

# 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

<b>0.1</b>	<b>Sobre esta guía</b>	<b>5</b>
<b>1</b>	<b>Conceptos generales</b>	<b>7</b>
<b>1.1</b>	<b>El origen</b>	<b>7</b>
1.1.1	Protocolos de internet . . . . .	7
<b>1.2</b>	<b>Arquitecturas cliente/servidor</b>	<b>9</b>
1.2.1	Separación de funciones . . . . .	10
1.2.2	El cliente . . . . .	11
1.2.3	El servidor . . . . .	12
1.2.4	Transferencia de páginas web a través de http . . . . .	12
<b>1.3</b>	<b>HTTP/HTTPS</b>	<b>13</b>
1.3.1	Características clave del protocolo HTTP . . . . .	14
1.3.2	¿Qué se puede controlar con HTTP? . . . . .	14
1.3.3	Mensajes HTTP . . . . .	15
1.3.4	Métodos de petición HTTP . . . . .	17
1.3.5	HTTP response status codes . . . . .	19
1.3.6	URL - URI . . . . .	21
<b>1.4</b>	<b>Modelo de Objetos de Documento - DOM</b>	<b>22</b>
<b>1.5</b>	<b>REST</b>	<b>23</b>
<b>1.6</b>	<b>Gestor de contenidos</b>	<b>24</b>
1.6.1	Características de un CMS . . . . .	24
<b>1.7</b>	<b>HTML</b>	<b>25</b>
1.7.1	Anatomía de un elemento HTML . . . . .	25
1.7.2	Elementos anidados . . . . .	26
1.7.3	Elementos de bloque y elementos en línea . . . . .	26
1.7.4	Elementos vacíos . . . . .	27
1.7.5	Atributos . . . . .	27

1.7.6	Anatomía de un documento HTML .....	27
<b>1.8</b>	<b>CSS</b>	<b>28</b>
1.8.1	Sintaxis del CSS .....	28
1.8.2	Especificaciones CSS .....	29
1.8.3	Agregar CSS a un documento .....	29
1.8.4	Añadir una clase .....	30
1.8.5	Dar formato según la ubicación en un documento .....	31
1.8.6	Dar formato según el estado .....	32
1.8.7	Estilos en línea .....	32
1.8.8	Selectores .....	33
1.8.9	Especificidad .....	34
1.8.10	@rules .....	35
1.8.11	Abreviaturas .....	35
1.8.12	¿Cómo funciona realmente el CSS? .....	35
1.8.13	Normal flow .....	36
1.8.14	Contenido desbordante .....	38
1.8.15	Frameworks CSS .....	40
<b>1.9</b>	<b>Javascript</b>	<b>41</b>
1.9.1	ECMAScript .....	41
1.9.2	Javascript .....	41
1.9.3	¿Cómo agregas JavaScript a tu página? .....	42
1.9.4	Un poco de sintaxis .....	43
<b>1.10</b>	<b>TypeScript</b>	<b>44</b>
1.10.1	TypeScript: un verificador de tipo estático .....	45
1.10.2	Un superconjunto tipado de JavaScript .....	45
1.10.3	Tipos .....	45
1.10.4	Frameworks .....	49
<b>1.11</b>	<b>Estructura de datos</b>	<b>50</b>
1.11.1	JSON .....	50
1.11.2	XML .....	50
1.11.3	DTO .....	50
<b>1.12</b>	<b>Localización e Internacionalización</b>	<b>50</b>
1.12.1	Localización .....	50
1.12.2	Internacionalización .....	51
	<b>Bibliography .....</b>	<b>53</b>
	<b>Books</b>	<b>53</b>
	<b>Online</b>	<b>53</b>
	<b>Articles</b>	<b>53</b>
	<b>Index .....</b>	<b>55</b>

## 0.1 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: <a href="mailto:fheredia@yavirac.edu.ec">fheredia@yavirac.edu.ec</a>



# 1. Conceptos generales

## 1.1 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 .<sup>a</sup>plicaciones 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.

 [https://rua.ua.es/dspace/bitstream/10045/16995/1/sergio\\_lujan-programacion\\_de\\_aplicaciones\\_web.pdf](https://rua.ua.es/dspace/bitstream/10045/16995/1/sergio_lujan-programacion_de_aplicaciones_web.pdf)

### 1.1.1 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.

[7]El protocolo de control de transmisión (TCP) es, al igual que el protocolo UDP como el SCTP, un protocolo de Internet que está ubicado en la capa de transporte del modelo OSI. El objetivo del protocolo TCP es crear conexiones dentro de una red de datos compuesta por redes de computadoras para intercambiar datos. Además, en cuanto a su funcionamiento, garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.

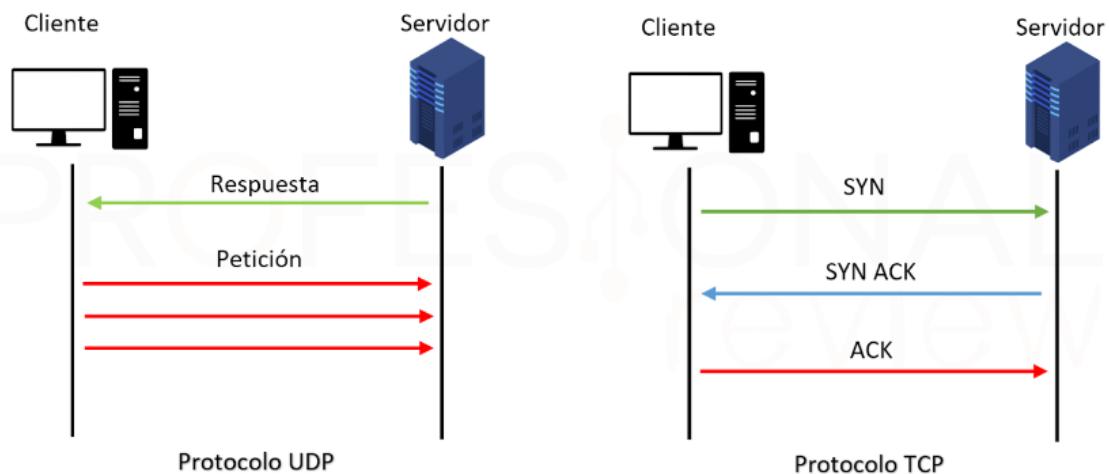


Figura 1.1: Comparativa TCP - UDP

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

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

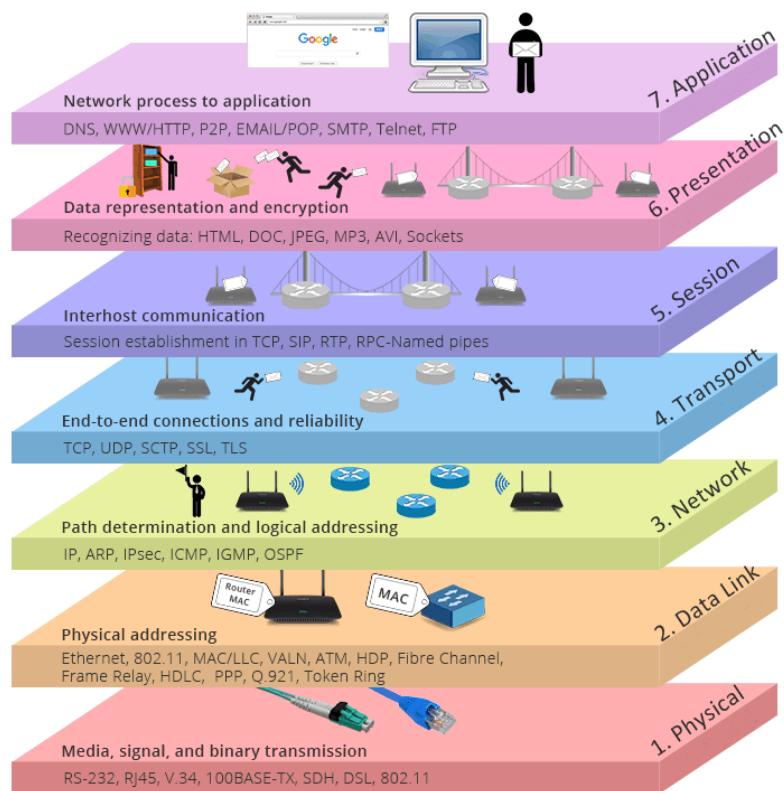


Figura 1.2: Modelo OSI



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

## 1.2 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), tránsito 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.

