



**autentia**

**HTML Y CSS**

---

# **FRONT**

---

**PRIMERA PARTE**

Guía para directivos y técnicos

V.1

# Front

---

## HTML y CSS

Este documento forma parte de las guías de onboarding de Autentia. Si te apasiona el desarrollo de software de calidad ayúdanos a difundirlas y ánimate a unirte al equipo. Este es un documento vivo y puedes encontrar la última versión, así como el resto de partes que completan este documento, en nuestra web.

<https://www.autentia.com/libros/>



Esta obra está licenciada bajo la licencia [Creative Commons Attribution ShareAlike 4.0 International \(CC BY-SA 4.0\)](#)

Hoy en día el negocio está en la red. Es en el mercado on-line donde se producen la mayor parte de los intercambios comerciales entre clientes y proveedores. El primer contacto de nuestros usuarios con nuestro negocio, y en muchos casos el único, es a través de una aplicación web o móvil. No disponer de un diseño atractivo, una experiencia de usuario agradable, accesible y que se adapte de manera adecuada para ser usada en diferentes dispositivos (Responsive), es garantía de una pérdida masiva de potenciales clientes. De la misma manera, la forma en la

que se publican y organizan los contenidos, además del grado de usabilidad y accesibilidad de los mismos, influye directamente en el posicionamiento que los motores de búsqueda asignan a las aplicaciones.

## **“No hay una segunda oportunidad para una primera impresión”**

Alcanzar la habilidad de realizar diseños profesionales y usables no es algo baladí y se necesita un conocimiento profundo en marketing digital, experiencia de usuario y en tecnologías front-end. Proporcionar aplicaciones bien diseñadas inspira seguridad y confianza en nuestros clientes.

Y es que el cliente, está en el centro de todo esto. ¿Cuánto conocemos a nuestro usuario final?, ¿tenemos políticas de feedback y procesos de mejora?

Actualmente un 8% de la población mundial es daltónica y según la OMS hay entorno a un 5% de personas con discapacidad visual y un 5% con discapacidad auditiva. No tener nuestra plataforma adaptada a estas personas, es aproximadamente lo mismo que atender a gritos a los clientes de entre 30 y 40 años que entren a comprar en nuestro establecimiento.

# Front

## HTML y CSS

---

### Índice

- Las bases del desarrollo web
- HTML
  - Anatomía de un documento HTML
  - HTML semántico
  - Class / id
  - Elementos Void
  - DOM
  - HTML5
- Etiquetas HTML5
  - Etiquetas para el <head>
    - Title
    - Meta
    - Link
    - Style
  - Etiquetas para el <body>
    - Div
    - Span
    - Paragraph
    - Section
    - Aside
    - Main
    - Header y Footer
    - Headings
    - Strong

- Em
- Button
- Input
- Image
- Nav
- Details y Summary
- Blockquote
- Article
- Anchor
- Etiquetas especiales
  - Script
  - Noscript
- CSS
  - Selectores
    - Selector de tipo
    - Selector de clase
    - Selector de id
    - Selector universal
    - Selector de atributo
    - Combinador de hermanos adyacentes
    - Combinador general de hermanos
    - Combinador de hijo
    - Combinador de descendientes
  - Unidades de longitud
    - Unidades absolutas
    - Unidades relativas
  - Layouts
    - Flexbox
    - Grid
  - Custom properties
    - Declarando una variable
    - Utilizando una variable
  - Posicionamiento
    - Static

- Relative
- Absolute
- Diseño fluido
- Diseño responsive
- Accesibilidad
  - Soluciones tecnológicas (AT)
    - Visuales
    - Auditivas
    - Motrices
    - Cognitivas
  - Normativas y estándares
    - WCAG 2.1
  - Técnicas básicas de HTML y CSS
- Lecciones aprendidas
- Bibliografía

## Las bases del desarrollo Web

El HTML, o Lenguaje de Marcado de Hipertextos, es el lenguaje utilizado para **crear documentos en la web**. Proporciona significado (semántica) y estructura al contenido de una página web. Por ejemplo, sus contenidos podrían ser párrafos, una lista con viñetas o imágenes y tablas de datos.

"Hipertexto" se refiere a enlaces que conectan páginas web entre sí, ya sea dentro de un único sitio web o entre distintos sitios web. Los enlaces son un aspecto fundamental de la Web. Al cargar contenido en Internet y vincularlo a páginas creadas por otras personas, se convierte en un participante activo en la World Wide Web.

HTML utiliza "marcado" para anotar texto, imágenes y otro contenido para mostrar en un navegador web. El marcado HTML incluye "elementos" especiales. Un elemento HTML se separa de otro texto en un documento mediante "etiquetas", que consisten en el nombre del elemento rodeado por "<" y ">". El nombre de un elemento dentro de una etiqueta no distingue entre mayúsculas y minúsculas. Es decir, se puede escribir en mayúsculas, minúsculas o una mezcla.

