SUPERIOR TECNOLÓGICO
DE TURISMO Y PATRIMONIO
YAVIRAC

Guía de estudio

Programación de aplicaciones WEB

Freddy Heredia

GUÍA DE ASIGNATURA: PROGRAMACIÓN DE APLICACIONES WEB

YAVIRAC.EDU.EC

Freddy Heredia fheredia@yavirac.edu.ec

Versión, Mayo 2022-1

Índice general

1. Parcial 1

1.1 Sobre esta guía

La siguiente guía tiene como objetivo presentar de manera sistemática los contenidos teórico práctico de la asignatura **Programación de aplicaciones web** en base al plan de carrera del Instituto Tecnológico Benito Juárez en su rediseño del año 2016 para Desarrollo de Software.

Datos generales de la asignatura

Nombre de la asignatura:	Programación Aplicaciones Web
Campo de formación:	Adaptación tecnológica e innovación
Unidad de organización curricular:	Formación técnica profesional
Número de período académico:	4
Número de horas de la asignatura:	122
Número de horas por cada componente:	Docencia: 60 Prácticas de aprendizaje: 24 Aprendizaje autónomo: 38
Docente:	Freddy Heredia: `fheredia@yavirac.edu.ec`

1.2 Lo que aprenderás

1.3 El origen

[1]Aunque los inicios de Internet se remontan a los años setenta, no ha sido hasta los años noventa cuando, gracias a la Web, se ha extendido su uso por todo el mundo. En pocos años la Web ha evolucionado enormemente: se ha pasado de páginas sencillas, con pocas imágenes y contenidos estáticos a páginas complejas con contenidos dinámicos que provienen de bases de datos, lo que permite la creación de .ªplicaciones web".

[1]Internet y la Web han incluido enormemente tanto en el mundo de la informática como en la sociedad en general. Si nos centramos en la Web, en poco menos de 10 años ha transformado los sistemas informáticos: ha roto las barreras físicas (debido a la distancia), económicas y lógicas

(debido al empleo de distintos sistemas operativos, protocolos, etc.) y ha abierto todo un abanico de nuevas posibilidades. Una de las áreas que más expansión está teniendo en la Web en los últimos años son las aplicaciones web.

[1]Las aplicaciones web permiten la generación automática de contenido, la creación de páginas personalizadas según el perfil del usuario o el desarrollo del comercio electrónico. Además, una aplicación web permite interactuar con los sistemas informáticos de gestión de una empresa, como puede ser gestión de clientes, contabilidad o inventario, a través de una página web.

[1]De forma breve, una aplicación web se puede definir como una aplicación en la cual el usuario por medio de un navegador realiza peticiones a una aplicación remota accesible a través de internet (o a través de una Intranet) y que recibe una respuesta que se muestra en el propio navegador.

R `https://rua.ua.es/dspace/bitstream/10045/16995/1/sergio_lujan-programacion_de_aplicaciones_web.pdf`

1.3.1 Protocolos de internet

El éxito de Internet se basa mucho en el empleo de TCP/IP, el conjunto de protocolos de comunicación que permiten el intercambio de información de forma independiente de los sistemas en que ésta se encuentra almacenada. TCP/IP constituye la solución problema de heterogeneidad de los sistemas informáticos. El 1 de enero de 1983, TCP/IP se estableció como el protocolo estándar de comunicación en Internet.
El conjunto de protocolos TCP/IP, también llamado la pila de protocolos TCP/IP, incluye una serie de protocolos que se encuentran en el nivel 7 o de aplicación de la arquitectura Open System Interconnection (OSI) y que proporcionan una serie de servicios.
Como un mismo ordenador puede atender varios servicios, cada servicio se identifica con un número llamado puerto. Por tanto, a cada protocolo le corresponde un número de puerto. Los protocolos que se encuentran estandarizados poseen un puerto reservado que no puede emplear ningún otro protocolo.

En el siguiente cuadro se muestran los protocolos del nivel 7 más comunes de Internet junto con el número de puerto que emplean.

[3]En la siguiente imagen se muestran las capas del modelo OSI:

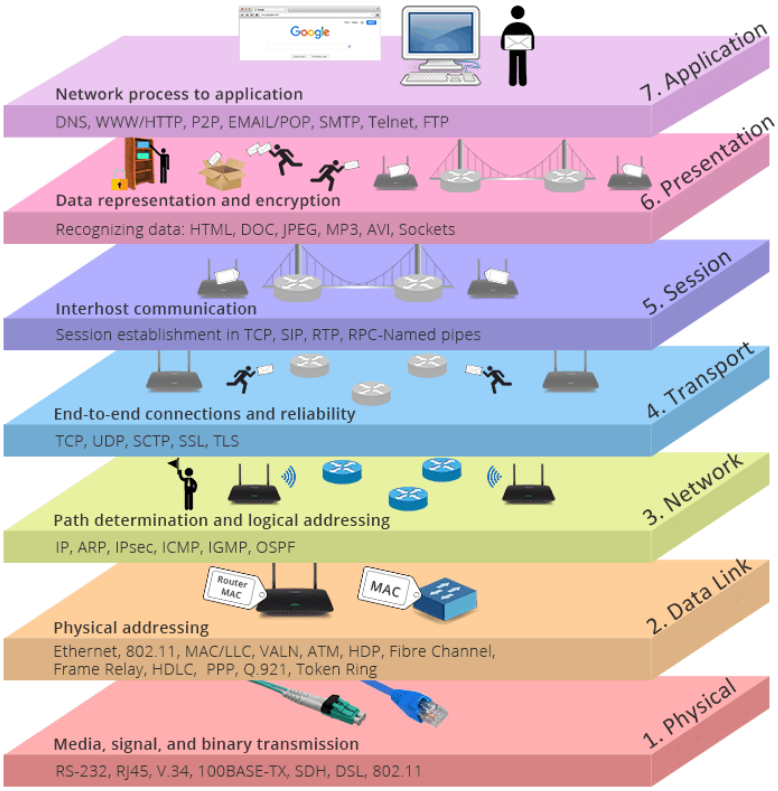

Figura 1.1: Modelo OSI

Véa el siguiente vídeo: Guerreros de la red
`https://youtu.be/1c2U1R8XXvA`

1.4 Arquitecturas cliente/servidor

[1]Las aplicaciones web son un tipo especial de aplicaciones cliente/servidor. Antes de aprender a programar aplicaciones web conviene conocer las características básicas de las arquitecturas cliente/servidor.

Cliente/servidor es una arquitectura de red en la que cada ordenador o proceso en la red es cliente o servidor . Normalmente, los servidores son ordenadores potentes dedicados a gestionar unidades de disco (servidor de ficheros), impresoras (servidor de impresoras), tráco de red (servidor de red), datos (servidor de bases de datos) o incluso aplicaciones (servidor de aplicaciones), mientras que los clientes son máquinas menos potentes y usan los recursos que ofrecen los servidores.

Esta arquitectura implica la existencia de una relación entre procesos que solicitan servicios (clientes) y procesos que responden a estos servicios (servidores). Estos dos tipos de procesos pueden ejecutarse en el mismo procesador o en distintos.

La arquitectura cliente/servidor permite la creación de aplicaciones distribuidas. La principal ventaja de esta arquitectura es que facilita la separación de las funciones según su servicio, permitiendo situar cada función en la plataforma más adecuada para su ejecución. Además, también presenta las siguientes ventajas:

- Las redes de ordenadores permiten que múltiples procesadores puedan ejecutar partes distribuidas de una misma aplicación, logrando concurrencia de procesos.

- Existe la posibilidad de migrar aplicaciones de un procesador a otro con modificaciones mínimas en los programas.
- Se obtiene una escalabilidad de la aplicación. Permite la ampliación horizontal o vertical de las aplicaciones. La **escalabilidad horizontal** se refiere a la capacidad de añadir o suprimir estaciones de trabajo que hagan uso de la aplicación (clientes), sin que afecte sustancialmente al rendimiento general, La **escalabilidad vertical** se refiere a la capacidad de migrar hacia servidores de mayor capacidad o velocidad, o de un tipo distinto de arquitectura sin que afecte a los clientes.
- Posibilita el acceso a los datos independientemente de donde se encuentre el usuario.