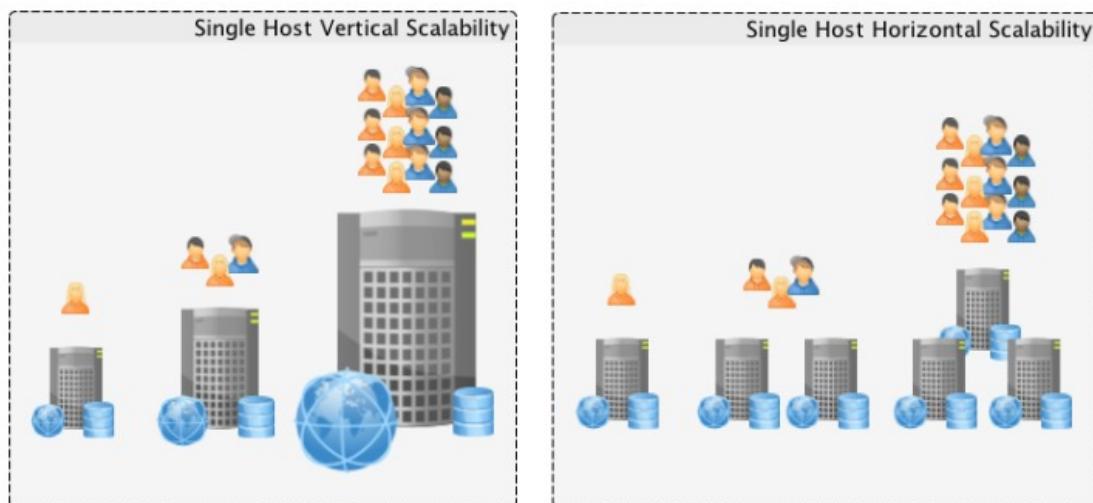


Figura 1.3: Escalamiento horizontal vs vertical

### 1.2.1 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.

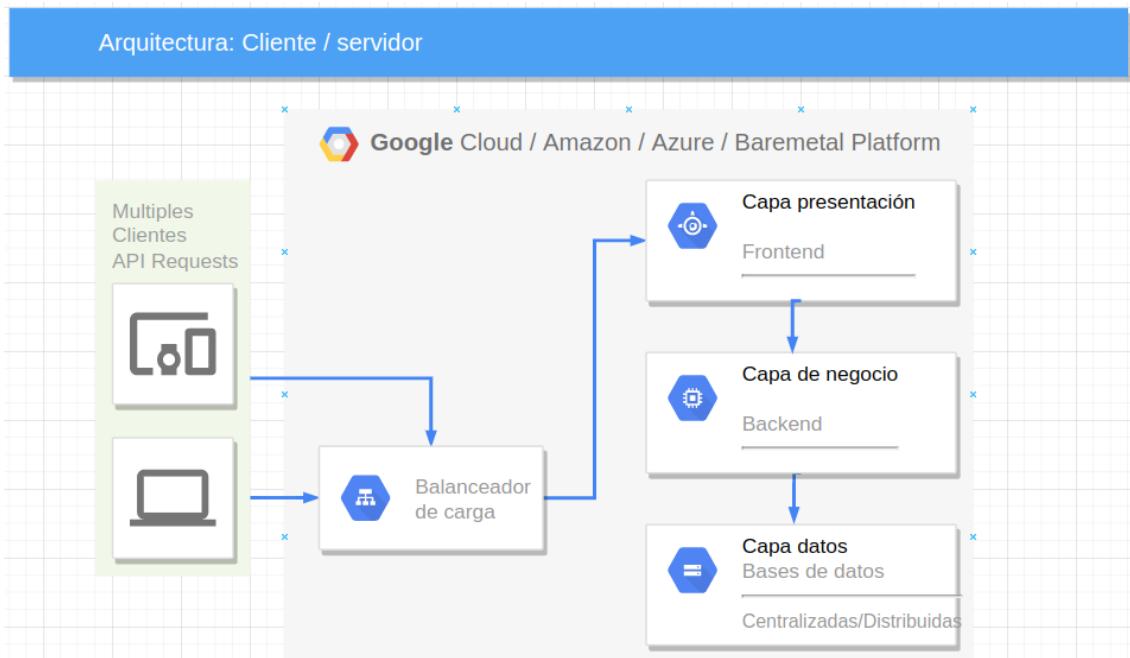


Figura 1.4: Arquitectura cliente/servidor

### 1.2.2 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

**R** Cada tecnología puede tener sus variantes.

WebAssembly es un proyecto prometedor.

Algunas tecnologías ya están en desuso (Applets, Flash entre otras.)



Figura 1.5: Cliente

### 1.2.3 El servidor



Figura 1.6: Sala de servidores

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 única 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.

### 1.2.4 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.

### 1.3 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.

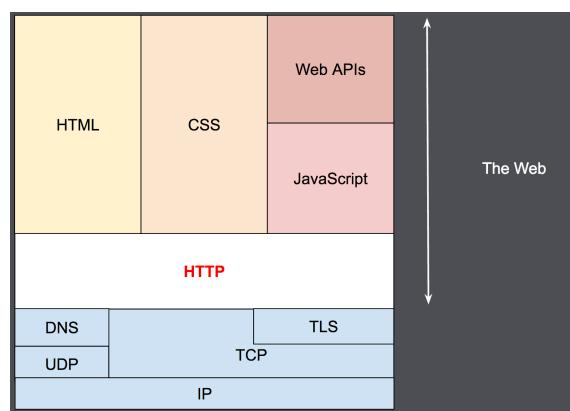


Figura 1.7: HTTP y capas

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 .

### 1.3.1 Características clave del protocolo HTTP

- **HTTP es sencillo:** HTTP está pensado y desarrollado para ser leído y fácilmente interpretado por las personas, haciendo de esta manera más fácil la depuración de errores, y reduciendo la curva de aprendizaje para las personas 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 misma 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 cestas de la compra.<sup>en</sup> 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 transporte, 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 experimentando con QUIC, que se apoya en el protocolo UDP y presenta mejoras en la fiabilidad y eficiencia de la comunicación.

### 1.3.2 ¿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 cuento 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, así que mediante cabeceras HTTP, se puede flexibilizar o relajar esta división entre cliente y servidor
- **Autenticación** 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 específica 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.

### 1.3.3 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-information 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 petición, mientras que su contenido en datos se conoce como el cuerpo del mensaje.

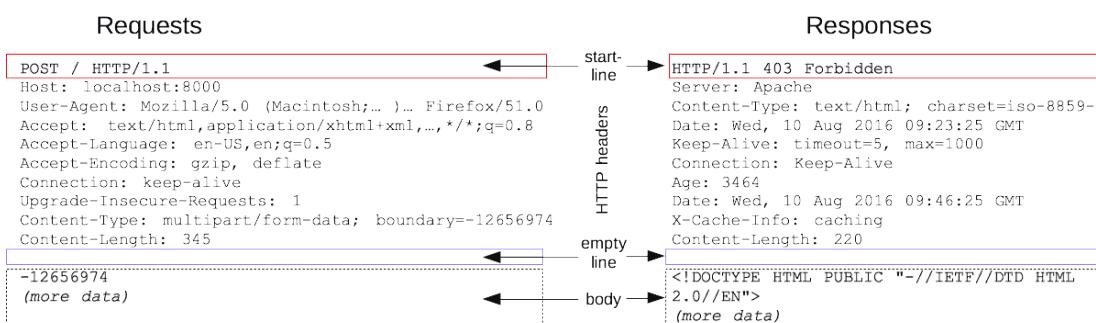


Figura 1.8: Mensajes

### Cabeceras

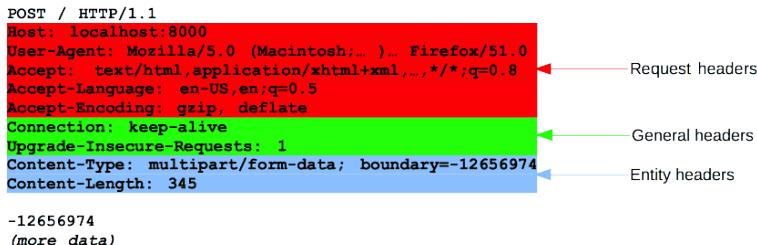
Las cabeceras HTTP de una petición siguen la misma estructura que la de una cabecera HTTP. Una cadena de caracteres, que no diferencia mayúsculas ni minúsculas, seguida por dos puntos (':') y un valor cuya estructura depende de la cabecera. La cabecera completa, incluido el valor, ha de ser formada en una única línea, y puede 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 inglés), 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).



```

POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh; ... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,...,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)

```

Figura 1.9: Cabeceras

## Cuerpo

La parte final de la petición es 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 categorías:

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

## Peticiones

Un ejemplo de petición HTTP:

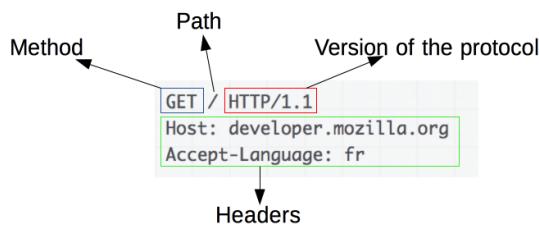


Figura 1.10: Petición

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.
- Un cuerpo de mensaje, en algún método, como puede ser POST, en el cual envía la información para el servidor.

### Respuestas

Un ejemplo de respuesta:

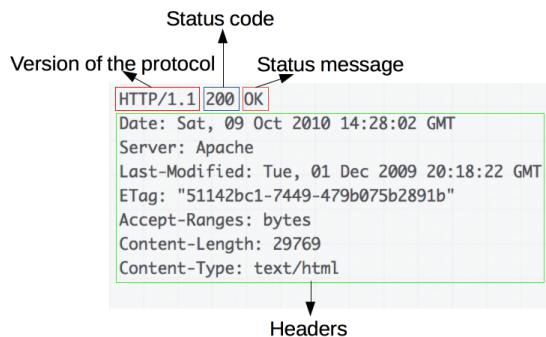


Figura 1.11: Respuesta

Las respuestas están formadas por los siguientes 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.

#### 1.3.4 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 sustitutivos, 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.

<u>Request has body</u>	No
<u>Successful response has body</u>	Yes
<u>Safe</u>	Yes
<u>Idempotent</u>	Yes
<u>Cacheable</u>	Yes
<u>Allowed in HTML forms</u>	Yes

Figura 1.12: 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.

<b>Request has body</b>	Yes
<b>Successful response has body</b>	Yes
<b>Safe</b>	No
<b>Idempotent</b>	No
<b>Cacheable</b>	Only if freshness information is included
<b>Allowed in <a href="#">HTML forms</a></b>	Yes

Figura 1.13: 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.

<b>Request has body</b>	Yes
<b>Successful response has body</b>	No
<b>Safe</b>	No
<b>Idempotent</b>	Yes
<b>Cacheable</b>	No
<b>Allowed in <a href="#">HTML forms</a></b>	No

Figura 1.14: PUT

## DELETE

El método DELETE borra un recurso en específico.

Request has body	May
Successful response has body	May
<a href="#">Safe</a>	No
<a href="#">Idempotent</a>	Yes
<a href="#">Cacheable</a>	No
Allowed in <a href="#">HTML forms</a>	No

Figura 1.15: 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
<a href="#">Safe</a>	No
<a href="#">Idempotent</a>	No
<a href="#">Cacheable</a>	No
Allowed in <a href="#">HTML forms</a>	No

Figura 1.16: 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.

**1.3.5 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.

### Respuestas satisfactorias

- 200 Ok: La solicitud ha tenido éxito. El significado de un éxito varía dependiendo del método HTTP.
- 201 Created: La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello. Ésta es típicamente la respuesta enviada después de una petición PUT.
- 202 Accepted: La solicitud se ha recibido, pero aún no se ha actuado. Es una petición "sin compromiso", lo que significa que no hay manera en HTTP que permita enviar una respuesta asíncrona que indique el resultado del procesamiento de la solicitud. Está pensado para los casos en que otro proceso o servidor maneja la solicitud, o para el procesamiento por lotes.

### Redirecciones

- 300 Multiple Choice: Esta solicitud tiene más de una posible respuesta. User-Agent o el usuario debe escoger uno de ellos. No hay forma estandarizada de seleccionar una de las respuestas.
- 301 Moved Permanently: Este código de respuesta significa que la URI del recurso solicitado ha sido cambiado. Probablemente una nueva URI sea devuelta en la respuesta.

### Errores de cliente

- 400 Bad Request: Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.
- 401 Unauthorized: Es necesario autenticar para obtener la respuesta solicitada. Esta es similar a 403, pero en este caso, la autenticación es posible.
- 403 Forbidden: El cliente no posee los permisos necesarios para cierto contenido, por lo que el servidor está rechazando otorgar una respuesta apropiada.
- 404 Not Found: El servidor no pudo encontrar el contenido solicitado. Este código de respuesta es uno de los más famosos dada su alta ocurrencia en la web.

### Errores de servidor

- 500 Internal Server Error: El servidor ha encontrado una situación que no sabe cómo manejarla.
- 501 Not Implemented: El método solicitado no está soportado por el servidor y no puede ser manejado.
- 502 Bad Gateway: Esta respuesta de error significa que el servidor, mientras trabaja como una puerta de enlace para obtener una respuesta necesaria para manejar la petición, obtuvo una respuesta inválida.
- 503 Service Unavailable: El servidor no está listo para manejar la petición. Causas comunes

puede ser que el servidor está caído por mantenimiento o está sobrecargado. Hay que tomar en cuenta que junto con esta respuesta, una página usuario-amigable explicando el problema debe ser enviada.

### 1.3.6 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.<sup>o</sup> 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.

