usará, de lo contrario buscará la propiedad en el prototipo del prototipo (`obj.prototype.prototype.prop`), y así recursivamente hasta encontrar la propiedad o hasta que un prototipo sea `null`.

En otras palabras. Si el intento de acceder a la propiedad `obj.prop` es exitoso, es porque `obj` tiene una propiedad `prop`, o alguno de los prototipos de `obj` define la propiedad `prop`. Este comportamiento es aproximadamente el mismo que el logrado por la herencia de clases, y es la cadena de prototipos es el mecanismo que JavaScript utiliza para simular herencia.

Cada vez que se invoca el operador `new` con una función constructora, éste implícitamente crea una propiedad `prototype` al nuevo objeto y hace que apunte al `prototype` de la función constructora. De esta forma, cualquier propiedad que sea definida en el objeto prototipo de la función constructora (o pseudoclase), será compartida por todos los objetos inicializados con dicha pseudoclase. El pseudocódigo del Listado 87 explica lo que hace el operador `new` internamente:

Listado 87. Pseudocódigo del trabajo realizado internamente por el operador `new`

```
// El programador define una función constructora
function Constructor(a,b) { this.prop1 = a; this.prop2 = b; }

   // La pseudoclase Function inserta una propiedad length en la función implícitamente
   Constructor.length = 2;

   // La pseudoclase Function inserta una propiedad prototype en la función implícitamente
   Constructor.prototype = {};


// El programador crea un nuevo objeto con su función constructora
let obj1 = new Constructor(a,b);

   // El operador new hace esto internamente:

   // 1. Crea un objeto vacío
   let newObject = {};

   // 2. Crea una propiedad prototype en el objeto que apunta al prototipo de la función
   newObject.prototype = Constructor.prototype;

   // 3. Invoca a la función constructora para que inicialice el nuevo objeto, la cual
   // creará las propiedades prop1 y prop2
   Constructor.call(newObject, a, b);

   // 4. Retorna el objeto recién inicializado
   return newObject;

// obj1 tendrá al final las siguientes propiedades:
obj1.prop1
obj1.prop2
obj1.prototype
```

En resumen, cualquier propiedad asignada al objeto `Constructor.prototype`, será automáticamente una propiedad compartida por todos los objetos construidos con la función `Constructor`, incluso, aunque dicha propiedad sea asignada después de que los objetos fueron creados. De esta forma, las propiedades creadas por la función constructora directamente en el objeto serán copias independientes en cada objeto, y las propiedades creadas en el prototipo de la misma, serán propiedades compartidas por todos los objetos (lo que en C++ y Java se conoce como miembros estáticos). Por ejemplo:

```
// Construye un rectángulo. Recibe un objeto en el parámetro oculto this
function Rectangle(width, height)
{
   // Cada objeto tendrá una copia independiente de estas propiedades
   this.width = width;
   this.height = height;
}

// Este método será compartido por todos los objetos creados con new Rectangle()
Rectangle.prototype.area = function() { return this.width * this.height; }
```

Nótese que el método `area()` no se definió dentro de la función constructora. Si eso se hubiera hecho, se haría la asignación por cada rectángulo creado durante la ejecución del programa, lo cual es ineficiente.

El `prototype` es el lugar ideal para definir métodos, constantes y variables compartidas por todos los objetos de una misma pseudoclase. El programador querrá la mayor parte del tiempo tratar estas propiedades como de sólo lectura. Si modifica una propiedad en el prototipo directamente, el cambio afectará a todos los demás objetos inmediatamente; pero si intenta modificar la propiedad a través de un objeto cualquiera, creará una nueva propiedad en ese objeto que oculta la del prototipo. El Listado 88 compara el efecto de definir métodos en objetos y el prototipo.

Listado 88. Conflicto de propiedades en el objeto y el prototipo (object_prototype.html).

```
// El programador define una pseudoclase con una función constructora
function Circle(radius) { this.radius = radius; }

// Y un método que será compartido por todos los objetos creados con new Rectangle()
Circle.prototype.area = function() { return Math.PI * this.radius * this.radius; }

// circle1 y circle2 tienen su respectivo radius, son copias independientes
let circle1 = new Circle(7);
let circle2 = new Circle(2.5);

// circle1 y circle2 comparten el mismo método area()
if ( circle1.area === circle2.area )
{
   console.log('circle1.area() = ', circle1.area() ); // Imprime 153.94
   console.log('circle2.area() = ', circle2.area() ); // Imprime 19.63
}
else
   console.log('Circles do not share the same area() method!');

// El programador trata de modificar el método area() a traves de un objeto
circle2.area = function() { return Math.PI * this.radius * this.radius / 2; }

// Lo anterior crea una propiedad area() sólo en circle2 que tapa la del prototipo
console.log('circle1.area() = ', circle1.area() ); // Imprime 153.94
console.log('circle2.area() = ', circle2.area() ); // Imprime 9.82

// Al modificar una propiedad del prototipo afecta a todos los objetos, creados o nuevos
Circle.prototype.area = function() { return 0; }

// Un nuevo objeto aparece
let circle3 = new Circle(3.1);

// Todos los objetos reflejan el cambio excepto aquellos que han sobrescrito la propiedad
console.log('circle1.area() = ', circle1.area() ); // Imprime 0
console.log('circle2.area() = ', circle2.area() ); // Imprime 9.82
console.log('circle3.area() = ', circle3.area() ); // Imprime 0
```

Idealmente cuando un programador define una pseudoclase, debe redefinir algunos métodos heredados de la pseudoclase `Object`. El método `toString()` debe regresar una representación "string" del objeto. Opcionalmente puede redefinir el método `parse()`. Si su pseudoclase se puede convertir en un valor primitivo, implemente también `valueOf()`.