1.4.1 Separación de funciones

La arquitectura cliente/servidor nos permite la separación de funciones en tres niveles:

- **Lógica de presentación**. Se encarga de la entrada y salida de la aplicación con el usuario. Sus principales tareas son: obtener información del usuario, enviar la información del usuario a la lógica de negocio para su procesamiento, recibir los resultados del procesamiento de la lógica de negocio y presentar estos resultados al usuario.
- **Lógica de negocio (o aplicación).** Se encarga de gestionar los datos a nivel de procesamiento. Actúa de puente entre el usuario y los datos. Sus principales tareas son: recibir la entrada del nivel de presentación, interactuar con la lógica de datos para ejecutar las reglas de negocio (business rules) que tiene que cumplir la aplicación (facturación, cálculo de nóminas, control de inventario, etc.) y enviar el resultado del procesamiento al nivel de presentación.
- **Lógica de datos.** Se encarga de gestionar los datos a nivel de almacenamiento. Sus principales tareas son: almacenar los datos, recuperar los datos, mantener los datos y asegurar la integridad de los datos.

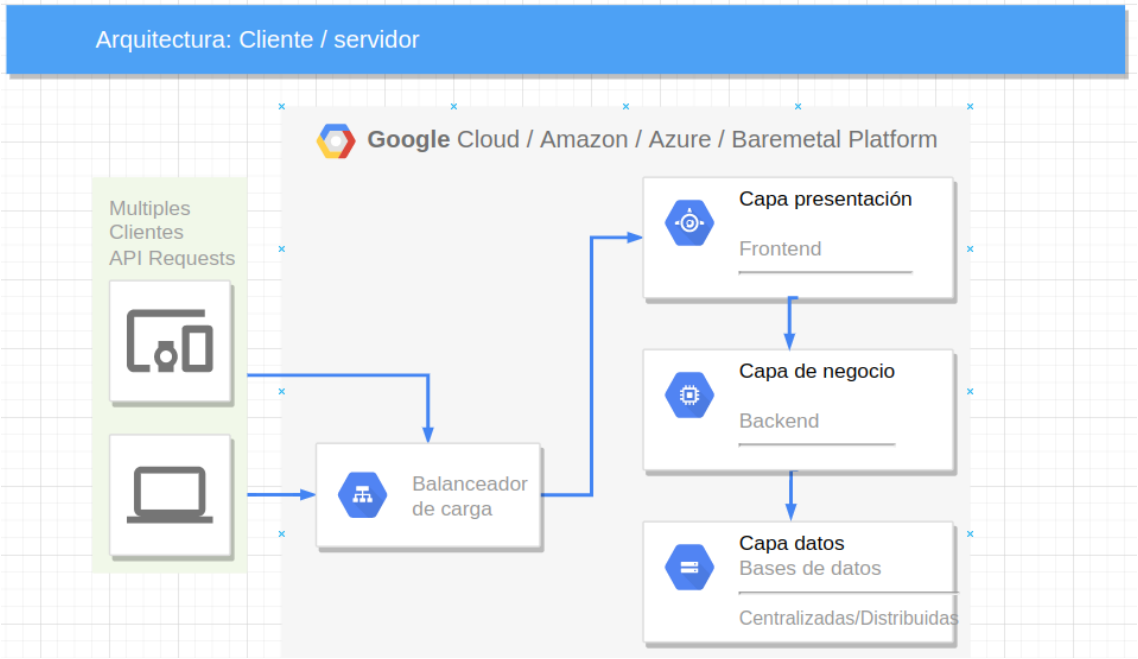

Figura 1.2: Arquitectura cliente/servidor

1.4.2 El cliente

[2]El cliente es cualquier herramienta que actué en representación del usuario para solicitar a un servidor web el envío de los recursos que desea obtener mediante HTTP. Esta función es realizada

en la mayor parte de los casos por un navegador Web. Hay excepciones, como el caso de programas específicamente usados por desarrolladores para desarrollar y depurar sus aplicaciones.
El navegador es siempre el que inicia una comunicación (petición), y el servidor nunca la comienza (hay algunos mecanismos que permiten esto, pero no son muy habituales).
Para poder mostrar una página Web, el navegador envía una petición de documento HTML al servidor. Entonces procesa este documento, y envía más peticiones para solicitar scripts, hojas de estilo (CSS), y otros datos que necesite (normalmente vídeos y/o imágenes). El navegador, une todos estos documentos y datos, y compone el resultado final: la página Web. Los scripts, los ejecuta también el navegador, y también pueden generar más peticiones de datos en el tiempo, y el navegador, gestionará y actualizará la página Web en consecuencia.
Una página Web, es un documento de hipertexto (HTTP), luego habrá partes del texto en la página que puedan ser enlaces (links) que pueden ser activados (normalmente al hacer click sobre ellos) para hacer una petición de una nueva página Web, permitiendo así dirigir su agente de usuario y navegar por la Web. El navegador, traduce esas direcciones en peticiones de HTTP, e interpretara y procesará las respuestas HTTP, para presentar al usuario la página Web que desea.
La parte cliente de las aplicaciones web suele estar formada por el código HTML que forma la página web más algo de código ejecutable realizado en lenguaje de script del navegador (JavaScript o VBScript) o mediante pequeños programas (applets) realizados en Java. También se solían emplear plugins que permiten visualizar otros contenidos multimedia (como Macromedia Flash), aunque no se encuentran tan extendidos como las tecnologías anteriores y plantean problemas de incompatibilidad entre distintas plataformas. Por tanto, la misión del cliente web es interpretar las páginas HTML y los diferentes recursos que contienen (imágenes, sonidos, etc.).
Las tecnologías que se suelen emplear para programar el cliente web son:

- HTML
- CSS
- Javascript

Cada tecnología puede tener sus variantes.
WebAssembly es un proyecto prometedor.
Algunas tecnologías ya están en desuso (Applets, Flash entre otras.)

1.4.3 El servidor

Al otro lado del canal de comunicación, está el servidor, el cual "sirve"los datos que ha pedido el cliente. Un servidor conceptualmente es una unica entidad, aunque puede estar formado por varios elementos, que se reparten la carga de peticiones, (load balancing), u otros programas, que gestionan otros computadores (como cache, bases de datos, servidores de correo electrónico, ...), y que generan parte o todo el documento que ha sido pedido.
Un servidor no tiene que ser necesariamente un único equipo físico, aunque si que varios servidores pueden estar funcionando en un único computador. En el estándar HTTP/1.1 y Host , pueden incluso compartir la misma dirección de IP.

1.4.4 Transferencia de páginas web a través de http

El proceso completo, desde que el usuario solicita una página, hasta que el cliente web (navegador) se la muestra con el formato apropiado, es el siguiente:

1. El usuario especifica en el cliente web la dirección de la página que desea consultar: el usuario escribe en el navegador la dirección (URL) de la página que desea visitar o pulsa un enlace.
2. El cliente establece una conexión con el servidor web.
3. El cliente solicita la página o el objeto deseado.

4. El servidor envía dicha página u objeto (o, si no existe, devuelve un código de error).
5. Si se trata de una página HTML, el cliente inicia sus labores de interpretación de los códigos HTML. Si el cliente web encuentra instrucciones que hacen referencia a otros objetos que se tienen que mostrar con la página (imágenes, sonidos, animaciones multimedia, etc.), establece automáticamente comunicación con el servidor web para solicitar dichos objetos.
6. Se cierra la conexión entre el cliente y el servidor.
7. Se muestra la página al usuario.

Obsérvese que siempre se libera la conexión, por lo que ésta sólo tiene la duración correspondiente a la transmisión de la página solicitada. Esto se hace así para no desperdiciar innecesariamente el ancho de banda de la red mientras el usuario lee la página recibida. Cuando el usuario activa un enlace de la página, se establece una nueva conexión para recibir otra página o elemento multimedia. Por ello, el usuario tiene la sensación de que está disfrutando de una conexión permanente cuando realmente no es así. Un detalle importante es que para cada objeto que se transfiere por la red se realiza una conexión independiente. Por ejemplo, si el cliente web solicita una página que contiene dos imágenes integradas, se realizan tres conexiones: una para el documento HTML y dos para los archivos de las imágenes.

Una aplicación web (web-based application) es un tipo especial de aplicación cliente/servidor, donde tanto el cliente (el navegador, explorador o visualizador/browser)) como el servidor (el servidor web) y el protocolo mediante el que se comunican (HTTP) están estandarizados y no han de ser creados por el programador de aplicaciones.
El protocolo HTTP forma parte de la familia de protocolos de comunicaciones TCP/IP, que son los empleados en Internet. Estos protocolos permiten la conexión de sistemas heterogéneos, lo que facilita el intercambio de información entre distintos ordenadores. HTTP se sitúa en el nivel 7 (aplicación) del modelo OSI.

1.5 HTTP/HTTPS

[2]Hypertext Transfer Protocol (HTTP) (o Protocolo de Transferencia de Hipertexto en español) es un protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML. Fue diseñado para la comunicación entre los navegadores y servidores web, aunque puede ser utilizado para otros propósitos también. Sigue el clásico modelo cliente-servidor, en el que un cliente establece una conexión, realizando una petición a un servidor y espera una respuesta del mismo. Se trata de un protocolo sin estado, lo que significa que el servidor no guarda ningún dato (estado) entre dos peticiones. Aunque en la mayoría de casos se basa en una conexión del tipo TCP/IP, puede ser usado sobre cualquier capa de transporte segura o de confianza, es decir, sobre cualquier protocolo que no pierda mensajes silenciosamente, tal como UDP.

Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.

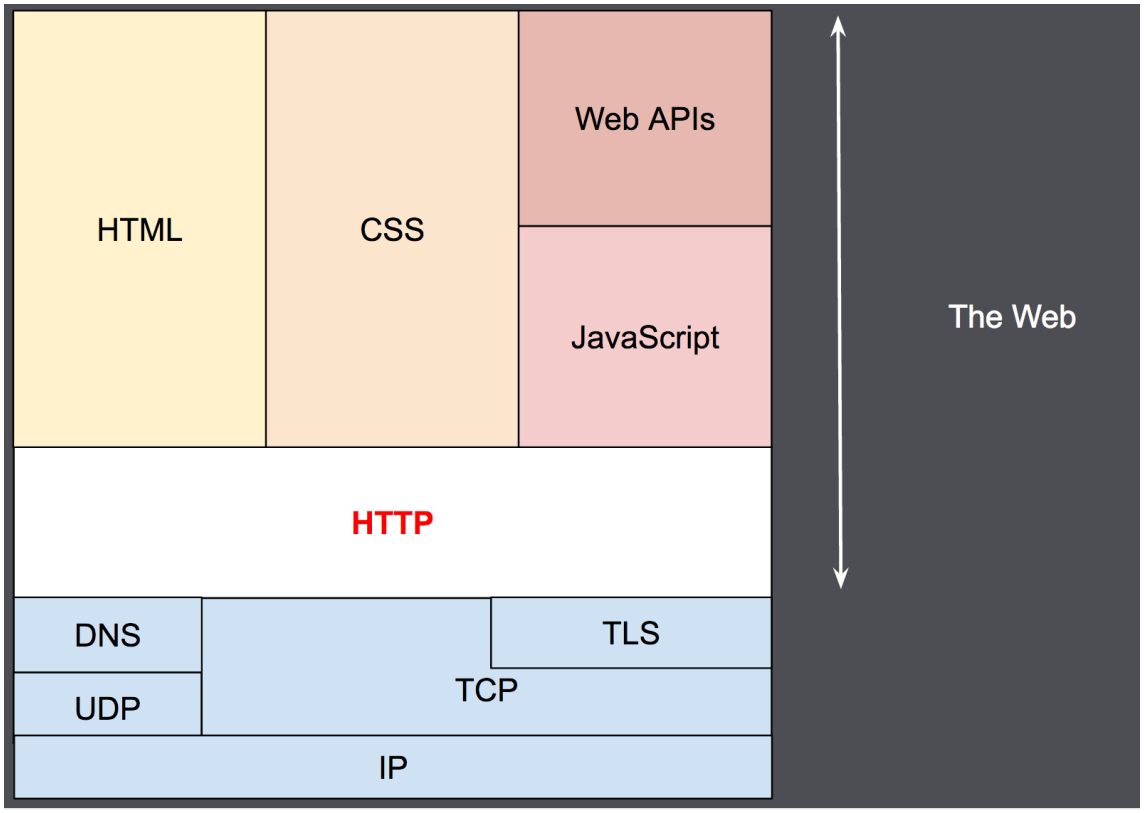

Figura 1.3: HTTP y capas

Diseñado a principios de la década de 1990, HTTP es un protocolo ampliable, que ha ido evolucionando con el tiempo. Es lo que se conoce como un protocolo de la capa de aplicación, y se transmite sobre el protocolo TCP, o el protocolo encriptado TLS (en-US), aunque teóricamente podría usarse cualquier otro protocolo fiable. Gracias a que es un protocolo capaz de ampliarse, se usa no solo para transmitir documentos de hipertexto (HTML), si no que además, se usa para transmitir imágenes o vídeos, o enviar datos o contenido a los servidores, como en el caso de los formularios de datos. HTTP puede incluso ser utilizado para transmitir partes de documentos, y actualizar páginas Web en el acto.

En realidad, hay más elementos intermedios, entre un navegador y el servidor que gestiona su petición: hay otros tipos de dispositivos: como routers, modems ... Es gracias a la arquitectura en capas de la Web, que estos intermediarios, son transparentes al navegador y al servidor, ya que HTTP se apoya en los protocolos de red y transporte. HTTP es un protocolo de aplicación, y por tanto se apoya sobre los anteriores. Aunque para diagnosticar problemas en redes de comunicación, las capas inferiores son irrelevantes para la definición del protocolo HTTP .

1.5.1 Características clave del protocolo HTTP

- **HTTP es sencillo**: HTTP esta pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más facil la depuración de errores, y reduciendo la curva de aprendizaje para las personan que empieza a trabajar con él.
- **HTTP es extensible**: Presentadas en la versión HTTP/1.0, las cabeceras de HTTP, han hecho que este protocolo sea fácil de ampliar y de experimentar con él. Funcionalidades nuevas pueden desarrollarse, sin más que un cliente y su servidor, comprendan la misma semántica sobre las cabeceras de HTTP.
- **HTTP es un protocolo con sesiones, pero sin estados**: HTTP es un protocolo sin estado, es

decir: no guarda ningún dato entre dos peticiones en la mísma sesión. Esto crea problemáticas, en caso de que los usuarios requieran interactuar con determinadas páginas Web de forma ordenada y coherente, por ejemplo, para el uso de çestas de la compra.[en] páginas que utilizan en comercio electrónico. Pero, mientras HTTP ciertamente es un protocolo sin estado, el uso de HTTP cookies, si permite guardar datos con respecto a la sesión de comunicación. Usando la capacidad de ampliación del protocolo HTTP, las cookies permiten crear un contexto común para cada sesión de comunicación.

- **HTTP y conexiones:** Una conexión se gestiona al nivel de la capa de trasporte, y por tanto queda fuera del alcance del protocolo HTTP. Aún con este factor, HTTP no necesita que el protocolo que lo sustenta mantenga una conexión continua entre los participantes en la comunicación, solamente necesita que sea un protocolo fiable o que no pierda mensajes (como mínimo, en todo caso, un protocolo que sea capaz de detectar que se ha pedido un mensaje y reporte un error). De los dos protocolos más comunes en Internet, TCP es fiable, mientras que UDP, no lo es. Por lo tanto HTTP, se apoya en el uso del protocolo TCP, que está orientado a conexión, aunque una conexión continua no es necesaria siempre.
  Todavía hoy se sigue investigando y desarrollando para conseguir un protocolo de transporte más conveniente para el HTTP. Por ejemplo, Google está experimentado con QUIC, que se apoya en el protocolo UDP y presenta mejoras en la fiabilidad y eficiencia de la comunicación.

1.5.2 ¿Qué se puede controlar con HTTP?

Se presenta a continuación una lista con los elementos que se pueden controlar con el protocolo HTTP:

- **Cache**: El como se almacenan los documentos en la caché, puede ser especificado por HTTP. El servidor puede indicar a los proxies y clientes, que quiere almacenar y durante cuanto tiempo. Aunque el cliente, también puede indicar a los proxies de caché intermedios que ignoren el documento almacenado.
- **Flexibilidad del requisito de origen** Para prevenir invasiones de la privacidad de los usuarios, los navegadores Web, solamente permiten a páginas del mismo origen, compartir la información o datos. Esto es una complicación para el servidor, asi que mediante cabeceras HTTP, se puede flexibilizar o relajar esta división entre cliente y servidor
- **Autentificación** Hay páginas Web, que pueden estar protegidas, de manera que solo los usuarios autorizados puedan acceder. HTTP provee de servicios básicos de autentificación, por ejemplo mediante el uso de cabeceras como: WWW-Authenticate, o estableciendo una sesión especifica mediante el uso de HTTP cookies.
- **Proxies y tunneling** Servidores y/o clientes pueden estar en intranets y esconder así su verdadera dirección IP a otros. Las peticiones HTTP utilizan los proxies para acceder a ellos. Pero no todos los proxies son HTTP proxies. El protocolo SOCKS, por ejemplo, opera a un nivel más bajo. Otros protocolos, como el FTP, pueden ser servidos mediante estos proxies.
- **Sesiones** El uso de HTTP cookies permite relacionar peticiones con el estado del servidor. Esto define las sesiones, a pesar de que por definición el protocolo HTTP es un protocolo sin estado. Esto es muy útil no sólo para aplicaciones de comercio electrónico, sino también para cualquier sitio que permita configuración al usuario.