*“HTML y CSS son dos de los pilares  
(junto con JavaScript) del desarrollo  
front-end.”*

¿Y qué es el CSS o Cascading Style Sheets? Se trata del lenguaje utilizado para la presentación (diseño o aspecto visual) del contenido de una página web. Utilizando CSS daremos estilo a cualquier tag de HTML.

**CSS**

**¿Qué es?**

CSS (Cascade Style Sheets) es un lenguaje de diseño gráfico que nos permite definir la apariencia visual de los elementos HTML que componen las interfaces web.

**¿EN QUÉ CONSISTE?**

Junto con HTML y JavaScript, CSS permite crear interfaces web y GUIs de dispositivos móviles visualmente atractivas.

CSS está pensado para definir el estilo de una página web, incluyendo diseño, layout, fuentes y variaciones en la interfaz, separándolo del contenido de ésta. Gracias al uso de estilos CSS, podremos dar distintas apariencias a una misma página HTML.

Los estilos CSS se definen dentro de las hojas de estilo (.css), aunque pueden incluirse directamente dentro del HTML. Normalmente las hojas de estilos son ficheros independientes de modo que puedan reutilizarse en distintas interfaces para dar una apariencia común.

Como cualquier lenguaje, CSS ha ido evolucionando con el paso del tiempo. La versión más reciente es la conocida como **CSS3**. Su especificación es mantenida por el World Wide Web Consortium (**W3C**).

**SINTAXIS**

El código CSS se compone de reglas. Una regla es el conjunto de propiedades que se van a aplicar a un elemento determinado.

En una regla distinguimos el selector y la declaración.

Regla CSS  
 Selector → Declaración  
 Section h2 {padding: 5px 0px; font-size: 24px; color: red}  
 ↓  
 Propiedad → Valor

El selector nos permite referenciar los elementos que se quieren modificar. Se clasifican en:

- Selector de etiqueta <section> y el selector section [...].
- Selector de clase <div class="relevant"> y el selector .relevant [...].
- Selectores de id <div id="cabecera"> y selector #cabecera [...].
- Selector descendente como el del esquema, aplicando estilo a todos los h2 incluidos en elemento section.

La declaración engloba un listado de pares propiedad-valor encerrado entre llaves {} y separados por punto y coma (;). Cada propiedad se separa de su valor por dos puntos (:).

CSS es un lenguaje de hojas de estilo utilizado para describir la presentación de un documento escrito en HTML o XML. Describe cómo se deben representar los elementos tanto visualmente, como de forma auditiva, por ejemplo, para los que sufren de alguna discapacidad visual y necesitan de alguna tecnología de asistencia de lectura de pantallas. CSS es una tecnología incluida dentro de la Web abierta y está estandarizado en todos los navegadores web de acuerdo con la especificación.



## Web Abierta. Open Web Platform

**autentia**

### ¿Qué es?

Open Web Platform es una colección de tecnologías abiertas y estándares desarrollados por organismos como W3C, Unicode Consortium, Internet Engineering Task Force, y Ecma International. Algunas de las tecnologías que cubre son HTML5, CSS, SVG, MathML, WAI-ARIA, ECMAScript, WebGL, etc.

#### PRINCIPIOS BÁSICOS

Open Web tiene los siguientes principios :

- **Evita ser controlado** por ninguna empresa u organización, ya que está descentralizado. Perteenece a cualquiera que quiera usarlo.
- **Aporta transparencia**, haciendo visibles todos los niveles desde el origen de documento hasta las URLs y las capas HTTP.
- **Capacidad de integración**. Debería ser posible integrar con facilidad y de manera segura una fuente externa de otro sitio.
- **Documentación y especificaciones abiertas**. Sin derechos de autor o patentes sobre las especificaciones.
- **Libertad de Uso**. Emplea las tecnologías abiertas tanto en los proyectos libres como privados.
- **Discurso abierto**. Fomentar el diálogo y la participación de millones de personas usando la web como hilo conductor.
- **Cadena de favores**. Ser integrante de la Open Web significa compartir lo aprendido con blogs, conferencias o el uso de tecnologías abiertas.

#### EL FUTURO Y SUS FUNDAMENTOS

Open Web Platform propone una taxonomía con ocho fundamentos en los que se centrará la próxima generación de aplicaciones. Cada fundamento representa un conjunto de servicios y capacidades disponibles para todas las aplicaciones. Los fundamentos a cubrir son:

- Seguridad y privacidad.
- Diseño y desarrollo web central.
- Interacción del dispositivo.
- Ciclo de vida de la aplicación.
- Medios y comunicaciones en tiempo real.
- Rendimiento y afinación.
- Usabilidad y accesibilidad.
- Servicios.