Ejercicio 81 [random_document_1, 80 pts]

Escriba una jerarquía de pseudoclases (funciones constructoras) que generen un documento HTML válido de longitud y estructura aleatoria. Por ejemplo, comience con la pseudoclase `Document`. Su función constructora crea las propiedades de un documento típico: el encabezado y el cuerpo, los cuales son a su vez otros objetos construidos con pseudoclases.

La función constructora de `Body` crea un arreglo de una cantidad aleatoria de elementos hijos escogidos también aleatoriamente. Para efectos de este ejercicio, basta con la siguiente jerarquía.

```
Document: Header, Body
Header: Style
Body: (Paragraph|Heading|List)+
List: ListItem+
ListItem: Text|Paragraph+|List
```

De esta forma cada elemento de su jerarquía tendrá al menos: un arreglo de elementos hijos (llámese `children`); y un método `print()` que imprime el elemento usando `console.log()`, y propaga el llamado a todos sus hijos. El método `print()` debe declararse en el prototipo de la función constructora para que sea compartido por todos los objetos de la misma pseudoclase.

Escriba un documento HTML que cree un objeto `Document` y llame su método `print()`, el cual debe imprimir el cuerpo del documento. Sugerencia, puede utilizar el siguiente ejemplo:

```
<!DOCTYPE html>
<html>
<head>
   <title>Random document</title>
   <script src="random_doc.js"></script>
   <script>
      let doc = new Document();
      doc.printHeader();
   </script>
</head>

<body>
   <script>
      doc.print();
   </script>
</body>
</html>
```

Ejercicio 82 [random_document_2, 40 pts]

Implemente en su jerarquía de elementos una pseudoclase `Style` que es hijo de `Header` y que se encarga de generar estilos CSS aleatorios en el documento, de tal forma que al refrescar, la apariencia del mismo sea compleamente aleatoria. Para esto, agregue un método `printHeader()` a su pseudoclase `Document`, el cual invoca al `print()` de `Header` y éste al de `Style`.

6.5.1. Clases

En otros lenguajes de programación como C++ y Java, es obligatorio definir una clase para poder construir un objeto. Ese no es el caso de JavaScript, donde los objetos puede crearse sin clases. El concepto de clase se agregó hasta ECMAScript6.

Referencias

Brown, D. M. (2011). *Communicating Design: Developing Web Site Documentation for Design and Planning* (2nd ed.). New Riders.

Connolly, R., & Hoar, R. (2018). *Fundamentals of Web Development* (2nd ed., p. 1240). Pearson.

Flanagan, D. (2020). *JavaScript: The Definitive Guide* (7th ed., p. 706). O'Reilly Media, Inc. https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/

Goldfarb, C., & Prescod, P. (1999). *Manual de XML* (1st ed.). Prentice Hall.

Laubheimer, P. (2016). Wireflows: A UX Deliverable for Workflows and Apps. In *Nielsen Norman Group*. https://www.nngroup.com/articles/wireflows/

Pólya, G. (1957). *How to Solve It* (2nd ed.). Anchor Books.

Ravisankar, V. (2018). *2018 HackerRank Student Developer Report* (p. 11). HackerRank. https://research.hackerrank.com/student-developer/2018

Ravisankar, V. (2019). *2019 HackerRank Developer Skills Report* (p. 18). HackerRank. https://research.hackerrank.com/developer-skills/2019

Ravisankar, V. (2020). *2020 HackerRank Developer Skills Report* (p. 25). HackerRank. https://research.hackerrank.com/developer-skills/2020/

Robertson, I. (2016). *Problem Solving, Perspectives from Cognition and Neuroscience* (2nd ed., p. 286). Psychology Press.

Rosenfeld, L., Morville, P., & Arango, J. (2015). Information Architecture: For the Web and Beyond. In *Book* (4th ed., p. 486). O'Reilly Media, Inc. http://oreilly.com/catalog/errata.csp?isbn=9781491911686

Tullis, T. (T., & Albert, B. (W. (2013). *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics* (2nd ed., p. 320). Elsevier Inc. http://www.sciencedirect.com/science/book/9780124157811

W3C. (2008). *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. World Wide Web Consortium. https://www.w3.org/TR/xml/

WHATWG. (2020). *HTML Living Standard*. https://html.spec.whatwg.org/

Índice de materias