1.5.3 Mensajes HTTP

Existen dos tipos de mensajes HTTP: peticiones y respuestas, cada uno sigue su propio formato. Las peticiones y respuestas HTTP, comparten una estructura similar, compuesta de:

- Una línea de inicio (’start-line’ en inglés) describiendo la petición a ser implementada, o su estado, sea de éxito o fracaso. Esta línea de comienzo, es siempre una única línea.
- Un grupo opcional de cabeceras HTTP, indicando la petición o describiendo el cuerpo (’body’

en inglés) que se incluye en el mensaje.
- Una línea vacía (’empty-line’ en inglés) indicando toda la meta-información ha sido enviada.
- Un campo de cuerpo de mensaje opcional (’body’ en inglés) que lleva los datos asociados con la petición (como contenido de un formulario HTML), o los archivos o documentos asociados a una respuesta (como una página HTML, o un archivo de audio, vídeo ...) . La presencia del cuerpo y su tamaño es indicada en la línea de inicio y las cabeceras HTTP.

La línea de inicio y las cabeceras HTTP, del mensaje, son conocidas como la cabeza de la peticiones, mientras que su contenido en datos se conoce como el cuerpo del mensaje.

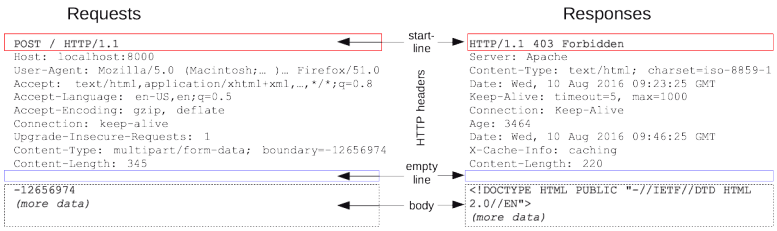

Figura 1.4: Mensajes

Cabeceras

Las cabeceras HTTP de una petición siguen la misma estructura que la de una cabecera HTTP. Una cadena de caracteres, que no diferencia mayusculas ni minusculas, seguida por dos puntos (’:’) y un valor cuya estructura depende de la cabecera. La cabecera completa, incluido el valor, ha de ser formada en una única línea, y pude ser bastante larga.

Hay bastantes cabeceras posibles. Estas se pueden clasificar en varios grupos:
- Cabeceras generales, (’General headers’ en inglés), como Via (en-US), afectan al mensaje como una unidad completa.
- Cabeceras de petición, (’Request headers’ en inglés), como User-Agent, Accept-Type, modifican la petición especificándola en mayor detalle (como: Accept-Language (en-US), o dándole un contexto, como: Referer, o restringiéndola condicionalmente, como: If-None.
- Cabeceras de entidad, (’Entity headers’ en ingles), como Content-Length las cuales se aplican al cuerpo de la petición. Por supuesto, esta cabecera no necesita ser transmitida si el mensaje no tiene cuerpo (’body’ en inglés).

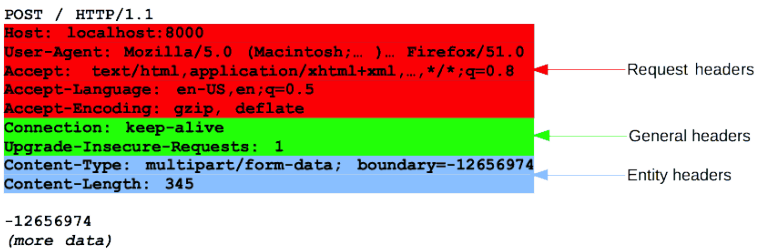

Figura 1.5: Cabeceras

Cuerpo

La parte final de la petición el el cuerpo. No todas las peticiones llevan uno: las peticiones que reclaman datos, como GET, HEAD, DELETE, o OPTIONS, normalmente, no necesitan ningún cuerpo. Algunas peticiones pueden mandar peticiones al servidor con el fin de actualizarlo: como es el caso con la petición POST (que contiene datos de un formulario HTML).
Los cuerpos pueden ser divididos en dos categorias:

- Cuerpos con un único dato, que consisten en un único archivo defindo por las dos cabeceras: Content-Type y Content-Length.
- Cuerpos con múltiples datos, que están formados por distintos contenidos, normalmente estan asociados con los formularios HTML.

Peticiones

Un ejemplo de petición HTTP:

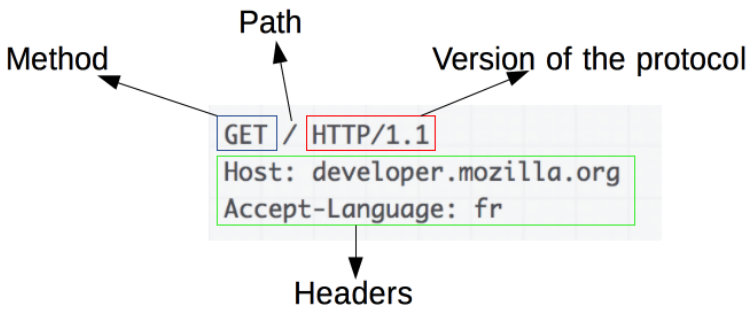

Figura 1.6: Petición

Una petición de HTTP, está formado por los siguientes campos:

- Un método HTTP, normalmente pueden ser un verbo, como: GET, POST o un nombre como: OPTIONS (en-US) o HEAD (en-US), que defina la operación que el cliente quiera realizar. El objetivo de un cliente, suele ser una petición de recursos, usando GET, o presentar un valor de un formulario HTML, usando POST, aunque en otras ocasiones puede hacer otros tipos de peticiones.
- La dirección del recurso pedido; la URL del recurso, sin los elementos obvios por el contexto, como pueden ser: sin el protocolo (http://), el dominio (aquí developer.mozilla.org), o el puerto TCP (aquí el 80).
- La versión del protocolo HTTP.
- Cabeceras HTTP opcionales, que pueden aportar información adicional a los servidores.
- O un cuerpo de mensaje, en algún método, como puede ser POST, en el cual envía la información para el servidor.

Respuestas

Un ejemplo de repuesta:

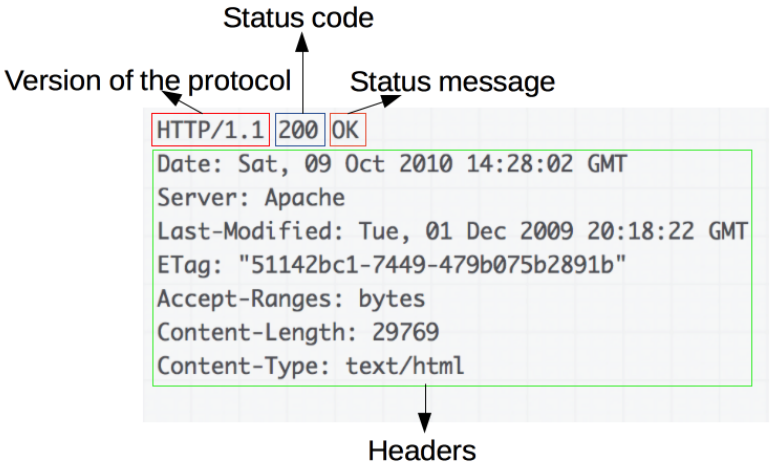

Figura 1.7: Respuesta

Las respuestas están formadas por los siguentes campos:

- La versión del protocolo HTTP que están usando.
- Un código de estado, indicando si la petición ha sido exitosa, o no, y debido a que. Códigos de estado muy comunes son: 200, 404, o 302
- Un mensaje de estado, una breve descripción del código de estado.
- Cabeceras HTTP, como las de las peticiones.
- Opcionalmente, el recurso que se ha pedido.

1.5.4 Métodos de petición HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementan una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser safe, idempotent (en-US), o cacheable.

GET

El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

Request has body	No
Successful response has body	Yes
Safe	Yes
Idempotent	Yes
Cacheable	Yes
Allowed in HTML forms	Yes

Figura 1.8: GET

HEAD

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

POST

El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
El tipo de cuerpo de la solicitud se indica mediante el encabezado Content-Type.

Request has body	Yes
Successful response has body	Yes
Safe	No
Idempotent	No
Cacheable	Only if freshness information is included
Allowed in HTML forms	Yes

Figura 1.9: POST

PUT

El método de solicitud HTTP PUT crea un nuevo recurso o reemplaza una representación del recurso de destino con la carga útil de la solicitud.

La diferencia entre PUT y POST es que PUT es idempotente: llamarlo una o varias veces sucesivamente tiene el mismo efecto (eso no es un efecto secundario), mientras que las sucesivas solicitudes POST idénticas pueden tener efectos adicionales, como realizar un pedido varias veces.