Open Web Platform Application Foundation	Seguridad y Privacidad	Usabilidad y Accesibilidad	Ciclo de Vida de la Aplicación	Servicios Comunes
	Identidad, cifrado del API, múltiples factores de autenticación	Contenido y Software Accesible, Internacionalización	Modo "Sin conexión", despliegues, geoposicionamiento, sincronización	Social, pagos, anotaciones, red de datos
	Rendimiento y Optimización	Medios y Comunicación en tiempo real	Interacción con el dispositivo	Diseño y Desarrollo Web Esencial
	Perfilado, mejoras, diseño flexible	WebRTC, transmisión de medios, multipantalla	Sensores, orientación, vibración, pantallas táctiles, bluetooth, etc.	Composición, estilo, HTML, animaciones, tipografía

Podemos pensar en HTML, CSS y JavaScript como si fueran el cuerpo humano. El HTML sería el esqueleto, que es la estructura del cuerpo, JavaScript sería los músculos, lo que proporciona movimiento y flexibilidad, y CSS sería la piel, el cabello, los ojos y todo lo que se corresponde con la estética visual.

# HTML

HTML (Hyper Text Markup Language) es un lenguaje markup que sirve para describir la estructura de una página web, no el diseño, colores, etc., sino sólo la estructura. Se basa en etiquetas. Por ejemplo, si queremos utilizar un espacio donde haya algún tipo de contenido, como puede ser simplemente texto, podríamos hacerlo así:

```
<div> Este texto se mostraría en la página </div>
```

Dentro de las etiquetas puede haber otras etiquetas. Podríamos, por ejemplo, tener dos párrafos dentro de un `<div>`:

```
<div>
  <p> Este es el primer párrafo </p>
  <p> Este es el segundo párrafo </p>
</div>
```

**HTML**

## ¿Qué es?

HTML (Hyper Text Markup Language) es un lenguaje markup que sirve para describir la estructura de una página web, no el diseño, colores, etc., sino sólo la estructura haciendo uso de etiquetas para organizar el contenido.

### HISTORIA Y EVOLUCIÓN

El HTML fue inventado por **Tim Berners-Lee** (físico del CERN) en 1989. Se le ocurrió la idea de un sistema de hipertexto (enlaces a contenido) en Internet. Necesitaba **crear documentos con referencias a otros para facilitar el acceso y la colaboración** de otros equipos. Desde esos días hasta hoy, lo que conocemos como HTML (HTML4 publicada W3C en 1999) ha tenido una evolución increíble en su última versión HTML5 (publicada en 2014), introduciendo nuevas características como etiquetas semánticas, incrustar audio y vídeo o soportar SVG y MathML para fórmulas matemáticas.

### ¿CÓMO FUNCIONA?

El contenido se guarda en archivos con extensión .html o .htm y se ve a través de cualquier navegador. Cada página HTML consta de un **conjunto de etiquetas**, que representan los componentes básicos de la página web. Con ellos creamos una jerarquía que **estructura el contenido** en secciones, párrafos, encabezados y otros bloques de contenido. La mayoría de los elementos HTML tienen **una apertura y un cierre** que utilizan la sintaxis <tag> </tag>. Un ejemplo de estructura HTML podría ser:

```
<div>
  <p> Este es el primer párrafo </p>
  <p> Este es el segundo párrafo </p>
</div>
```

### VENTAJAS E INCONVENIENTES

**Ventajas:**

- Lenguaje ampliamente utilizado por la comunidad.
- Se ejecuta de forma nativa en todos los navegadores.
- Lenguaje limpio, consistente y sencillo.
- La especificación sigue un estándar mantenido por la W3C.
- Tiene una fácil integración con lenguajes Backend.

**Inconvenientes:**

- Páginas estáticas. Necesitando de JavaScript para hacerlas dinámicas..
- No permite implementar lógica o comportamiento.

HTML necesita de CSS y JavaScript para implementar algunas funciones. Se puede pensar en HTML como el esqueleto de un persona, CSS como la piel, pelo, etc. y JavaScript los músculos.

HTML es un lenguaje de marcado “hermano” de XML y XHTML. Todos extienden de **SGML** (Standard Generalized Markup Language o lenguaje de marcado generalizado estándar), pero cada uno está pensado para un propósito diferente:

- **XML:** se utiliza para definir un conjunto de reglas para codificar documentos en un formato que sea comprensible tanto para humanos como para máquinas. Es una restricción de SGML que no permite etiquetas sin cerrar (deben estar cerradas con su correspondiente etiqueta de cierre o, en el caso de ser de autocierre, con la terminación “/>”), junto con otras restricciones más. Una de sus tantas aplicaciones es, por ejemplo, el intercambio de información entre sistemas.

- **HTML:** restringe SGML definiendo una lista de etiquetas que se permite utilizar y sólo es adecuado para la representación de páginas web.
- **XHTML:** XHTML restringe SGML con las etiquetas de HTML (con algunas exclusiones, como frameset y otras), agregando además, las restricciones de etiqueta y entidad de XML. Es, por lo tanto, un documento HTML con especificaciones adicionales para cumplir con el estándar XML.