Una URL está compuesta de diferentes partes, algunas obligatorias y otras opcionales. Veamos las partes más importantes usando la siguiente URL:

---

<sup>1</sup> <http://www.example.com:80/path/to/myfile.html?key1=value1&key2=value2#Somewhere>

---



Figura 1.17: Protocolo

http es el protocolo. La primera parte de la URL indica qué protocolo debe usar el navegador. Un protocolo es un método establecido para intercambiar o transferir datos alrededor de una red informática. Por lo general, para sitios web es el protocolo HTTP o su versión segura, HTTPS. La Web requiere uno de estos dos, pero los navegadores también saben cómo manejar otros protocolos como mailto: (para abrir un cliente de correo) o ftp: para manejar la transferencia de archivos, así que no se sorprenda si ve tales protocolos.



Figura 1.18: Dominio

www.example.com es el nombre de dominio o autoridad que gobierna el espacio de nombres. Indica cuando es solicitado el servidor Web



Figura 1.19: Puerto

:80 es el puerto en este caso. Indica la técnica **puerta** usada para acceder a los recursos en el servidor web. Usualmente es omitido si el servidor web usa los puertos estándares del protocolo HTTP (80 para HTTP y 443 para HTTPS) para permitir el acceso a sus recursos. De lo contrario, es obligatorio.



Figura 1.20: Puerto

/path/to/myfile.html es la ruta de acceso al recurso en el servidor Web. En los primeros días de la Web, una ruta como esta presentaba la ubicación física del archivo en el servidor Web. Hoy en día, es sobre todo una abstracción manejada por los servidores Web sin ningún tipo de realidad física.

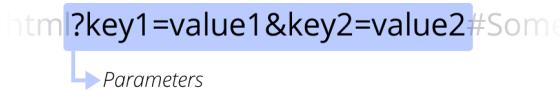


Figura 1.21: Parámetros

?key1=value1&key2=value2 son unos parámetros adicionales proporcionados al servidor Web. Esos parámetros son una lista de pares llave/valores separados por el símbolo &. El servidor Web puede utilizar estos parámetros para hacer cosas adicionales antes de retornar el recurso al usuario.

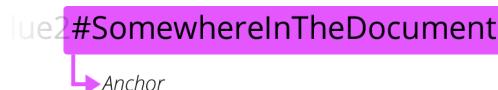


Figura 1.22: Ancla

#SomewhereInTheDocument es una referencia a otra parte del propio recurso. Esto representa una especie de "marcador" dentro del recurso, otorgandole al navegador las instrucciones para mostrar el contenido que se encuentra en esa referencia señalada.

## 1.4 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).



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.

## 1.5 REST

[2] El término "Transferencia de Estado Representacional"(REST) representa un conjunto de características de diseño de arquitecturas software que aportan confiabilidad, eficiencia y escalabilidad 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 definición de REST. Los principiantes pueden pensar que una API REST es un servicio HTTP que puede ser llamado mediante librerías y herramientas web estandar.

[redhat] 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.
- 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.

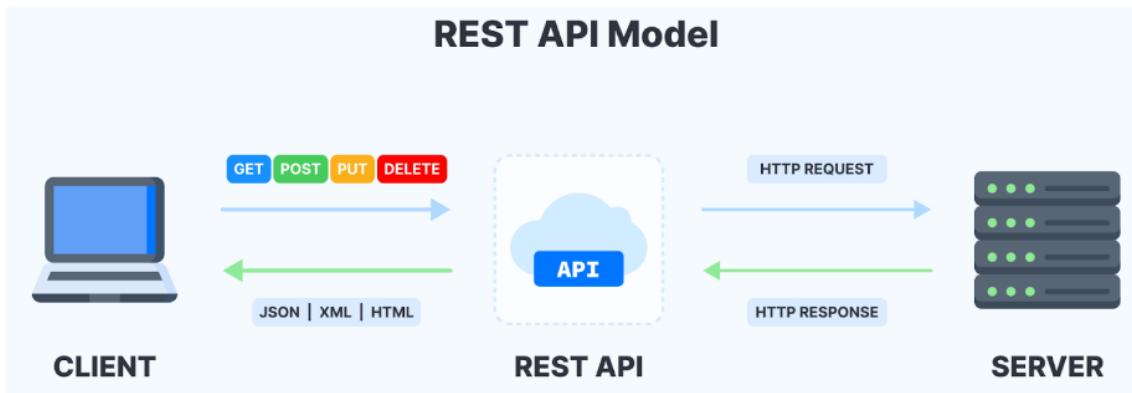


Figura 1.23: REST API Model

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.

## 1.6 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.

[7]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).

### 1.6.1 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:	Foros:	Galerías:	Wikis:	Educación:	Comercio:	Almacenamiento de archivos:	Portales:
• b2evolution	• miniBB	• Coppermine	• Dokumento	• Chamilo	• Kentico CMS	• Apache	
• Blogger	• MyBB	• Gallery	• MediaWiki	• Claroline	• Magento	• Lenya	
• Movable Type	• phpBB	• Gallery 2	• PmWiki	• Mahara	• OpenCart	• ASPInvision	
• Nucleus CMS	• punBB	• rapid_CMS	• TiddlyWiki	• Moodle	• osCommerce	• Comitium Suite	
• Rapid CMS	• Simple		• WikkaWiki		• PrestaShop		
• Simple PHP Blog	Machines Forum		• Desired		• Shopify	• Content-SORT	
• Textpattern			• Moinmoin		• Zen Cart	• Drupal	
• WordPress			• Gitit			• Envolution	
			• hatta Wiki			• Jaws	
			• Ikiwiki			• Joomla!	
			• Tikiwiki			• Kentico CMS	
			• FOSwiki			• Liferay	
			• XWiki			• Mambo	
						• myphnukre	

Figura 1.24: Algunos sistemas gestores de contenido



Figura 1.25: Otros sistemas gestores de contenido populares

## 1.7 HTML

[2] El Lenguaje de Marcado de Hipertexto (HTML) es el código que se utiliza para estructurar y desplegar una página web y sus contenidos. Por ejemplo, sus contenidos podrían ser párrafos, una lista con viñetas, o imágenes y tablas de datos

HTML no es un lenguaje de programación; es un lenguaje de marcado que define la estructura de tu contenido. HTML consiste en una serie de elementos que usarás para encerrar diferentes partes del contenido para que se vean o comporten de una determinada manera. Las etiquetas de encierre pueden hacer de una palabra o una imagen un hipervínculo a otro sitio, se pueden cambiar palabras a cursiva, agrandar o achicar la letra, etc. Por ejemplo, toma la siguiente línea de contenido:

---

1 Mi gato es muy gruon

---

Si quieras especificar que se trata de un párrafo, podrías encerrar el texto con la etiqueta de párrafo (<p>):

---

1 <p>Mi gato es muy gruon</p>

---

### 1.7.1 Anatomía de un elemento HTML

Explora este párrafo en mayor profundidad.

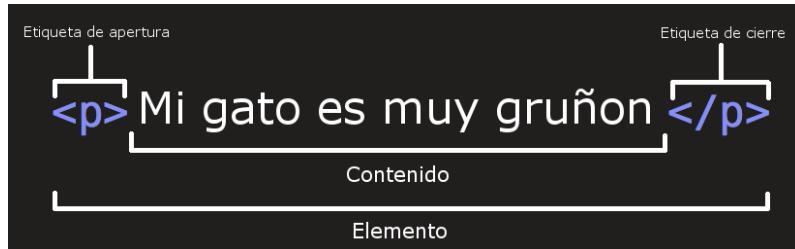


Figura 1.26: Estructura de un elemento html

Las partes principales del elemento son:

- La etiqueta de apertura: consiste en el nombre del elemento (en este caso, p), encerrado por paréntesis angulares (<>) de apertura y cierre. Establece dónde comienza o empieza a tener efecto el elemento —en este caso, dónde es el comienzo del párrafo—.
- El contenido: Este es el contenido del elemento. En este ejemplo, es el texto del párrafo.
- La etiqueta de cierre: Es lo mismo que la etiqueta de apertura, excepto que incluye una barra diagonal antes del nombre del elemento. Esto indica dónde termina el elemento; en este caso, dónde finaliza el párrafo. No incluir una etiqueta de cierre es un error común de principiante, y puede conducir a extraños resultados.

### 1.7.2 Elementos anidados

El elemento lo conforman la etiqueta de apertura, seguida del contenido, seguido de la etiqueta de cierre.

Se pueden poner elementos dentro de otros elementos. Esto se llama anidamiento. Si quisieramos decir que nuestro gato es muy gruñón, podríamos encerrar la palabra muy en un elemento **<strong>** para que aparezca destacada.

---

<sup>1</sup> `<p>Mi gato es <strong>muy</strong> grun.</p>`

---

Hay una forma correcta e incorrecta de anidar. En el ejemplo anterior, primero abrimos el elemento p, luego abrimos el elemento strong. Para un anidamiento adecuado, primero debemos cerrar el elemento strong, antes de cerrar el p.

El siguiente es un ejemplo de la forma incorrecta de anidar:

---

<sup>1</sup> `<p>Mi gato es <strong>muy grun .</p></strong>`

---

### 1.7.3 Elementos de bloque y elementos en línea

Hay dos categorías importantes de elementos en HTML — Estos son los elementos de bloque y los elementos en línea.

- Los elementos de bloque forman un bloque visible en la página. Aparecerán en una línea nueva después de cualquier contenido anterior. Cualquier contenido que vaya después también aparecerá en una línea nueva. Los elementos a nivel de bloque suelen ser elementos estructurales de la página. Por ejemplo, un elemento a nivel de bloque puede representar encabezados, párrafos, listas, menús de navegación o pies de página. Un elemento a nivel de bloque no estaría anidado dentro de un elemento en línea, pero podría estar anidado dentro de otro elemento a nivel de bloque.
- Los elementos en línea están contenidos dentro de elementos de bloque y delimitan solo pequeñas partes del contenido del documento; (no párrafos enteros o agrupaciones de contenido) Un elemento en línea no hará que aparezca una nueva línea en el documento. Suelen

utilizarse con texto. Por ejemplo es el caso de un elemento `<a>`(hipervínculo) o elementos de énfasis como `<em>`o `<strong>`.



Nota: HTML5 redefinió las categorías de elementos: consulta Categorías de contenido de elementos. Si bien estas definiciones son más precisas y menos ambiguas que sus predecesoras, las nuevas definiciones son mucho más complicadas de entender que `block` e `inline`. Este artículo seguirá con estos dos términos.

#### 1.7.4 Elementos vacíos

No todos los elementos siguen el patrón de etiqueta de apertura, contenido y etiqueta de cierre. Algunos elementos consisten solo en una etiqueta única, que se utiliza generalmente para insertar/incrustar algo en el documento en el lugar donde se le quiere incluir. Por ejemplo, el elemento `<img>`inserta una imagen en la página:

---

```

1 

```

---



Figura 1.27: Ejemplo elemento img

#### 1.7.5 Atributos

Los elementos también pueden tener atributos. Los atributos tienen este aspecto:



Figura 1.28: Atributos en un elemento html

Los atributos contienen información extra sobre el elemento que no se mostrará en el contenido. En este caso, el atributo `class` asigna al elemento un identificador que se puede utilizar para dotarlo de información de estilo.