Request has body	Yes
Successful response has body	No
Safe	No
Idempotent	Yes
Cacheable	No
Allowed in HTML forms	No

Figura 1.10: PUT

DELETE

El método DELETE borra un recurso en específico.

Request has body	May
Successful response has body	May
Safe	No
Idempotent	Yes
Cacheable	No
Allowed in HTML forms	No

Figura 1.11: DELETE

PATCH

El método PATCH es utilizado para aplicar modificaciones parciales a un recurso. PATCH es algo análogo al concepto de .ªctualización"que se encuentra en CRUD (en general, HTTP es diferente a CRUD y no se deben confundir los dos).

Request has body	Yes
Successful response has body	Yes
Safe	No
Idempotent	No
Cacheable	No
Allowed in HTML forms	No

Figura 1.12: PATHC

R Un método HTTP es seguro si no altera el estado del servidor. En otras palabras, un método es seguro si conduce a una operación de solo lectura. Varios métodos HTTP comunes son seguros: GET, HEAD u OPTIONS. Todos los métodos seguros también son idempotentes, pero no todos los métodos idempotentes son seguros. Por ejemplo, PUT y DELETE son idempotentes pero inseguros.

R Una respuesta almacenable en caché es una respuesta HTTP que se puede almacenar en caché, que se almacena para recuperarla y usarla más tarde, guardando una nueva solicitud en el servidor. No todas las respuestas HTTP se pueden almacenar en caché

R Un método HTTP es idempotente si se puede realizar una solicitud idéntica una o varias veces seguidas con el mismo efecto y dejando el servidor en el mismo estado. En otras palabras, un método idempotente no debería tener efectos secundarios (excepto para llevar estadísticas). Implementados correctamente, los métodos GET, HEAD, PUT y DELETE son idempotentes, pero no el método POST. Todos los métodos seguros también son idempotentes.

1.5.5 HTTP response status codes

Los códigos de estado de respuesta HTTP indican si una solicitud HTTP específica se completó con éxito. Las respuestas se agrupan en cinco clases:

1. Respuestas informativas (100–199)
2. Respuestas exitosas (200–299)
3. Mensajes de redirección (300–399)
4. Respuestas de error del cliente (400–499)
5. Respuestas de error del servidor (500–599)

Respuestas informativas

- 100 Continue: indica que todo está bien hasta el momento y que el cliente debe continuar con la solicitud o ignorarla si ya finalizó.
- 101 Protocolos de conmutación: este código se envía en respuesta a un encabezado de solicitud de actualización del cliente e indica el protocolo al que se está cambiando el servidor.
- 102 Procesamiento (WebDAV): Este código indica que el servidor ha recibido y está procesando la solicitud, pero aún no hay respuesta disponible.

1.5.6 URL - URI

La «Uniform Resource Locator» (URL o Localizadora Uniforme de Recursos en Español) es una cadena de texto que especifica dónde se puede encontrar un recurso (como una página web, una imagen o un video) en Internet.

En el contexto de HTTP, las URLs se denominan "dirección web.º .enlace". Tu navegador muestra las URLs en su barra de direcciones, por ejemplo: https://developer.mozilla.org — Algunos navegadores muestran solo la parte de una URL después de //", es decir, el Nombre de dominio.

Las URLs también se pueden utilizar para la transferencia de archivos (FTP), correos electrónicos (SMTP) y otras aplicaciones.

Un URI (Identificador Uniforme de Recursos de sus siglas en inglés: Uniform Resource Identifier) es una cadena que se refiere a un recurso. Los más comunes son URLs, que identifican el recurso dando su ubicación en la Web. URNs (en-US), por el contrario, se refiere a un recurso por un nombre, en un espacio de nombres determinados, como el ISBN(International Standard Book Number) de un libro.

`https://developer.mozilla.org/es/docs/Learn/Common_questions/What_is_a_URL`

1.6 Modelo de Objetos de Documento - DOM

El DOM (Document Object Model en español Modelo de Objetos del Documento) es una API definida para representar e interactuar con cualquier documento HTML o XML. El DOM es un modelo de documento que se carga en el navegador web y que representa el documento como un árbol de nodos, en donde cada nodo representa una parte del documento (puede tratarse de un elemento, una cadena de texto o un comentario).

> R Una Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) define un conjunto de directivas que pueden ser usadas para tener una pieza de software funcionando con algunas otras.

El DOM es una de las APIs más usadas en la Web, pues permite ejecutar código en el navegador para acceder e interactuar con cualquier nodo del documento. Estos nodos pueden crearse, moverse o modificarse. Pueden añadirse a estos nodos manejadores de eventos (event listeners en inglés) que se ejecutarán/activarán cuando ocurra el evento indicado en este manejador.

El DOM surgió a partir de la implementación de JavaScript en los navegadores. A esta primera versión también se la conoce como DOM 0 o "Legacy DOM". Hoy en día el grupo WHATWG es el encargado de actualizar el estándar de DOM.

1.7 REST

El término "Transferencia de Estado Representacional"(REST) representa un conjunto de características de diseño de arquitecturas software que aportan confiabilidad, eficiencia y escalibilidad a los sistemas distribuidos. Un sistema es llamado RESTful cuando se ajusta a estas características.

La idea básica de REST es que un recurso, e.j. un documento, es transferido con su estado y su relaciones (hipertexto) mediante formatos y operaciones estandarizadas bien definidas.

Como HTTP, el protocolo estandar de la Web, también transfiere documentos e hipertexto, las APIs HTTP a veces son llamadas APIs RESTful, servicios RESTful, o simplemente servicios REST, aunque no se ajusten del todo a la deficinición de REST. Los principiantes pueden pensar que una API REST es un servicio HTTP que puede ser llamado mediante librerias y herramientas web estandar.

Una API de REST, o API de RESTful, es una interfaz de programación de aplicaciones (API o API web) que se ajusta a los límites de la arquitectura REST y permite la interacción con los servicios web de RESTful. El informático Roy Fielding es el creador de la transferencia de estado representacional (REST).

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones.Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).Por ejemplo, el diseño de una API de servicio meteorológico podría requerir que el usuario escribiera un código postal y que el productor diera una respuesta en dos partes: la primera sería la temperatura máxima y la segunda, la mínima.

En otras palabras, las API le permiten interactuar con una computadora o un sistema para obtener datos o ejecutar una función, de manera que el sistema comprenda la solicitud y la cumpla.

Imagínelas como si fueran los mediadores entre los usuarios o clientes y los recursos o servicios web que quieren obtener. Con ellas, las empresas pueden compartir recursos e información mientras conservan la seguridad, el control y la autenticación, lo cual les permite determinar el contenido al que puede acceder cada usuario.

Otra ventaja de las API es que usted no necesita saber cómo se recibe el recurso ni de dónde proviene.

REST no es un protocolo ni un estándar, sino más bien un conjunto de límites de arquitectura. Los desarrolladores de las API pueden implementarlo de distintas maneras.

Cuando el cliente envía una solicitud a través de una API de RESTful, esta transfiere una representación del estado del recurso requerido a quien lo haya solicitado o al extremo. La información se entrega por medio de HTTP en uno de estos formatos: JSON (JavaScript Object Notation), HTML, XLT, Python, PHP o texto sin formato. JSON es el lenguaje de programación más popular, ya que tanto las máquinas como las personas lo pueden comprender y no depende de ningún lenguaje, a pesar de que su nombre indique lo contrario.

También es necesario tener en cuenta otros aspectos. Los encabezados y los parámetros también son importantes en los métodos HTTP de una solicitud HTTP de la API de RESTful, ya que contienen información de identificación importante con respecto a los metadatos, la autorización, el identificador uniforme de recursos (URI), el almacenamiento en caché, las cookies y otros elementos de la solicitud. Hay encabezados de solicitud y de respuesta, pero cada uno tiene sus propios códigos de estado e información de conexión HTTP.

Para que una API se considere de RESTful, debe cumplir los siguientes criterios:

Arquitectura cliente-servidor compuesta de clientes, servidores y recursos, con la gestión de solicitudes a través de HTTP. Comunicación entre el cliente y el servidor sin estado, lo cual implica que no se almacena la información del cliente entre las solicitudes de GET y que cada una de ellas es independiente y está desconectada del resto. Datos que pueden almacenarse en caché y optimizan las interacciones entre el cliente y el servidor. Una interfaz uniforme entre los

elementos, para que la información se transfiera de forma estandarizada. Para ello deben cumplirse las siguientes condiciones: Los recursos solicitados deben ser identificables e independientes de las representaciones enviadas al cliente. El cliente debe poder manipular los recursos a través de la representación que recibe, ya que esta contiene suficiente información para permitirlo. Los mensajes autodescriptivos que se envíen al cliente deben contener la información necesaria para describir cómo debe procesarla. Debe contener hipertexto o hipermedios, lo cual significa que cuando el cliente acceda a algún recurso, debe poder utilizar hipervínculos para buscar las demás acciones que se encuentren disponibles en ese momento. Un sistema en capas que organiza en jerarquías invisibles para el cliente cada uno de los servidores (los encargados de la seguridad, del equilibrio de carga, etc.) que participan en la recuperación de la información solicitada. Código disponible según se solicite (opcional), es decir, la capacidad para enviar códigos ejecutables del servidor al cliente cuando se requiera, lo cual amplía las funciones del cliente. Si bien la API de REST debe cumplir todos estos parámetros, resulta más fácil de usar que un protocolo definido previamente, como SOAP (protocolo simple de acceso a objetos), el cual tiene requisitos específicos, como la mensajería XML y la seguridad y el cumplimiento integrados de las operaciones, que lo hacen más lento y pesado.