## Anatomía de un documento HTML

El siguiente código es sencillo, pero nos permitirá mostrar la estructura básica de un documento HTML:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="iso-8859-1">
    <title>Página de prueba</title>
  </head>
  <body>
    
  </body>
</html>
```

Aquí podemos encontrar los siguientes elementos:

- `<!DOCTYPE html>`: todo documento HTML debe comenzar con una declaración Doctype. Su función es informar a los navegadores de qué tipo de documento están procesando. En HTML 5 la declaración es tan sencilla como la del ejemplo, pero en versiones anteriores esta

era más complicada porque debía referir a un fichero DTD (Document Type Definition).

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

- **<html>**: es el elemento raíz del documento HTML y envuelve todo el contenido de la página.
- **<head>**: este elemento sirve como contenedor para todo aquello que necesitemos incluir en nuestra página pero que no deseemos mostrar a los usuarios. Toda esta información es conocida también como metadatos (información sobre la información) y no se muestra a los usuarios. Aquí es donde solemos definir el título de la página, los íconos, el conjunto de caracteres, los estilos, scripts y más. Sólo puede haber un elemento head por documento HTML.
- **<body>**: este elemento envuelve a todo el contenido que deseamos mostrar a los usuarios cuando visitan nuestra página como texto, imágenes, audio, vídeo, etc. Sólo puede haber un elemento body por documento HTML.

## HTML semántico

El HTML semántico es aquél que introduce significado o la semántica a la página, no sólo la presentación. Por ejemplo, la etiqueta **<b>** (negrita) o **<i>** (cursiva) no son semánticas ya que definen cómo se debería mostrar el texto, pero no aportan ningún significado.

También, si utilizamos un **<div>**, este no aporta nada de información ya que podría contener cualquier cosa y mostrarse de cualquier forma,

dependiendo del CSS. Sin embargo, si por ejemplo utilizamos `<p>`, estamos indicando que lo que hay entre estas etiquetas será un párrafo. Las personas saben lo que es un párrafo y los navegadores saben cómo representarlo, por tanto, `<p>` sí es una etiqueta semántica pero `<div>` no lo es.

También, por ejemplo, si utilizamos una etiqueta `<h1>` para envolver un texto, estamos dándole significado, estamos indicando que ese es un título principal en la página por tanto, también es una etiqueta semántica.

## Class / id

Class e id son formas de identificar un elemento HTML. Class se utiliza para referenciar el elemento desde un archivo CSS y poder darle un estilo. El atributo global id define un identificador único, el cual no debe repetirse en todo el documento y cuyo propósito es identificar el elemento para poder hacer referencia al mismo, tanto en scripts como en hojas de estilo.

En el documento de CSS se pueden referenciar elementos de cualquiera de estas dos formas, entre otras. La diferencia entre una clase y un id, es que con id se puede identificar un sólo elemento, mientras que la clase se puede usar para identificar más de uno. Así es como se utilizarían en el HTML:

```
<p class="class-example"> Este es el ejemplo con clase </p>
<p id="id-example"> Este es el ejemplo con id </p>
```

## Elementos Void

En HTML existen los **Void Elements** o elementos “vacíos”. Estos elementos solo tienen etiqueta de apertura ya que no tienen contenido y **no deben**

**tener etiqueta de cierre.** Ejemplos de este tipo de elementos son las etiquetas `<input>`.

Ejemplo:

```
<input type="number" placeholder="Introduzca un número">
```

O

```
<input type="number" placeholder="Introduzca un número"/>
```

son correctos, mientras que

```
<input type="number" placeholder="Introduzca un número"></input>
```

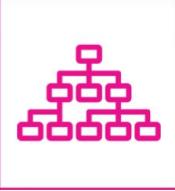
es incorrecto.

## DOM

**DOM (Document Object Model)** es la interfaz que permite que lenguajes de programación como JavaScript, modifiquen la estructura, el estilo o el contenido de un sitio web.

El modelo nos ofrece una abstracción que nos **permite tratar el documento y sus elementos como si fuesen objetos**. Para ello, se construye lo que se conoce como un “**Árbol de DOM**” en el que cada elemento del documento es un nodo.

**Árbol DOM**



### ¿Qué es?

DOM (Document Object Model) es una interfaz para documentos HTML y XML que se representa como un **árbol de elementos**. Permite leer y manipular el contenido, la estructura y los estilos de la página con un lenguaje de scripting como JavaScript.

### RENDER TREE

La forma en que un navegador pasa de un documento HTML, a mostrar una página con estilo e interactiva, se denomina **Critical Rendering Path**. Primero se establece que se va a renderizar y se denomina **render tree (DOM + CSSOM)**. Luego, el navegador realiza el renderizado.

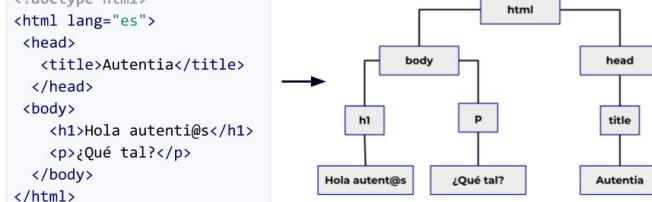
- CSSOM: representa los estilos asociados a los elementos.
- DOM: representa los elementos.

El render tree excluye los elementos que no están visibles como por ejemplo, los que tienen el estilo `display: none`. **El DOM si lo incluiría en su árbol de nodos**.

### ÁRBOL DE NODOS

El objetivo del DOM es convertir la estructura y el contenido del documento HTML en un modelo de objeto que puede ser utilizado por varios programas. La estructura del documento es conocida como un **árbol de nodos (node tree)**. El elemento raíz es la etiqueta `html`, las ramas son los elementos anidados y las hojas serían el contenido de esos elementos.