Un atributo debería tener:

- Un espacio entre este y el nombre del elemento. (Para un elemento con más de un atributo, los atributos también deben estar separados por espacios).
- El nombre del atributo, seguido por un signo igual.
- Un valor del atributo, rodeado de comillas de apertura y cierre.

#### 1.7.6 Anatomía de un documento HTML

Los elementos HTML no son muy útiles por sí mismos. Ahora veremos cómo combinar los elementos individuales para formar una página HTML completa:

---

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">

```

---

```

5      <title>Mi pgina de prueba</title>
6  </head>
7  <body>
8      <p>Esta es mi pgina</p>
9  </body>
10 </html>

```

---

Aquí tenemos:

1. <!DOCTYPE html>: El elemento doctype. en la actualidad se ignora y se considera un legado histórico que hay que incluir para que todo funcione correctamente. <!DOCTYPE html>es la secuencia de caracteres más corta que se acepta como elemento doctype válido.
2. <html></html>: El elemento <html>. Este elemento envuelve todo el contenido de la página. A veces se lo conoce como el elemento raíz.
3. <head></head>: El elemento «head» (cabecera). Este elemento actúa como contenedor para todos los parámetros que quieras incluir en el documento HTML que no serán visibles a los visitantes de la página. Incluye cosas como palabras clave y la descripción de la página que quieras mostrar en los resultados de búsqueda, así como la hoja de estilo para formatear nuestro contenido, declaraciones de codificación de caracteres y más. Aprenderás más acerca de esto en el siguiente artículo de la serie.
4. <meta charset="utf-8">: Este elemento establece que tu documento HTML usará la codificación UTF-8, que incluye la gran mayoría de caracteres de todos los idiomas humanos escritos. En resumen: puede gestionar cualquier contenido textual que pongas en tu documento. No hay razón para no configurar este valor y te puede ayudar a evitar problemas más adelante.
5. <title></title>: El elemento <title>. Este establece el título de la página, que es el título que aparece en la pestaña del navegador en la que se carga la página. El título de la página también se utiliza para describir la página cuando se marca como favorita.
6. <body></body>: El elemento <body>. Contiene todo el contenido que quieras mostrar a los usuarios cuando visitan tu página, ya sea texto, imágenes, vídeos, juegos, pistas de audio reproducibles o cualquier otra cosa.

## 1.8 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.

### 1.8.1 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>

## 1.8.2 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.

## 1.8.3 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:

---

```
1 <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



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:

---

```
1 p, li {
2   color: green;
3 }
```

---

#### 1.8.4 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:

---

```
1 <ul>
2 <li>Punto uno</li>
3 <li class = "special">Punto dos</li>
4 <li>Punto <em>tres</em></li>
5 </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:

---

```
1 .special {
2   color: orange;
3   font-weight: bold;
4 }
```

---

Puedes aplicar la clase special a cualquier elemento de la página que deseas que tenga el mismo aspecto que este elemento de lista. Por ejemplo, es posible que deseas que el <span> 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:

---

```
1 li.special {
2   color: orange;
```

---

```

3     font-weight: bold;
4 }
```

---

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

---

```

1 li.special,
2 span.special {
3   color: orange;
4   font-weight: bold;
5 }
```

---

### 1.8.5 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 <em>: uno dentro de un párrafo y el otro dentro de un elemento de la lista. Para seleccionar solo un <em>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.



Combinador Css de descendiente: (espacio en blanco) o

---

```

1 li em {
2   color: rebeccapurple;
3 }
```

---

Este selector separará cualquier elemento <em>que esté dentro de (un descendiente de) <li>. Entonces, en tu documento de ejemplo, deberías encontrar que el <em>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.

---

```

1 h1 + p {
2   font-size: 200%;
3 }
```

---



---

```

1 <h1>I am a level one heading</h1>
2
3 <p>This is a paragraph of text. In the text is a <span>span element</span>
4 and also a <a href="http://example.com">link</a>.</p>
5
6 <p>This is the second paragraph. It contains an <em>emphasized</em> element.</p>
7
8 <ul>
9 <li>Item <span>one</span></li>
10 <li>Item two</li>
11 <li>Item <em>three</em></li>
12 </ul>
```

---

Salida:

I am a level one heading

This is a paragraph of text. In the text is a span element and also a [link](#).

This is the second paragraph. It contains an *emphasized* element.

- Item one
- Item two
- Item *three*

Figura 1.29: Salida de formato según la ubicación

### 1.8.6 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.

---

```
1 a:link {  
2     color: pink;  
3 }  
4  
5 a:visited {  
6     color: green;  
7 }
```

---

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

---

```
1 a:hover {  
2     text-decoration: none;  
3 }
```

---

### 1.8.7 Estilos en línea

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

---

```
1 <!DOCTYPE html>  
2 <html>  
3 <head>  
4 <meta charset="utf-8">  
5 <title>Mi experimento CSS</title>  
6 </head>  
7 <body>
```

```
8 <h1 style="color: blue; background-color: yellow; border: 1px solid black;">Hola  
9   mundo!</h1>  
10 <p style="color:red;">Este es mi primer ejemplo de CSS</p>  
11 </body>  
11 </html>~
```

---



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

### 1.8.8 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.

---

```
1 h1  
2 a:link  
3 .manythings  
4 #onething  
5 *  
6 .box p  
7 .box p:first-child  
8 h1, h2, .intro
```

---

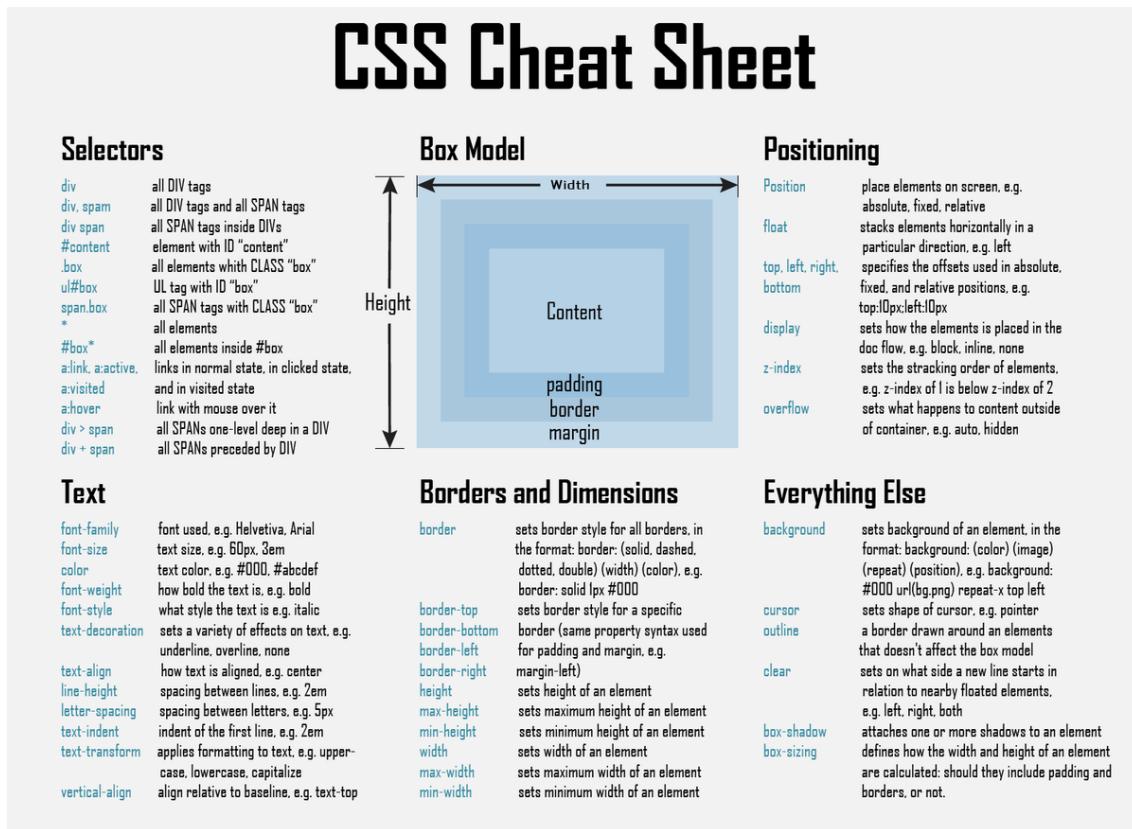


Figura 1.30: Hoja de trucos CSS

### 1.8.9 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.

```

1 .special {
2   color: red;
3 }
4
5 p {
6   color: blue;
7 }
```

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.

### 1.8.10 @rules

Las @rules (leído `@-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`:

---

```
1 @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.

---

```
1 body {  
2     background-color: pink;  
3 }  
4  
5 @media (min-width: 30em) {  
6     body {  
7         background-color: blue;  
8     }  
9 }
```

---

### 1.8.11 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:

---

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

---

Hace lo mismo que todas estas juntas:

---

```
1 padding-top: 10px;  
2 padding-right: 15px;  
3 padding-bottom: 15px;  
4 padding-left: 5px;
```

---

### 1.8.12 ¿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.

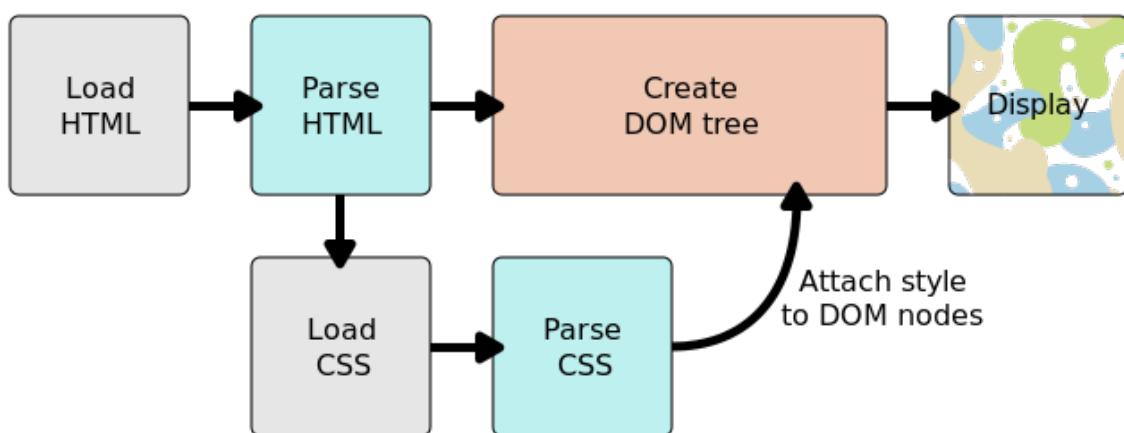


Figura 1.31: Carga del css

### 1.8.13 Normal flow

Los elementos en las páginas web se presentan de acuerdo con el flujo normal, hasta que hacemos algo que cambie eso. Esta sección explica las bases del flujo normal para aprender cómo cambiarlo.

¿Cómo se presentan por defecto los elementos?

En primer lugar, se toma de las cajas de cada uno de los elementos el contenido, luego se añade cualquier área de relleno, borde y margen alrededor de ellas; es el modelo de cajas que hemos visto antes.