Por el contrario, REST es un conjunto de pautas que pueden implementarse según sea necesario. Por esta razón, las API de REST son más rápidas y ligeras, cuentan con mayor capacidad de ajuste y, por ende, resultan ideales para el Internet de las cosas (IoT) y el desarrollo de aplicaciones para dispositivos móviles.

`https://www.redhat.com/es/topics/api/what-is-a-rest-api`

1.8 Gestor de contenidos

[2]Un sistema de gestión de contenidos o CMS es un programa informático que permite a los usuarios publicar, organizar, cambiar o eliminar diferentes tipos de contenido como texto, imágenes incrustadas, vídeo, audio y código interactivo.

[4]Cuenta con una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio web. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio web sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación en el sitio a varios editores. Un ejemplo clásico es el de editores que cargan el contenido al sistema y otro de nivel superior (moderador o administrador) que permite que estos contenidos sean visibles a todo el público (los aprueba).

1.8.1 Características de un CMS

Los sistemas de gestión de contenidos se definen por las siguientes particularidades, muchas de las cuales son, a su vez, grandes ventajas:

- Uso intuitivo y fácil para simplificar la edición y publicación de contenidos. No se requieren conocimientos de programación.
- Configuración flexible y personalizada a través de múltiples opciones.
- Velocidad y rendimiento elevados gracias a su excelente capacidad para el desarrollo de tareas.
- Seguridad presente gracias a opciones como aprobación de contenido, verificación de correo electrónico, historial de login o registro de auditoría, entre otras.
- Medios de soporte para ayudar a los usuarios a la resolución de dudas y problemas.

Blogs:
- b2evolution
- Blogger
- Movable Type
- Nucleus CMS
- Rapid CMS
- Simple PHP Blog
- Textpattern
- WordPress

Foros:
- miniBB
- MyBB
- phpBB
- punBB
- Simple Machines Forum

Galerías:
- Coppermine
- Gallery
- Gallery 2
- rapid_CMS

Wikis:
- Dokuwiki
- MediaWiki
- Pmwiki
- TiddlyWiki
- WikkaWiki
- Desired
- Moinmoin
- Gitit
- hatta Wiki
- Ikiwiki
- Tikiwiki
- FOSwiki
- XWiki

Educación:
- Chamilo
- Claroline
- Mahara
- Moodle

Comercio:
- Kentico CMS
- Magento
- OpenCart
- osCommerce
- PrestaShop
- Shopify
- Zen Cart

Almacenamiento de archivos:
- Dataprius
- ownCloud
- NextCloud

Portales:
- Apache Lenya
- ASPInvision
- Comitium Suite
- Content-SORT
- Drupal
- Envolution
- Jaws
- Joomla!
- Kentico CMS
- Liferay
- Mambo
- myphpnuke

Figura 1.13: Algunos sistemas gestores de contenido

Figura 1.14: Otros sistemas gestores de contenido populares

1.9 Entorno de desarrollo

nodejs https://github.com/nodesource/distributions/blob/master/README.md

Angular: npm install -g @angular/cli

Set-ExecutionPolicy RemoteSigned -Scope CurrentUser get-ExecutionPolicy -list

1.10 Backend

1.10.1 Postman - Thunder client - curl

1.10.2 Frameworks

1.10.3 Java

1.10.4 Spring boot

1.10.5 Spring data

1.10.6 Postgres - Mongo - Elastic Search

1.10.7 CRUD

Entidades - DTOs

Repositorios

Servicios

Controladores

1.11 Frontend

1.11.1 Frameworks

[4]Un entorno de trabajo (del inglés framework), o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar. En el desarrollo de software, un entorno de trabajo es una estructura conceptual y tecnológica de asistencia definida, normalmente, con artefactos o módulos concretos de software, que puede servir de base para la organización y desarrollo de software. Típicamente, puede incluir soporte de programas, bibliotecas, y un lenguaje interpretado, entre otras herramientas, para así ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio, y provee una estructura y especial metodología de trabajo, la cual extiende o utiliza las aplicaciones del dominio.

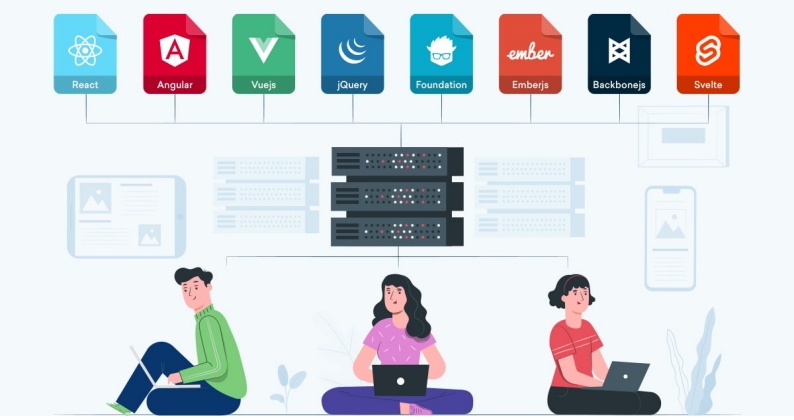

Figura 1.15: Algunos frameworks para frontend

1.11.2 HTML

1.11.3 CSS

[2]CSS (Cascading Style Sheets - en español Hojas de Estilo en Cascadas) es usado para darle estilo y diseño a las páginas Web — por ejemplo, para cambiar la fuente de letra, color, tamaño y el espaciado de tu contenido; dividir en múltiples columnas, o agregar animaciones y otras propiedades decorativas. Este modulo provee un inicio suave para tu ruta de aprendizaje hacia el

dominio de CSS con su funcionamiento básico, como luce su sintaxis, y cómo puedes comenzar a utilizarlo y añadir estilo a HTML.

El CSS se puede usar para estilos de texto muy básicos como, por ejemplo, cambiar el color y el tamaño de los encabezados y los enlaces. Se puede utilizar para crear un diseño, como podría ser convertir una columna de texto en una composición con un área de contenido principal y una barra lateral para información relacionada. Incluso se puede usar para crear efectos de animación.

1.11.4 Sintaxis del CSS

El CSS es un lenguaje basado en reglas: cada usuario define las reglas que especifican los grupos de estilos que van a aplicarse a elementos particulares o grupos de elementos de la página web. Por ejemplo: «Quiero que el encabezado principal de mi página se muestre en letras grandes de color rojo».

El código siguiente muestra una regla CSS muy simple que proporcionaría el estilo descrito en el párrafo anterior:

```
1   h1 {
2      color: red;
3      font-size: 5em;
4   }
```

La regla se abre con un selector (en-US). Este selecciona el elemento HTML que vamos a diseñar. En este caso, diseñaremos encabezados de nivel uno (<h1>(en-US)).

Luego tenemos un conjunto de llaves . Entre estas habrá una o más declaraciones, que tomarán la forma de pares de propiedad y valor. Cada par especifica cada una de las propiedades de los elementos seleccionados y el valor que queremos dar a esa propiedad.

Antes de los dos puntos, tenemos la propiedad; y después, el valor. Las propiedades (en-US) CSS admiten diferentes valores, dependiendo de qué propiedad se esté especificando. En el ejemplo anterior, tenemos la propiedad color, que puede tomar varios valores de color. También tenemos la propiedad de font-size, que puede tomar varias unidades de tamaño como valor.

Una hoja de estilo CSS contendrá muchas de estas reglas, escritas una tras otra.

```
1 h1 {
2    color: red;
3    font-size: 5em;
4 }
5
6 p {
7    color: black;
8 }
```

Algunos valores se aprenden rápidamente, mientras que otros deberán buscarse. Las páginas de propiedades individuales que hay en el proyecto MDN proporcionan una forma rápida de buscar propiedades y sus valores en caso de olvidarlos o desear saber qué más se puede usar como valor.