```
<!doctype html>
<html lang="es">
  <head>
    <title>Autentia</title>
  </head>
  <body>
    <h1>Hola autenti@s</h1>
    <p>¿Qué tal?</p>
  </body>
</html>
```



El DOM nos ofrece una multitud de propiedades y métodos para acceder a los diferentes elementos y poder modificarlos. Algunos de los métodos más utilizados son `getElementById()` o `getElementsByClassName()`. Ellos nos permiten acceder a los elementos del DOM indicando su atributo `id` y `class` respectivamente, similar al uso de selectores en HTML vistos en apartados anteriores.

Esto nos permite consultar y modificar los atributos de cada elemento. Algunos ejemplos de esto serían: extraer el texto introducido en una caja de texto o cambiar el estado de un elemento, deshabilitando un botón. Esta capacidad de manipular el DOM, nos ofrece muchas posibilidades como crear aplicaciones personalizadas que cambien su diseño sin necesidad de actualizar la página.

El DOM forma parte del árbol de renderizado, parte fundamental dentro del

ciclo de vida de una página web, representando todos sus componentes visibles en píxeles.



### Ciclo de vida de una página web

#### ¿Qué es?

También referido como **Critical Rendering Path** (ruta de renderizado crítico) es la **secuencia de pasos** que sigue el navegador para convertir **HTML, CSS y JavaScript** en **píxeles en la pantalla**. Esta secuencia de pasos es realizada por el motor de renderizado del navegador.

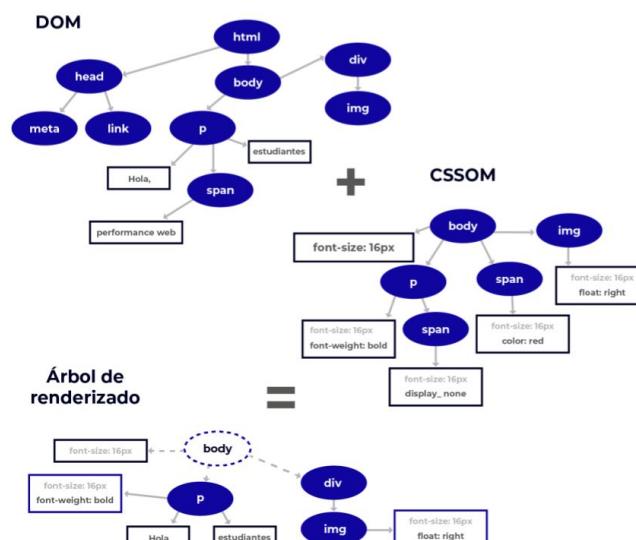
**autentia**

---

 **¿EN QUÉ CONSISTE?**

Una solicitud de una página web o aplicación comienza con una petición HTML. Al realizar una solicitud, el servidor devuelve los encabezados y datos de respuesta.

1. Se crea el **Document Object Model (DOM)** a partir de la respuesta HTML. También inicia solicitudes cada vez que encuentra enlaces a recursos externos, ya sean hojas de estilo, scripts o referencias de imágenes incrustadas.
  - a. Algunas solicitudes son bloqueantes, lo que significa que el parseo del resto del HTML se detiene hasta que se cargue el recurso.
2. Se crea el **CSS Object Model (CSSOM)**.
3. Cuando tiene el DOM y el CSSOM listos, se combinan en el **árbol de renderizado** (Render Tree), obteniendo los estilos para todo el contenido visible.
4. Una vez que se completa el árbol de procesamiento, el diseño calcula la posición y el tamaño exactos de cada objeto (**layout**) del árbol de procesamiento.
5. Una vez completado, se procede a la **representación o "pintado"**, que consiste en tomar el árbol de renderizado final y renderizar los píxeles en la pantalla.



**DOM**

```

graph TD
    head --- meta
    head --- link
    head --- body
    body --- p
    body --- div
    div --- img
    p --- span
    span --- estudiantes
    span --- Hola
    
```

**CSSOM**

```

graph TD
    body["font-size: 16px"] --- p
    body --- img
    p --- span1["font-size: 16px; font-weight: bold"]
    p --- span2["font-size: 16px; color: red"]
    span1 --- estudiantes
    span1 --- Hola
    span2 --- div
    