De manera predeterminada, el contenido de un elemento de nivel de bloque es el 100% del ancho de su elemento padre y su altura viene determinada por su contenido. Los elementos en línea tienen su altura y anchura determinados por su contenido. No puedes establecer el ancho o la altura de los elementos en línea, simplemente se ubican dentro del contenido de los elementos de nivel de bloque. Si deseas controlar el tamaño de un elemento en línea de esta manera, debes configurarlo para que se comporte como un elemento de nivel de bloque con `display: block;` (o incluso, `display: inline-block;`, que combina características de ambos).

Esto explica los elementos individuales, pero ¿qué hay del modo como los elementos interactúan entre sí? El flujo de diseño normal (mencionado en el artículo de introducción al diseño) es el sistema mediante el cual los elementos se colocan en la ventana gráfica del navegador. De manera predeterminada, los elementos de nivel de bloque se presentan en la dirección del flujo del bloque, en función del modo de escritura de los padres (initial: horizontal-tb): cada uno aparecerá en una línea nueva debajo de la última, y estarán separados por cualquier margen que se establezca en ellos. Por lo tanto, en inglés, o en cualquier otro modo de escritura horizontal y de arriba a abajo, los elementos de nivel de bloque se disponen verticalmente.

Los elementos en línea se comportan de manera diferente: no aparecen en líneas nuevas; en su lugar, se asientan en la misma línea entre sí y con cualquier contenido de texto adyacente (o envuelto), siempre que tengan espacio dentro del ancho del elemento de nivel de bloque primario. Si no hay espacio, el texto o los elementos que desborden bajarán a la línea siguiente.

Si dos elementos adyacentes tienen algún margen configurado y los dos márgenes se tocan, se mantiene el mayor de los dos y el menor desaparece; esto se llama colapso del margen, y ya lo hemos visto antes.

## Flujo de los documentos básicos

Soy un elemento básico de nivel de bloque. Mis elementos de nivel de bloque adyacentes se encuentran en líneas nuevas debajo de mí.

Cubrimos por defecto el 100% del ancho de nuestro elemento principal, y somos tan altos como nuestro contenido secundario. Nuestro ancho y alto total es nuestro contenido + área de relleno + ancho/alto del borde.

Estamos separados por nuestros márgenes. Debido al colapso del margen, estamos separados por el ancho de uno de nuestros márgenes, no por ambos.

Los elementos en línea **como este** y **este otro** se ubican en la misma y la de los nodos de texto adyacentes, mientras hay espacio en la misma línea. Si un elemento en línea desborda, **sigue por la línea siguiente, si es posible (como la que contiene este texto)**, o simplemente pasa a una línea nueva, como hace esta imagen:



Figura 1.32: Ejemplo Flujo normal

### 1.8.14 Contenido desbordante

El desbordamiento es lo que sucede cuando hay demasiado contenido para caber en un contenedor. En esta guía aprenderá qué es el desbordamiento y cómo gestionarlo.

Todo en CSS es una caja. Puede restringir el tamaño de estos cuadros asignando valores de ancho y alto (o tamaño en línea y tamaño de bloque). El desbordamiento ocurre cuando hay demasiado contenido para caber en una caja. CSS proporciona varias herramientas para administrar el desbordamiento. A medida que avance con el diseño de CSS y escriba CSS, encontrará más situaciones de desbordamiento.

CSS intenta evitar la "pérdida de datos". Consideremos dos ejemplos que demuestran el comportamiento predeterminado de CSS cuando hay desbordamiento.

El primer ejemplo es una caja que ha sido restringida estableciendo una altura. Luego agregamos contenido que excede el espacio asignado. El contenido desborda el cuadro y cae en el párrafo siguiente.

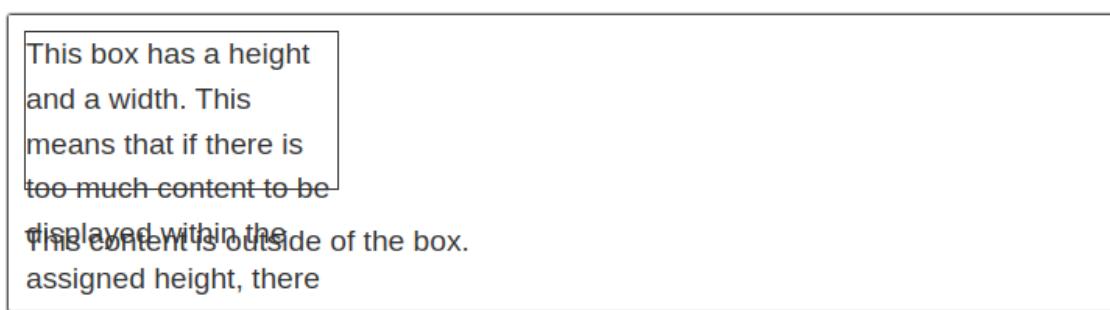


Figura 1.33: Ejemplo desbordamiento

El segundo ejemplo es una palabra en un cuadro. La caja se ha hecho demasiado pequeña para la palabra, por lo que se sale de la caja.



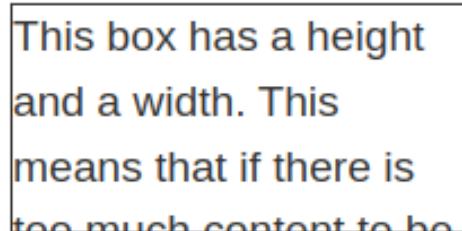
Figura 1.34: Ejemplo desbordamiento

Siempre que sea posible, CSS no oculta el contenido. Esto provocaría la pérdida de datos. El problema con la pérdida de datos es que es posible que no lo notes. Es posible que los visitantes del sitio web no se den cuenta. Si el botón de enviar en un formulario desaparece y nadie puede completarlo, ¡esto podría ser un gran problema! En cambio, CSS se desborda de manera visible. Es más probable que vea que hay un problema. En el peor de los casos, un visitante del sitio le informará que el contenido se superpone.

La propiedad de desbordamiento es cómo tomas el control del desbordamiento de un elemento. Es la forma en que le indica al navegador cómo debe comportarse. El valor predeterminado de overflow es visible. Con este valor predeterminado, podemos ver el contenido cuando se desborda.

Para recortar el contenido cuando se desborda, puede establecer desbordamiento: oculto. Esto hace exactamente lo que dice: oculta el desbordamiento. Tenga en cuenta que esto puede hacer que algunos contenidos sean invisibles. Solo debe hacer esto si ocultar contenido no causará problemas.

```
1 .box {  
2     border: 1px solid #333333;  
3     width: 200px;  
4     height: 100px;  
5     overflow: hidden;  
6 }
```



This content is outside of the box.

Figura 1.35: Ejemplo desbordamiento

Usando overflow: scroll, los navegadores con barras de desplazamiento visibles siempre las mostrarán, incluso si no hay suficiente contenido para desbordar.

```
1 .box {  
2     overflow: scroll;  
3 }
```



Para simplemente desplazarse en el eje y, puede usar la propiedad overflow-y, configurando overflow-y: scroll. y overflow-x para desplazarse en el eje de las x

## Flexbox

Flexbox es un método de diseño de página unidimensional para compaginar elementos en filas o columnas. Los elementos de contenido se ensanchan para llenar el espacio adicional y se encogen para caber en espacios más pequeños. En este artículo expondremos todas sus características básicas.

## Grid

[2]

CSS Grid Layout presenta un sistema de cuadrícula bidimensional para CSS. Las cuadrículas se pueden utilizar para posicionar áreas principales de la página o pequeños elementos de la interfaz de usuario.

CSS Grid layout contiene funciones de diseño dirigidas a los desarrolladores de aplicaciones web. El CSS grid se puede utilizar para lograr muchos diseños diferentes. También se destaca por permitir dividir una página en áreas o regiones principales, por definir la relación en términos de tamaño, posición y capas entre partes de un control construido a partir de primitivas HTML.

Al igual que las tablas, el grid layout permite a un autor alinear elementos en columnas y filas. Sin embargo, con CSS grid son posibles muchos más diseños y de forma más sencilla que con las tablas. Por ejemplo, los elementos secundarios de un contenedor de cuadrícula podrían posicionarse para que se solapen y se superpongan, de forma similar a los elementos posicionados en CSS.

### ¿Qué es una cuadrícula(grid)?

Una cuadrícula es un conjunto de líneas horizontales y verticales que se intersectan - un grupo define columnas y el otro filas. Los elementos se pueden colocar en la cuadrícula respetando estas columnas y filas. El diseño de cuadrícula CSS tiene las siguientes características:

- Tamaños fijos y flexibles: puede crear una cuadrícula con tamaños fijos, utilizando píxeles, también se puede crear una cuadrícula utilizando tamaños flexibles con porcentajes o con la nueva unidad de medida fr (fracción), diseñada para este propósito.
- Posicionamiento de elementos: puede colocar elementos en una ubicación precisa en la cuadrícula utilizando números de línea, nombres o seleccionando un área de la cuadrícula.
- Creación de líneas adicionales para alojar contenido: Grid Layout es lo suficientemente flexible como para permitir agregar filas y columnas adicionales cuando sea necesario.
- Control de alineación: contiene características de alineación para poder controlar la forma cómo se alinean los elementos una vez colocados en un área de cuadrícula y cómo está alineada toda la cuadrícula.
- Control de contenido superpuesto: Se puede colocar más de un elemento en una celda de la cuadrícula o área, las cuales pueden solaparse o superponerse total o parcialmente entre sí. Esta estratificación puede ser controlada con la propiedad z-index.



Para mayor información vea: [https://developer.mozilla.org/es/docs/Web/CSS/CSS\\_Grid\\_Layout/Basic\\_Concepts\\_of\\_Grid\\_Layout](https://developer.mozilla.org/es/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout)

## 1.8.15 Frameworks CSS

[7] Un framework de CSS es una biblioteca de estilos genéricos que puede ser usada para implementar diseños web. Aportan una serie de utilidades que pueden ser aprovechadas frecuentemente en los distintos diseños web.

### Ventajas

Un framework de CSS, si está bien diseñado e implementado, proporciona las siguientes ventajas:

- Proporcionar una forma fácil y por tanto rápida de implementar diseños web.
- Nos aseguran que el diseño va a funcionar en una amplia gama de navegadores
- Nos aseguran que su código cumple cierta normas estándar.
- Nos aseguran cierto grado de fiabilidad en la eficacia de las utilidades que nos aportan. El framework se supone que está bien probado para asegurarnos que no hay errores.

### Inconvenientes

El uso de un framework de CSS puede (no siempre) llevar aparejado las siguientes desventajas:

- La importación de código del framework que no es necesario en nuestro diseño web concreto. Esto provoca un incremento innecesario del consumo del ancho de banda y del tiempo de descarga.
- Hay un menor control por parte del maquetador de lo que realmente está sucediendo en la visualización de la página web. Esto suele ser un problema cuando se tiene que corregir algún efecto indeseado.
- Al diseñar con código prehecho, podemos estar limitándonos en cuanto las posibilidades de elección del diseño web.