Nota: Puedes encontrar enlaces a todas las páginas de las propiedades CSS (junto con otras características CSS) enumeradas en la referencia CSS del proyecto MDN. Alternativamente, deberías acostumbrarte a buscar «mdn css-feature-name» en tu motor de búsqueda favorito siempre que necesites obtener más información sobre una función CSS. Por ejemplo, intenta buscar «mdn color» y «mdn font-size».

`https://developer.mozilla.org/es/docs/Web/CSS/Reference`

1.11.5 Especificaciones CSS

Todas las tecnologías de estándares web (HTML, CSS, JavaScript, etc.) se definen en extensos documentos denominados especificaciones, publicados por organizaciones de estándares (como W3C (en-US), WHATWG, ECMA (en-US) o Khronos (en-US)) que definen con precisión cómo se supone que deben comportarse esas tecnologías.

El caso de CSS no es diferente: lo desarrolla un grupo del W3C llamado CSS Working Group. Este grupo está compuesto por representantes de proveedores de navegadores y otras compañías interesadas en CSS. También hay otras personas, conocidas como expertos invitados, que actúan como voces independientes y no están vinculados a ninguna organización.

1.11.6 Agregar CSS a un documento

Lo primero que se debe hacer es decirle al documento HTML que hay algunas reglas CSS que queremos que use. Hay tres formas diferentes de aplicar CSS a un documento HTML, sin embargo, por ahora, veremos la forma más habitual y útil de hacerlo: vincular el CSS desde el encabezado del documento.

Crea un archivo en la misma carpeta que tu documento HTML y guárdalo como styles.css. La extensión .css muestra que es un archivo CSS.

Para vincular styles.css a index.html, añade la siguiente línea en algún lugar dentro del <head>del documento HTML:

```
<link rel="stylesheet" href="styles.css">
```

Este elemento <link>le dice al navegador que hay una hoja de estilo con el atributo rel y la ubicación de esa hoja de estilo como el valor del atributo href

R El atributo rel indica la relación del documento enlazado con el actual.

Puedes determinar múltiples selectores a la vez, separándolos con una coma. Si queremos que todos los párrafos y todos los elementos de la lista sean verdes, el código se verá así:

```
p, li {
   color: green;
}
```

1.11.7 Añadir una clase

Hasta ahora, hemos utilizado elementos cuyo nombre se basa en el nombre de elemento que reciben en HTML. Esto funciona siempre que se desee que todos los elementos de ese tipo tengan el mismo aspecto en el documento. La mayoría de las veces no es el caso, por lo que deberás encontrar una manera de seleccionar un subconjunto de los elementos sin que cambien los demás. La forma más común de hacer esto es añadir una clase al elemento HTML y determinarla.
En tu documento HTML, añade al segundo elemento de la lista un atributo de clase. Debería verse así:

```
<ul>
<li>Punto uno</li>
<li class = "special">Punto dos</li>
<li>Punto <em>tres</em></li>
</ul>
```

En tu CSS, puedes seleccionar una clase special creando un selector que comience con un carácter de punto final. Añade lo siguiente a tu archivo CSS:

```
.special {
   color: orange;
   font-weight: bold;
}
```

Puedes aplicar la clase special a cualquier elemento de la página que desees que tenga el mismo aspecto que este elemento de lista. Por ejemplo, es posible que desees que el del párrafo también sea naranja y en negrita.

A veces verás reglas con un selector que enumera el selector de elementos HTML junto con la clase:

```
li.special {
   color: orange;
   font-weight: bold;
}
```

Esta sintaxis significa «determina cualquier elemento li que tenga una clase special». Si hicieras esto, ya no podrías aplicar la clase a un elemento u otro elemento simplemente añadiéndole la clase; tendrías que añadir ese elemento a la lista de selectores:

```
li.special,
span.special {
   color: orange;
   font-weight: bold;
}
```

1.11.8 Dar formato según la ubicación en un documento

Hay momentos en los que querrás que algo se vea diferente en función de dónde esté en el documento. Hay múltiples selectores que pueden hacerlo, pero por ahora veremos solo un par. En nuestro documento hay dos elementos : uno dentro de un párrafo y el otro dentro de un elemento de la lista. Para seleccionar solo un que esté anidado dentro de un elemento <li>, podemos usar un selector llamado combinador descendente, que simplemente toma la forma de un espacio entre otros dos selectores.

R Combinador Css de descendiente: (espacio en blanco) o

```
li em {
   color: rebeccapurple;
}
```

Este selector separará cualquier elemento que esté dentro de (un descendiente de) . Entonces, en tu documento de ejemplo, deberías encontrar que el del tercer elemento de la lista es morado, pero el que hay en el párrafo no ha cambiado.

Otra cosa que puedes probar es dar formato un párrafo que venga directamente a continuación de un título que esté en el mismo nivel de jerarquía en el HTML. Para hacerlo, coloca un + (un combinador hermano adyacente) entre los selectores.

```
h1 + p {
   font-size: 200%;
```

```
}
```

```
<h1>I am a level one heading</h1>

<p>This is a paragraph of text. In the text is a <span>span element</span>
and also a <a href="http://example.com">link</a>.</p>

<p>This is the second paragraph. It contains an <em>emphasized</em> element.</p>

<ul>
<li>Item <span>one</span></li>
<li>Item two</li>
<li>Item <em>three</em></li>
</ul>
```

Salida:

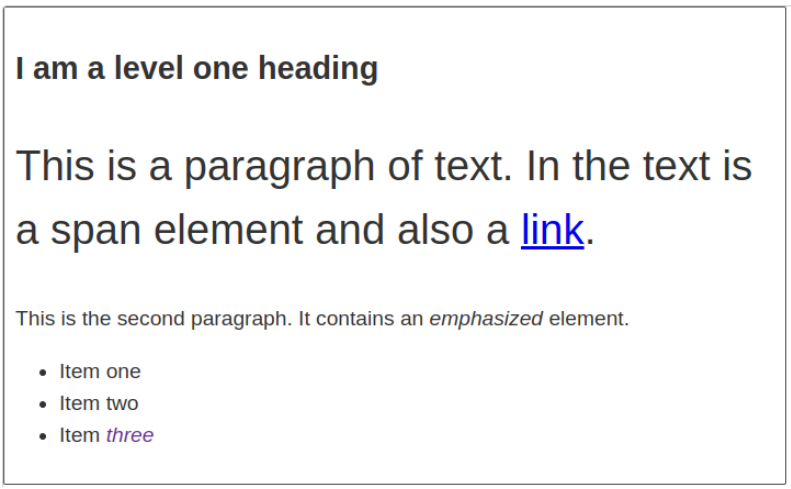

Figura 1.16: Salida de formato según la ubicación

1.11.9 Dar formato según el estado

Un ejemplo sencillo es el estilo de los enlaces. Cuando damos formato a un enlace, necesitamos seleccionar el elemento <a>(anclaje). Tiene diferentes estados dependiendo de si se ha visitado o no, se pasa por encima, o se presiona con el teclado o se hace clic (se activa). Puedes usar CSS para dar formato a estos diferentes estados. El CSS que encontrarás a continuación presenta en color rosa los enlaces que no se han visitado y en verde los que sí.

```
a:link {
   color: pink;
}

a:visited {
   color: green;
}
```

Puedes cambiar la apariencia del enlace, por ejemplo, eliminando el subrayado, lo que se logra mediante la siguiente regla:

```
   a:hover {
```

```
    text-decoration: none;
  }
```

1.11.10 Estilos en línea

Los estilos en línea son declaraciones CSS que afectan a un solo elemento, contenido dentro de un atributo de style:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Mi experimento CSS</title>
</head>
<body>
<h1 style="color: blue;background-color: yellow;border: 1px solid black;">Hola
    mundo!</h1>
<p style="color:red;">Este es mi primer ejemplo de CSS</p>
</body>
</html>~
```

R ¡No hagas esto a menos que realmente tengas que hacerlo! Es realmente malo a la hora de realizar el mantenimiento

1.11.11 Selectores

No se puede hablar de CSS sin mencionar los selectores, de los cuales ya hemos descubierto varios tipos diferentes en la lección Empezar con el CSS. Un selector es, como determinamos, un elemento de nuestro documento HTML para aplicarle estilo. Si los estilos no se aplican correctamente, es probable que el selector no coincida con lo que crees que debería coincidir.

Cada regla CSS comienza con un selector o una lista de selectores que indican al navegador a qué elemento o elementos deben aplicarse dichas reglas. Todos los siguientes son ejemplos de selectores válidos o listas de selectores.