```

**Árbol de renderizado**

```

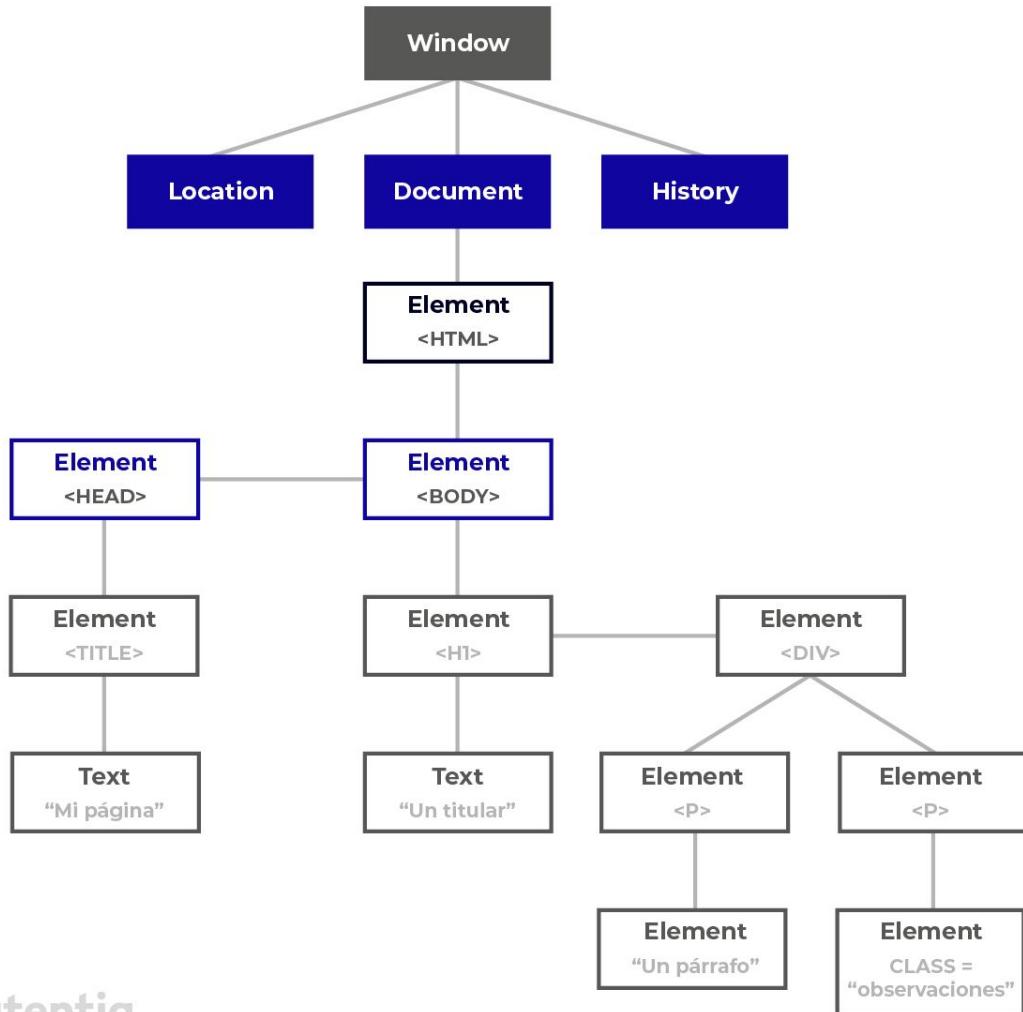
graph TD
    body["font-size: 16px; font-weight: bold"] --- p
    body --- div
    p --- span1["font-size: 16px; font-weight: bold"]
    p --- span2["font-size: 16px; color: red"]
    span1 --- estudiantes
    span1 --- Hola
    div --- img
    