### Algunos frameworks css

Tailwindcss <https://tailwindcss.com/>

Daisyui <https://daisyui.com/>

Material UI <https://mui.com/material-ui/getting-started/overview/>

**Bulma** <https://bulma.io/>  
**NextUI** <https://nextui.org/>  
**Ant Design** <https://ant.design/>  
**PrimeNG** <https://www.primefaces.org/primeng/>  
**PrimeReact** <https://www.primefaces.org/primereact/>  
**Chakra** <https://chakra-ui.com/>

## 1.9 Javascript

### 1.9.1 ECMAScript

[3] Ecma International es una asociación industrial dedicada a la estandarización de los sistemas de información y comunicación.

Ecma está impulsada por miembros de la industria para satisfacer sus necesidades, brindando un panorama competitivo saludable basado en la diferenciación de productos y servicios en lugar de modelos tecnológicos, generando confianza entre los proveedores y usuarios de nuevas tecnologías.

[7] ECMAScript es una especificación de lenguaje de programación publicada por ECMA International. El desarrollo empezó en 1996 y estuvo basado en el popular lenguaje JavaScript propuesto como estándar por Netscape Communications Corporation. Actualmente está aceptado como el estándar ISO/IEC 22275:2018.

ECMAScript define un lenguaje de tipos dinámicos ligeramente inspirado en Java y otros lenguajes del estilo de C. Soporta algunas características de la programación orientada a objetos mediante objetos basados en prototipos y pseudoclases.

[3] Estandarización del lenguaje de programación ECMAScript® de propósito general, multiplataforma e independiente del proveedor. Esto incluye la sintaxis, la semántica y las bibliotecas del lenguaje y las tecnologías complementarias que respaldan el lenguaje. Este trabajo pretende no utilizar patentes o, de ser así, solo patentes libres de regalías.

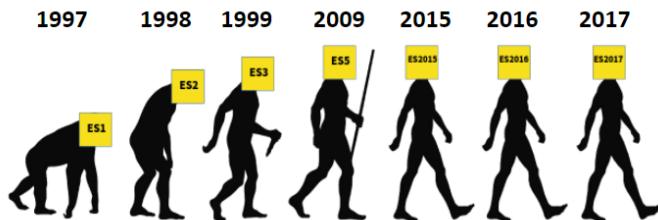


Figura 1.36: Evolución EcmaScript :-)



La versión actual del estandar EcmaScript en el momento de escribir esta guía es la 2022 y la puedes encontrar en el siguiente enlace: [https://www.ecma-international.org/wp-content/uploads/ECMA-262\\_13th\\_edition\\_june\\_2022.pdf](https://www.ecma-international.org/wp-content/uploads/ECMA-262_13th_edition_june_2022.pdf)

### 1.9.2 Javascript

[2] JavaScript es un lenguaje de programación o de secuencias de comandos que te permite implementar funciones complejas en páginas web, cada vez que una página web hace algo más que sentarse allí y mostrar información estática para que la veas, muestra oportunas actualizaciones de contenido, mapas interactivos, animación de Gráficos 2D/3D, desplazamiento de máquinas reproductoras de vídeo, etc., puedes apostar que probablemente JavaScript está involucrado. Es la

tercera capa del pastel de las tecnologías web estándar, dos de las cuales (HTML y CSS) hemos cubierto con mucho más detalle en otras partes del Área de aprendizaje.

El núcleo del lenguaje JavaScript de lado del cliente consta de algunas características de programación comunes que te permiten hacer cosas como:

- Almacenar valores útiles dentro de variables. En el ejemplo anterior, por ejemplo, pedimos que ingreses un nuevo nombre y luego almacenamos ese nombre en una variable llamada name.
- Operaciones sobre fragmentos de texto (conocidas como cadenas "strings" en programación).
- Ejecuta código en respuesta a ciertos eventos que ocurren en una página web.
- ¡Y mucho más!

Sin embargo, lo que aún es más emocionante es la funcionalidad construida sobre el lenguaje JavaScript de lado del cliente. Las denominadas interfaces de programación de aplicaciones (API) te proporcionan superpoderes adicionales para utilizar en tu código JavaScript.

Cuando el navegador encuentra un bloque de JavaScript, generalmente lo ejecuta en orden, de arriba a abajo. Esto significa que debes tener cuidado con el orden en el que colocas las cosas.

JavaScript es un lenguaje de programación interpretado ligero. El navegador web recibe el código JavaScript en su forma de texto original y ejecuta el script a partir de ahí. Desde un punto de vista técnico, la mayoría de los intérpretes de JavaScript modernos utilizan una técnica llamada compilación en tiempo real para mejorar el rendimiento; el código fuente de JavaScript se compila en un formato binario más rápido mientras se usa el script, de modo que se pueda ejecutar lo más rápido posible. Sin embargo, JavaScript todavía se considera un lenguaje interpretado, ya que la compilación se maneja en el entorno de ejecución, en lugar de antes.

### 1.9.3 ¿Cómo agregas JavaScript a tu página?

JavaScript se aplica a tu página HTML de manera similar a CSS. Mientras que CSS usa elementos <link> para aplicar hojas de estilo externas y elementos <style> para aplicar hojas de estilo internas a HTML, JavaScript solo necesita un amigo en el mundo de HTML: el elemento <script>.

Hay una serie de problemas relacionados con la carga de los scripts en el momento adecuado. ¡Nada es tan simple como parece! Un problema común es que todo el HTML de una página se carga en el orden en que aparece. Si estás utilizando JavaScript para manipular elementos en la página (o exactamente, el Modelo de objetos del documento (en-US)), tu código no funcionará si el JavaScript se carga y procesa antes que el HTML que estás intentando haga algo.

Una solución pasada de moda a este problema solía ser colocar tu elemento script justo en la parte inferior del cuerpo (por ejemplo, justo antes de la etiqueta </body>), para que se cargara después de haber procesado todo el HTML. El problema con esta solución es que la carga/procesamiento del script está completamente bloqueado hasta que se haya cargado el DOM HTML. En sitios muy grandes con mucho JavaScript, esto puede causar un importante problema de rendimiento y ralentizar tu sitio.

#### async y defer

En realidad, hay dos modernas características que podemos usar para evitar el problema del bloqueo de script: async y defer (que vimos anteriormente). Veamos la diferencia entre estas dos.

Los scripts cargados con el atributo async (ve más abajo) descargarán el script sin bloquear el renderizado de la página y lo ejecutará tan pronto como el script se termine de descargar. No tienes garantía de que los scripts se ejecuten en un orden específico, solo que no detendrán la visualización del resto de la página. Es mejor usar async cuando los scripts de la página se ejecutan de forma independiente y no dependen de ningún otro script de la página.

Los scripts cargados con el atributo defer (ve a continuación) se ejecutarán en el orden en que aparecen en la página y los ejecutará tan pronto como se descarguen el script y el contenido:

---

```
1 <script defer src="js/vendor/jquery.js"></script>
2
3 <script defer src="js/script2.js"></script>
4
5 <script defer src="js/script3.js"></script>
```

---

Todos los scripts con el atributo defer se cargarán en el orden en que aparecen en la página. Entonces, en el segundo ejemplo, podemos estar seguros de que jquery.js se cargará antes que script2.js y script3.js y que script2.js se cargará antes de script3.js. No se ejecutarán hasta que se haya cargado todo el contenido de la página, lo cual es útil si tus scripts dependen de que el DOM esté en su lugar (por ejemplo, modifican uno o más elementos de la página).

Para resumir:

- async y defer indican al navegador que descargue los scripts en un hilo separado, mientras que el resto de la página (el DOM, etc.) se descarga, por lo que los scripts no bloquean la carga de la página.
- Si tus scripts se deben ejecutar inmediatamente y no tienen ninguna dependencia, utiliza async.
- Si tus scripts necesitan esperar a ser procesados y dependen de otros scripts y/o del DOM en su lugar, cárgalos usando defer y coloca tus elementos <script> correspondientes en el orden que deseas que el navegador los ejecute.

#### 1.9.4 Un poco de sintaxis

##### Declaraciones if ... else

Echemos un vistazo a la declaración condicional más común que usarás en JavaScript.

###### — El humilde if ... else statement.

Una sintaxis básica if...else luce así. pseudocode:

---

```
1 if (condicin) {
2     cdigo a ejecutar si la condicin es verdadera
3 } else {
4     ejecuta este otro cdigo si la condicin es falsa
5 }
```

---

##### else if

Hay una forma de encadenar opciones / resultados adicionales extras a if...else — usando else if. Cada opción extra requiere un bloque adicional para poner en medio de bloque if() ... y else ...

##### Bucles

Los lenguajes de programación son muy útiles para completar rápidamente tareas repetitivas, desde múltiples cálculos básicos hasta cualquier otra situación en donde tengas un montón de elementos de trabajo similares que completar. Aquí vamos a ver las estructuras de bucles disponibles en JavaScript que pueden manejar tales necesidades.

Los bucles de programación están relacionados con todo lo referente a hacer una misma cosa una y otra vez — que se denomina como iteración en el idioma de programación.

###### El bucle estándar for

Exploraremos algunos constructores de bucles específicos. El primero, que usarás la mayoría de las veces, es el bucle for - este tiene la siguiente sintaxis:

---

```
1 for (inicializador; condicin de salida; expresin final) {
2     // cdigo a ejecutar
3 }
```

---

Observa un ejemplo real para poder entender esto más claramente.

---

```

1 var cats = ['Bill', 'Jeff', 'Pete', 'Biggles', 'Jasmin'];
2 var info = 'My cats are called ';
3 var para = document.querySelector('p');
4
5 for (var i = 0; i < cats.length; i++) {
6   info += cats[i] + ', ';
7 }
8
9 para.textContent = info;

```

---

Esto nos da el siguiente resultado:

---

```
1 My cats are called Bill, Jeff, Pete, Biggles, Jasmin,
```

---



Más información sobre bucles en: [https://developer.mozilla.org/es/docs/Learn/JavaScript/Building\\_blocks/Looping\\_code](https://developer.mozilla.org/es/docs/Learn/JavaScript/Building_blocks/Looping_code)

Por los momentos no se continua con la sintaxis de javascript, se explicarán mas conceptos desde el punto de vista de Typescript en la siguiente sección.

## 1.10 Typescript

[5] JavaScript (también conocido como ECMAScript) comenzó su vida como un simple lenguaje de secuencias de comandos para navegadores. En el momento en que se inventó, se esperaba que se usara para fragmentos cortos de código incrustados en una página web; escribir más de unas pocas docenas de líneas de código hubiera sido algo inusual. Debido a esto, los primeros navegadores web ejecutaban dicho código con bastante lentitud. Sin embargo, con el tiempo, JS se hizo cada vez más popular y los desarrolladores web comenzaron a usarlo para crear experiencias interactivas.

Los desarrolladores de navegadores web respondieron a este mayor uso de JS optimizando sus motores de ejecución (compilación dinámica) y ampliando lo que se podía hacer con él (agregando API), lo que a su vez hizo que los desarrolladores web lo usaran aún más. En los sitios web modernos, su navegador ejecuta con frecuencia aplicaciones que abarcan cientos de miles de líneas de código. Este es un crecimiento largo y gradual de "la web", comenzando como una simple red de páginas estáticas y evolucionando hacia una plataforma para aplicaciones ricas de todo tipo.