```
h1
a:link
.manythings
#onething
*
.box p
.box p:first-child
h1, h2, .intro
```

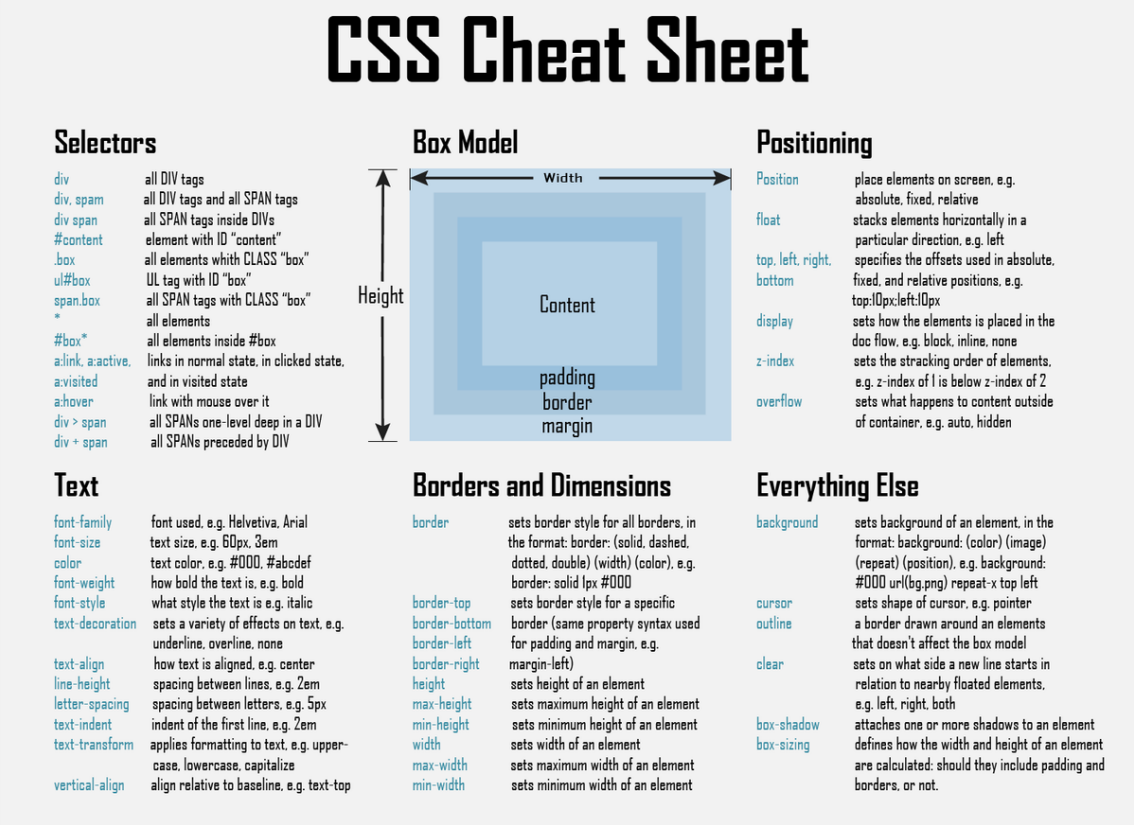

Figura 1.17: Hoja de trucos CSS

1.11.12 Especificidad

A menudo habrá situaciones en las que dos selectores podrían determinar un mismo elemento HTML. Considera la siguiente hoja de estilo, en que definimos una regla con un selector p que establecerá los párrafos en color azul, y también una clase que establecerá los elementos seleccionados en color rojo.

```
.special {
   color: red;
}

p {
   color: blue;
}
```

Digamos que en nuestro documento HTML hay un párrafo con una clase special. Ambas reglas podrían aplicarse. ¿Cuál ganará? ¿De qué color crees que será nuestro párrafo?

```
<p class="special">De  qu  color soy?</p>
```

El lenguaje CSS tiene reglas para controlar cuál ganará en caso de colisión; reciben el nombre de cascada y especificidad. En el siguiente bloque de códigos hemos definido dos reglas para el selector p, pero el párrafo termina siendo de color azul. Esto se debe a que la declaración que lo establece en azul aparece más abajo en la hoja de estilo, y los estilos posteriores anulan a los anteriores. Así funciona la regla de la cascada.

1.11.13 @rules

Las @rules (leído .ªt-rules.en inglés) dan al CSS algunas instrucciones sobre cómo comportarse. Algunas @rules son simples, con el nombre de la regla y un valor. Por ejemplo, para importar una hoja de estilo adicional a tu hoja de estilo CSS principal, puedes usar @import:

```
@import 'styles2.css';
```

Una de las @rules más comunes con las que te encontrarás es @media, que permite usar consultas a medios para aplicar CSS solo cuando se dan ciertas condiciones (por ejemplo, cuando la resolución de la pantalla supera un valor determinado o la anchura supera un valor concreto).

En el CSS que se muestra a continuación, tenemos una hoja de estilo que le da al elemento <body>un color de fondo rosado. Sin embargo, luego usamos @media para crear una sección de nuestra hoja de estilo que solo se aplicará en los navegadores con una ventana gráfica de más de 30em de ancho. Si el navegador es más ancho que 30em, el color de fondo será azul.

```
body {
   background-color: pink;
}

@media (min-width: 30em) {
   body {
      background-color: blue;
   }
}
```

1.11.14 Abreviaturas

Algunas propiedades como font, background, padding, border y margin se llaman propiedades abreviadas. Esto se debe a que permiten establecer varios valores de propiedad en una sola línea, lo que ahorra tiempo y ordena el código.

Por ejemplo, esta línea:

```
/* En propiedades abreviadas con 4 valores, como margin y padding (relleno), los
    valores se aplican
 segn  el orden: arriba, derecha, abajo e izquierda (en sentido horario desde la
    parte superior). Tambin hay otros
tipos de abreviaturas, como las propiedades abreviadas con 2 valores que
    establecen el relleno/margen,
arriba/abajo, y luego izquierda/derecha */
padding: 10px 15px 15px 5px;
```

Hace lo mismo que todas estas juntas:

```
padding-top: 10px;
padding-right: 15px;
padding-bottom: 15px;
padding-left: 5px;
```

1.11.15 ¿Cómo funciona realmente el CSS?

Cuando un navegador muestra un documento, ha de combinar el contenido con la información de estilo del documento. Procesa el documento en una serie de etapas, que enumeraremos a continuación. Ten en cuenta que este es un modelo muy simplificado de lo que sucede cuando un

navegador carga una página web y que cada navegador gestiona el proceso de manera diferente. Pero esto es más o menos lo que sucede.

- El navegador carga el HTML (por ejemplo, lo recibe de la red).
- Convierte el HTML en un DOM (Modelo de objetos del documento). El DOM representa el documento en la memoria del ordenador. Lo explicaremos más detalladamente en la sección siguiente.
- Entonces, el navegador va a buscar la mayor parte de los recursos vinculados al documento HTML, como las imágenes y los videos incrustados... ¡y también el CSS vinculado! JavaScript aparece un poco más adelante en el proceso, pero no vamos a hablar de ello aún para evitar complicar las cosas.
- El navegador analiza el CSS y ordena en diferentes «cubos» las diferentes reglas según el tipo de selector. Por ejemplo, elemento, clase, ID, y así sucesivamente. Para cada tipo de selector que encuentre, calcula qué reglas deben aplicarse y a qué nodos en el DOM se les aplica el estilo según corresponda (este paso intermedio se llama árbol de renderización).
- El árbol de renderización presenta la estructura en que los nodos deben aparecer después de aplicarle las reglas.
- En la pantalla se muestra el aspecto visual de la página (esta etapa se llama pintura).

El siguiente diagrama ofrece una visión sencilla de este proceso.

Figura 1.18: Carga del css

Frameworks CSS

Tailwind CSS

1.11.16 Javascript - ECMAScript

1.11.17 Typescript

1.11.18 CRUD

Entidades - DTOs

Servicios

Componentes

Input - Output

1.11.19 Temas

1.11.20 Menús

1.12 Ramificación GIT

2. Parcial 2

Bibliography

Books

[Ser02] Mora Sergio Luján. *Programación de aplicaciones web: historia principios básicos y clientes web*. 1.ª edición. 2. Alicante: Editorial Club Universitario, oct. de 2002. ISBN: 84-8454-206-8 (véanse páginas 7-9).

Online

[MDN] mozilla.org / MDN Community. *Resources for Developers, by Developers*. URL: `https://developer.mozilla.org/es/` (véanse páginas 10, 12, 22, 24).

[She] Traductor Don Juan Sheldon. *¿Cuál es la diferencia entre modelo OSI y modelo TCP/IP?* URL: `https://community.fs.com/es/blog/tcpip-vs-osi-whats-the-difference-between-the-two-models.html`. (06.08.2021) (véase página 8).

[Wik] Wikipedia. *La enciclopedia de contenido libre*. URL: `https://es.wikipedia.org/` (véanse páginas 22, 24).

Articles

Índice alfabético