```

**performance web**

El DOM no sólo contiene representaciones visibles, también contiene eventos que se pueden producir en el documento o sus elementos, como arrastrar, clicar o teclear, entre otros.

También dispone de objetos que nos ofrecen metainformación sobre el navegador (objeto `navigation`), la pantalla (objeto `screen`), la url (objeto `location`) y más. Al árbol que contiene alguno de los objetos mencionados anteriormente, como los que representan componentes del navegador, se le llama BOM (“Browser Object Model”).

Una representación del árbol BOM creado después de leer el documento con todos sus objetos, podría ser el siguiente:



autentia

## HTML5

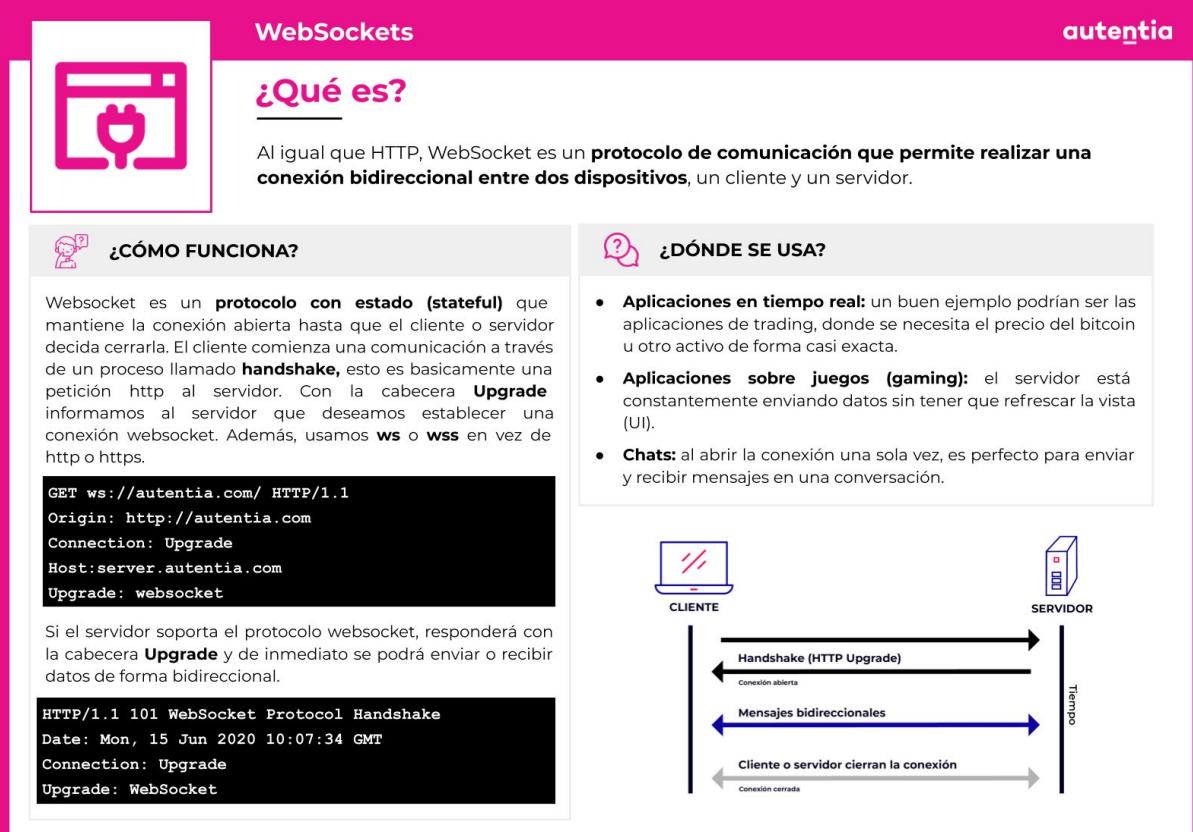


HTML5 es la última evolución del estándar que define HTML. El término representa dos conceptos diferentes. Es una nueva versión del lenguaje HTML, con nuevos elementos, atributos y comportamientos y un conjunto más amplio de tecnologías que permiten la creación de sitios web y aplicaciones más diversas y potentes.

Entre las nuevas características que introduce la nueva versión,

encontramos:

- » **Semántica:** con HTML5 se incluye un conjunto más rico de etiquetas, junto con RDFa, microdatos y microformatos, que permiten una web más útil basada en datos para los programas y sus usuarios.
- » **Fuera de línea y almacenamiento:** las aplicaciones inician más rápido gracias a la caché de aplicaciones HTML5, así como al almacenamiento local, la base de datos indexada y las especificaciones de la API de archivos.
- » **Acceso a dispositivos:** se han implementado innovaciones como la API de geolocalización, el acceso a dispositivos, el acceso de entrada de audio / vídeo a micrófonos y cámaras, datos locales como contactos y eventos e incluso, orientación de inclinación.
- » **Conectividad:** más eficiente y rápida, y una mejor comunicación. Web Sockets y Server-Sent Events, sirven para enviar datos entre el cliente y el servidor de manera más eficiente.



**WebSockets**

**¿Qué es?**

Al igual que HTTP, WebSocket es un **protocolo de comunicación que permite realizar una conexión bidireccional entre dos dispositivos**, un cliente y un servidor.

**¿CÓMO FUNCIONA?**

WebSocket es un **protocolo con estado (stateful)** que mantiene la conexión abierta hasta que el cliente o servidor decida cerrarla. El cliente comienza una comunicación a través de un proceso llamado **handshake**, esto es básicamente una petición http al servidor. Con la cabecera **Upgrade** informamos al servidor que deseamos establecer una conexión websocket. Además, usamos **ws** o **wss** en vez de http o https.

```
GET ws://autentia.com/ HTTP/1.1
Origin: http://autentia.com
Connection: Upgrade
Host:server.autentia.com
Upgrade: websocket
```

Si el servidor soporta el protocolo websocket, responderá con la cabecera **Upgrade** y de inmediato se podrá enviar o recibir datos de forma bidireccional.

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Mon, 15 Jun 2020 10:07:34 GMT
Connection: Upgrade
Upgrade: WebSocket
```

**¿DÓNDE SE USA?**

- **Aplicaciones en tiempo real:** un buen ejemplo podrían ser las aplicaciones de trading, donde se necesita el precio del bitcoin u otro activo de forma casi exacta.
- **Aplicaciones sobre juegos (gaming):** el servidor está constantemente enviando datos sin tener que refrescar la vista (UI).
- **Chats:** al abrir la conexión una sola vez, es perfecto para enviar y recibir mensajes en una conversación.