Más que esto, JS se ha vuelto lo suficientemente popular como para usarse fuera del contexto de los navegadores, como implementar servidores JS usando node.js. La naturaleza de "ejecutar en cualquier lugar" de JS lo convierte en una opción atractiva para el desarrollo multiplataforma. ¡ Hay muchos desarrolladores en estos días que usan solo JavaScript para programar toda su pila!

Para resumir, tenemos un lenguaje que fue diseñado para usos rápidos y luego creció hasta convertirse en una herramienta completa para escribir aplicaciones con millones de líneas. Cada idioma tiene sus propias peculiaridades : rarezas y sorpresas, y el humilde comienzo de JavaScript hace que tenga muchas de ellas. Algunos ejemplos:

- El operador de igualdad de JavaScript ( == ) coacciona sus argumentos, lo que genera un comportamiento inesperado:

---

```

1 if ('' == 0) {
2   // It is! But why??
3 }

```

---

- JavaScript también permite acceder a propiedades que no están presentes (Preste atención a **height**):

```
1 const obj = { width: 10, height: 15 };
2 // Why is this NaN? Spelling is hard!
3 const area = obj.width * obj.heigth;
```

La mayoría de los lenguajes de programación arrojarían un error cuando ocurren este tipo de errores, algunos lo harían durante la compilación, antes de que se ejecute cualquier código. Al escribir programas pequeños, tales peculiaridades son molestas pero manejables; al escribir aplicaciones con cientos o miles de líneas de código, estas sorpresas constantes son un problema grave.

### 1.10.1 TypeScript: un verificador de tipo estático

Dijimos anteriormente que algunos lenguajes no permitirían que esos programas con errores se ejecutaran en absoluto. La detección de errores en el código sin ejecutarlo se conoce como verificación estática . Determinar qué es un error y qué no en función de los tipos de valores en los que se opera se conoce como verificación de tipos estáticos.

TypeScript verifica un programa en busca de errores antes de la ejecución, y lo hace en función de los tipos de valores , es un verificador de tipo estático . Por ejemplo, el último ejemplo anterior tiene un error debido al tipo de obj. Aquí está el error encontrado por TypeScript:

```
1 const obj = { width: 10, height: 15 };
2 const area = obj.width * obj.heigth;
3
4 Property 'heigth' does not exist on type '{ width: number; height: number;
  }'. Did you mean 'height'?
```

### 1.10.2 Un superconjunto tipado de JavaScript

Sin embargo, ¿cómo se relaciona TypeScript con JavaScript?

Sintaxis TypeScript es un lenguaje que es un superconjunto de JavaScript: por lo tanto, la sintaxis JS es TS legal. La sintaxis se refiere a la forma en que escribimos texto para formar un programa.

TypeScript no considera ningún código JavaScript como un error debido a su sintaxis. Esto significa que puede tomar cualquier código JavaScript que funcione y ponerlo en un archivo TypeScript sin preocuparse por cómo está escrito exactamente.

### 1.10.3 Tipos

TypeScript es un superconjunto tipado , lo que significa que agrega reglas sobre cómo se pueden usar diferentes tipos de valores. El error anterior acerca de obj.heighthno fue un error de sintaxis : es un error de usar algún tipo de valor (un tipo ) de manera incorrecta.

En términos generales, una vez que el compilador de TypeScript termina de verificar su código, borra los tipos para producir el código compilado resultante. Esto significa que una vez que se compila su código, el código JS simple resultante no tiene información de tipo.

Esto también significa que TypeScript nunca cambia el comportamiento de su programa en función de los tipos que infiere. La conclusión es que, si bien es posible que vea errores de tipo durante la compilación, el sistema de tipo en sí no influye en cómo funciona su programa cuando se ejecuta.

Finalmente, TypeScript no proporciona ninguna biblioteca de tiempo de ejecución adicional. Sus programas utilizarán la misma biblioteca estándar (o bibliotecas externas) que los programas

de JavaScript, por lo que no hay un marco adicional específico de TypeScript para aprender.

## Primitivos

JavaScript tiene tres primitivas muy utilizadas: string, number, y boolean. Cada una tiene un tipo correspondiente en TypeScript. Como es de esperar, estos son los mismos nombres que verías si usaras el operador `typeof` de JavaScript en un valor de esos tipos:

- string representa valores de cadena como "Hola, mundo".
- number es para números como 42. JavaScript no tiene un valor especial en tiempo de ejecución para los enteros, así que no hay equivalente a int o float - todo es simplemente número.
- boolean es para los dos valores true y false.

## Arreglos

Para especificar el tipo de un array como [1, 2, 3], puedes utilizar la sintaxis `número[]`; esta sintaxis funciona para cualquier tipo (por ejemplo, `cadena[]` es un array de cadenas, y así sucesivamente). También puedes ver esto escrito como `Array<número>`, que significa lo mismo.

## any

TypeScript también tiene un tipo especial, `any`, que puedes usar siempre que no quieras que un valor particular cause errores de comprobación de tipos.

Cuando un valor es de tipo `any`, puedes acceder a sus propiedades (que a su vez serán de tipo `any`), llamarlo como una función, asignarlo a (o desde) un valor de cualquier tipo, o cualquier otra cosa que sea sintácticamente legal.

Para mayor información sobre typescript leer los siguientes apartados de documentación oficial:  
<https://www.typescriptlang.org/docs/handbook/2/everyday-types.html> <https://www.typescriptlang.org/docs/handbook/2/functions.html> <https://www.typescriptlang.org/docs/handbook/2/objects.html> <https://www.typescriptlang.org/docs/handbook/2/classes.html>

A continuación algunas hojas de trucos

### TypeScript Cheat Sheet

## Interface

### Key points

Used to describe the shape of objects, and can be extended by others.

Almost everything in JavaScript is an object and `interface` is built to match their runtime behavior.

### Built-in Type Primitives

`boolean`, `string`, `number`, `undefined`, `null`, `any`, `unknown`, `never`, `void`, `bignumber`, `symbol`

### Common Built-in JS Objects

`Date`, `Error`, `Array`, `Map`, `Set`, `Regex`, `Promise`

### Type Literals

```
Object:  
{ field: string }  
Function:  
(arg: number) => string  
Arrays:  
string[] or Array<string>  
Tuple:  
[string, number]
```

### Avoid

`Object`, `String`, `Number`, `Boolean`

## Common Syntax

```
interface JsonResponse extends Response, HTTPable {  
    version: number;  
    /** In bytes */  
    payloadSize: number;  
    outOfStock?: boolean;  
    update: (retryTimes: number) => void;  
    update(retryTimes: number): void;  
    (): JsonResponse  
    new(s: string): JsonResponse;  
    [key: string]: number;  
    readonly body: string;  
}
```

Optional take properties from existing interface or type

JSDoc comment attached to show in editors

This property might not be on the object

These are two ways to describe a property which is a function

You can call this object via () - (functions in JS are objects which can be called)

You can use `new` on the object this interface describes

Any property not described already is assumed to exist, and all properties must be numbers

Tells TypeScript that a property can not be changed

## Generics

Type parameter

Declare a type which can change in your interface

```
interface APICall<Response> {  
    data: Response  
}  
Usage  
const api: APICall<ArtworkCall> = ...  
api.data // Artwork
```

Used here

You can constrain what types are accepted into the generic parameter via the `extends` keyword.

```
interface APICall<Response extends { status: number }> {  
    data: Response  
}  
const api: APICall<ArtworkCall> = ...  
api.data.status
```

Gets a constraint on the type which means only types with a 'status' property can be used

## Overloads

A callable interface can have multiple definitions for different sets of parameters

```
interface Expect {  
    (matcher: boolean): string  
    (matcher: string): boolean;  
}
```

## Get & Set

Objects can have custom getters or setters

```
interface Ruler {  
    get size(): number;  
    set size(value: number | string);  
}
```

### Usage

```
const r: Ruler = ...  
r.size = 12  
r.size = "36"
```

## Extension via merging

Interfaces are merged, so multiple declarations will add new fields to the type definition.

```
interface APICall {  
    data: Response  
}  
  
interface APICall {  
    error?: Error  
}
```

## Class conformance

You can ensure a class conforms to an interface via `implements`:

```
interface Syncable { sync(): void }  
class Account implements Syncable { ... }
```

Figura 1.37: Interfaces

TypeScript Cheat Sheet **Class**

Key points	A TypeScript class has a few type-specific extensions to ES2015 JavaScript classes, and one or two runtime additions.
<b>Creating an class instance</b>	<code>class ABC { ... }</code> <code>const abc = new ABC()</code>
Parameters to the new ABC come from the constructor function.	
<b>private x vs #private</b>	The prefix private is a type-only addition and has no effect at runtime. Code outside of the class can reach into the item in the following case:
	<code>class Bag {     private item: any }</code>
Vs #private which is runtime private and has enforcement inside the JavaScript engine that it is only accessible inside the class:	<code>class Bag { #item: any }</code>
'this' in classes	The value of 'this' inside a function depends on how the function is called. It is not guaranteed to always be the class instance which you may be used to in other languages.
Vs 'this' parameters, use the bind function, or arrow functions to work around the issue when it occurs.	
<b>Type and Value</b>	Surprise, a class can be used as both a type or a value.
	<code>const a:Bag = new Bag()</code>
So, be careful not to do this:	<code>class C implements Bag {}</code>
<b>Common Syntax</b>	<code>class User extends Account implements Updatable, Serializable {     id: string; // A field     displayName?: boolean; // An optional field     name: string; // A 'trust me, it's there' field     #attributes: Map&lt;any, any&gt;; // A private field     roles = ["user"]; // A field with a default     readonly createdAt = new Date() // A readonly field with a default      constructor(id: string, email: string) {         super(id);         this.email = email; // In strict: true this code is checked against the fields to ensure it is set up correctly         ...     };      setName(name: string) { this.name = name }     verifyName = (name: string) =&gt; { ... } // Ways to describe class methods (and arrow function fields)      sync(): Promise&lt;...&gt;     sync(cb: (result: string) =&gt; void): void     sync(cb?: (result: string) =&gt; void): void   Promise&lt;...&gt; { ... } // A function with 2 overload definitions      get accountID(): string {         return accountID; // Getters and setters     }     set accountID(value: string) {         accountID = value; // Setters and getters     };      private makeRequest() { ... }     protected handleRequest() { ... } // Private access is just to this class, protected allows to subclasses. Only used for type checking, public is the default.      static #userCount = 0;     static registerUser(user: User) { ... } // Static fields / methods      static { this.#userCount = -1 } // Static blocks for setting up static vars. 'this' refers to the static class }</code>
<b>Generics</b>	Declare a type which can change in your class methods.
	<code>class Box&lt;Type&gt; {     contents: Type     constructor(value: Type) {         this.contents = value;     }     const stringBox = new Box("a package") // Used here }</code>
	These features are TypeScript specific language extensions which may never make it to JavaScript with the current syntax.
	<b>Parameter Properties</b>
	A TypeScript specific extension to classes which automatically set an instance field to the input parameter.
	<code>class Location {     constructor(public x: number, public y: number) {} } const loc = new Location(20, 40); loc.x // 20 loc.y // 40</code>
	<b>Abstract Classes</b>
	A class can be declared as not implementable, but as existing to be subclassed in the type system. As can members of the class.
	<code>abstract class Animal {     abstract getName(): string;     printName() {         console.log("Hello, " + this.getName());     } } class Dog extends Animal { getName(): { ... } }</code>
	<b>Decorators and Attributes</b>
	You can use decorators on classes, class methods, accessors, property and parameters to methods.
	<code>import {     Syncable, triggersSync, preferCache, required } from "mylib"  @Syncable class User {     @triggersSync()     save() { ... }      @preferCache(false)     get displayName() { ... }      update(@required info: Partial&lt;User&gt;) { ... } }</code>

Figura 1.38: Clases

TypeScript Cheat Sheet	Type	Key points	Full name is "type alias" and are used to provide names to type literals	Supports more rich type-system features than interfaces.	
Type vs Interface					These features are great for building libraries, describing existing JavaScript code and you may find you rarely reach for them in mostly TypeScript applications.
• Interfaces can only describe object shapes • Interfaces can be extended by declaring it multiple times • In performance critical types interface comparison checks can be faster.					
Think of Types Like Variables					
Much like how you can create variables with the same name in different scopes, a type has similar semantics.					
Build with Utility Types					
TypeScript includes a lot of global types which will help you do common tasks in the type system. Check the site for them.					
Primitive Type					
Useful for documentation mainly					
<code>type SanitizedInput = string;</code>					
<code>type MissingNo = 404;</code>					
Object Literal Type					
<code>type Location = {</code>					
<code>x: number;</code>					
<code>y: number;</code>					
<code>};</code>					
Tuple Type					
A tuple is a special-cased array with known types at specific indexes.					
<code>type Data = [</code>					
<code>location: Location,</code>					
<code>timestamp: string</code>					
<code>];</code>					
Union Type					
Describes a type which is one of many options, for example a list of known strings.					
<code>type Size =</code>					
<code>"small"   "medium"   "large"</code>					
Intersection Types					
A way to merge/extend types					
<code>type Location =</code>					
<code>{ x: number } &amp; { y: number }</code>					
<code>// { x: number, y: number }</code>					
Type Indexing					
A way to extract and name from a subset of a type.					
<code>type Response = { data: { ... } }</code>					
<code>type Data = Response["data"]</code>					
<code>// { ... }</code>					
Type from Value					
Re-use the type from an existing JavaScript runtime value via the <code>typeof</code> operator.					
<code>const data = { ... }</code>					
<code>type Data = typeof data</code>					
Type from Func Return					
Re-use the return value from a function as a type.					
<code>const createFixtures = () =&gt; { ... }</code>					
<code>type Fixtures = ReturnType&lt;typeof createFixtures&gt;</code>					
<code>function test(fixture: Fixtures) {}</code>					
Type from Module					
<code>const data: import("./data").data</code>					
Mapped Types					
Acts like a map statement for the type system, allowing an input type to change the structure of the new type.					
<code>type Artist = { name: string, bio: string }</code>					
<code>type Subscriber&lt;Type&gt; = {</code>					Sets type as a function with original type as param
<code>[Property in keyof Type]</code>					
<code>(newValue: Type[Property]) =&gt; void</code>					
<code>}</code>					
<code>type ArtistSub = Subscriber&lt;Artist&gt;</code>					
<code>// { name: (nv: string) =&gt; void,</code>					
<code>// bio: (nv: string) =&gt; void }</code>					
Conditional Types					
Acts as "if statements" inside the type system. Created via generics, and then commonly used to reduce the number of options in a type union.					
<code>type HasFourLegs&lt;Animal&gt; =</code>					
<code>Animal extends { legs: 4 } ? Animal</code>					
<code>: never</code>					
<code>type Animals = Bird   Dog   Ant   Wolf;</code>					
<code>type FourLegs = HasFourLegs&lt;Animals&gt;</code>					
<code>// Dog   Wolf</code>					
Template Union Types					
A template string can be used to combine and manipulate text inside the type system.					
<code>type SupportedLangs = "en"   "pt"   "zh";</code>					
<code>type FooterLocaleIDs = "header"   "footer";</code>					
<code>type AllLocaleIDs =</code>					
<code>`\$({SupportedLangs}).\${(FooterLocaleIDs)}_id`;</code>					
<code>// "en_header_id"   "en_footer_id"</code>					
<code>  "pt_header_id"   "pt_footer_id"</code>					
<code>  "zh_header_id"   "zh_footer_id"</code>					

Figura 1.39: Tipos

<https://www.typescriptlang.org/cheatsheets>

#### 1.10.4 Frameworks

[7]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.

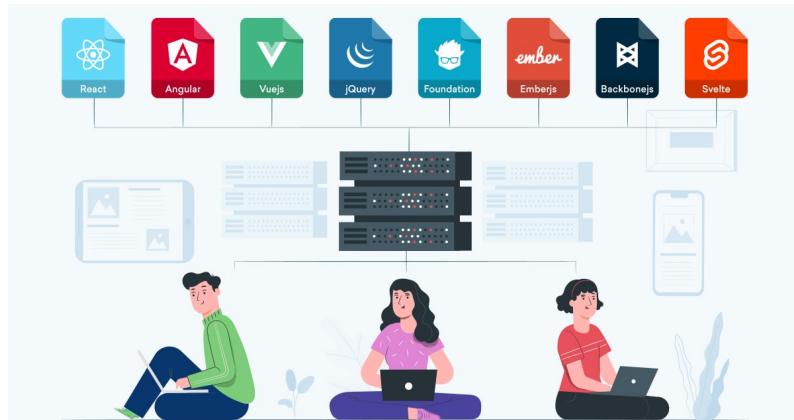


Figura 1.40: Algunos frameworks para frontend

## 1.11 Estructura de datos

### 1.11.1 JSON

### 1.11.2 XML

### 1.11.3 DTO

## 1.12 Localización e Internacionalización

[6] ¿Qué significan los términos "internacionalización" y "localización" como se relacionan?

Ofrecemos aquí algunas descripciones generales, de alto nivel, que reflejan el modo en que tendemos a utilizar estos términos en el sitio del W3C dedicado a la internacionalización.

### 1.12.1 Localización

Se entiende por localización la adaptación de un producto, una aplicación o el contenido de un documento con el fin de adecuarlos a las necesidades (lingüísticas, culturales u otras) de un mercado destinatario concreto (una "localidad" o "local" [locale]).

**R** La palabra localización a veces se escribe "l10n", donde 10 es la cantidad de letras entre la 'l' y la 'n'.

Aunque se la considera a menudo sinónimo de traducción de la interfaz de usuario y de la documentación, la localización suele ser un asunto considerablemente más complejo, que puede implicar la adaptación del contenido en relación con:

1. formatos numéricos, de fecha y de hora;
2. uso de símbolos de moneda;
3. uso del teclado;
4. algoritmos de comparación y ordenamiento;
5. símbolos, iconos y colores;
6. texto y gráficos que contengan referencias a objetos, acciones o ideas que, en una cultura dada, puedan ser objeto de mala interpretación o considerados ofensivos;
7. diferentes exigencias legales;
8. y muchas otras cuestiones.

La localización puede requerir incluso una reelaboración exhaustiva de la lógica, el diseño visual o la presentación, si la forma de hacer negocios (por ejemplo, las normas contables) o el paradigma aceptado de aprendizaje (por ejemplo, énfasis en el individuo o en el grupo) en la localidad de destino difieren mucho en relación con la cultura originaria.

### 1.12.2 Internacionalización

Existen diferentes definiciones para la palabra internacionalización. La que damos aquí es una definición operativa de alto nivel para usar con los materiales de la Actividad de internacionalización del W3C. Algunas personas utilizan otros términos para referirse al mismo concepto, por ejemplo, "globalización".

La internacionalización es el diseño y desarrollo de un producto, una aplicación o el contenido de un documento de modo tal que permita una fácil localización con destino a audiencias de diferentes culturas, regiones o idiomas.

-  La palabra internacionalización a veces se escribe "i18n", donde 18 es la cantidad de letras entre la i y la ene.

La internacionalización generalmente implica:

1. Un modo de diseño y desarrollo que elimine obstáculos a la localización o la distribución internacional. Esto incluye cuestiones tales como (entre otras) usar Unicode o asegurar, allí donde corresponda, un correcto tratamiento de las codificaciones de caracteres anticuadas; controlar la concatenación de cadenas; o evitar que la programación dependa de valores de cadenas pertenecientes a la interfaz de usuario.
2. Habilitar características que tal vez no sean usadas hasta el momento de la localización. Por ejemplo, añadir en la DTD etiquetas para habilitar el texto bidireccional o la identificación de idiomas. O hacer la CSS compatible con texto vertical u otras características tipográficas ajenas al alfabeto latino.
3. Preparar el código para hacer frente a las preferencias locales, regionales, lingüísticas o culturales. Por lo general, esto supone incorporar características y datos de localización predefinidos a partir de bibliotecas existentes o de las preferencias del usuario. Algunos ejemplos son: formatos de fecha y hora, calendarios locales, formatos y sistemas de números, ordenamiento y presentación de listas, uso de nombres personales y formas de tratamiento, etc.
4. Separar del código o contenido fuente los elementos localizables, de modo que puedan cargarse o seleccionarse alternativas localizadas según determinen las preferencias internacionales del usuario.

Obsérvese que esta lista no incluye necesariamente la localización del contenido, la aplicación o el producto hacia otro idioma; se trata más bien de prácticas de diseño y desarrollo que facilitan esa migración en el futuro, pero que también pueden tener una utilidad considerable aunque la localización jamás se produzca.

La internacionalización influye considerablemente en la facilidad de localización del producto. Obviamente, resulta mucho más difícil y demanda mucho más tiempo adaptar retrospectivamente un producto centrado en un idioma y cultura particulares que diseñarlo desde un primer momento con la intención de presentarlo mundialmente. (Piénsese en el Y2K y todo lo que demandó tratar de "deshacer" la opción por campos de fecha de dos caracteres, basada en el supuesto de que se interpretarían siempre como "19xx").

De modo que, en condiciones ideales, la internacionalización se da como un paso fundamental en el proceso de diseño y desarrollo, más que como un agregado posterior que, a menudo, puede implicar un difícil y costoso proceso de reingeniería.



# Bibliography

## Books

- [Ser02] Mora Sergio Luján. *Programación de aplicaciones web: historia principios básicos y clientes web.* 1.<sup>a</sup> 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 11, 13, 23-25, 28, 39, 41).
- [Ecm] Ecma-International. *Association dedicated to the standardization of information and communication systems.* URL: <https://www.ecma-international.org> (véase página 41).
- [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).
- [Typ] TypeScript. *TypeScript is JavaScript with syntax for types.* URL: <https://www.typescriptlang.org> (véase página 44).
- [W3o] W3.org. *Diferencias entre localizacion e internacionalizacion.* URL: <https://www.w3.org/International/questions/qa-i18n.es.html> (véase página 50).
- [Wik] Wikipedia. *La enciclopedia de contenido libre.* URL: <https://es.wikipedia.org/> (véanse páginas 8, 24, 40, 41, 49).

## Articles



# Índice alfabético

## A

Arquitectura cliente/servidor ..... 9

## E

El origen ..... 7

## P

Protocolos de internet.....7

## S

Separación de funciones ..... 10

Sobre la guía ..... 5