Diagrama que ilustra el flujo de datos entre un cliente y un servidor:

- El cliente envía un "Handshake (HTTP Upgrade)" al servidor.
- El servidor responde con "Mensajes bidireccionales".
- Finalmente, el cliente o servidor cierra la conexión.

**Multimedia:** se añade un mejor soporte sobre contenido multimedia, permitiendo la reproducción de audio y vídeo sin necesidad de componentes o plugins adicionales y permitiendo que la página reproduzca de forma nativa dicho contenido. De esta forma, las páginas tardan menos en cargar y su navegación es más rápida.

**3D, gráficos y efectos:** funciones 3D de SVG, Canvas, WebGL y CSS3 para imágenes representadas de forma nativa en el navegador.

**Performance e integración:** implementación de tecnologías como Web Workers y XMLHttpRequest 2 para aplicaciones y contenido web dinámico más rápidos.

Web workers

## JavaScript asíncrono

Los web workers son scripts escritos en JavaScript que se ejecutan de forma paralela y en segundo plano al procesamiento de la interfaz gráfica. Podemos calcular o solicitar información sin que el usuario perciba una interrupción visual o funcional.

### CREACIÓN

Para crear un web worker se utiliza el constructor, asígnale el objeto creado a una variable. Se puede hacer dentro de cualquier script de JavaScript.

```
let myWorker = new Worker('worker.js');
```

El Worker utilizará el script 'worker.js' para su funcionamiento.

```

graph TD
    Cliente((Cliente)) -- "new Worker('worker.js') -->|<br/>"--> WebWorker[Web Worker]
    WebWorker -- "postMessage('Hola web worker') -->|<br/>"--> onmessage1[onmessage(m)...]
    onmessage1 -- "postMessage('Hola cliente') -->|<br/>"--> Cliente
    onmessage1 -- "onmessage(m)..." --> Cliente
  
```

### COMUNICACIÓN

Una vez inicializado, tanto el worker como su cliente se comunicarán a través del método `postMessage` y el manejador de eventos `onmessage`.

**client.js**

```
myWorker.postMessage('Hola web worker');
myWorker.onmessage = function(message) {
  console.log('Worker dice: ' + message);
}
```

**worker.js**

```
onmessage = function(message) {
  console.log('Cliente dice: ' + message);
  postMessage('Hola cliente');
}
```

### LIMITACIONES

1. Los elementos del DOM no son manipulables dentro del contexto de un web worker.
2. El estándar que define la API para web workers considera su inicialización como “relativamente costosa”, con lo cual se debe cuidar de no crear muchos.

**CSS3:** CSS3 ofrece una amplia gama de estilización y efectos, mejorando las aplicaciones web sin sacrificar su estructura semántica o rendimiento. Además, Web Open Font Format (WOFF) proporciona flexibilidad tipográfica y control, mucho más allá de lo que la web ha ofrecido antes.

# Etiquetas HTML5

Estas son algunas de las etiquetas HTML5 más utilizadas:

## Etiquetas para el <head>

### Title

Esta etiqueta sirve para definir el título del documento. El título debe ser solo de texto y se muestra en la barra de título del navegador o en la pestaña de la página. Esta etiqueta es requerida en todos los documentos HTML.

El contenido del título de una página es muy importante para la optimización de motores de búsqueda (SEO). Los algoritmos de los motores de búsqueda utilizan el título de la página para decidir el orden cuando se enumeran las páginas en los resultados de búsqueda.

```
<head>
  <title>Este es el título de la página</title>
</head>
```

**SEO**

**autentia**

**¿Qué es?**

**Search Engine Optimization (SEO)** o en español, **optimización en motores de búsqueda**, es un proceso que pretende mejorar el posicionamiento de una web en los resultados de los motores de búsqueda, como Google o Bing.

**CONCEPTO**

Search Engine Optimization (SEO) es un conjunto de técnicas para la optimización del posicionamiento en buscadores. Mediante el SEO, un sitio web aparece en más resultados naturales y se aumenta la calidad y cantidad del tráfico. Puede optimizarse el resultado en búsquedas de imágenes, videos, artículos académicos, compras, etc.

Hay que diferenciar los **resultados "orgánicos" o "naturales"**, que se consiguen porque el motor de búsqueda considera que son relevantes a la búsqueda del usuario, de los resultados pagados que son campañas de marketing dirigidas a un público.

La optimización tiene dos partes:

- Optimización interna: se trabaja tanto con **elementos técnicos de la web** (estructura HTML y metadatos), **como con el contenido interno** para hacerlo más relevante al usuario.
- Optimización externa: se mejora la **notoriedad de la página** web al aparecer referencias a ella en otros sitios (enlaces naturales y redes sociales).

**¿CÓMO FUNCIONAN LOS BUSCADORES?**

Los motores de búsqueda recorren los sitios mediante **arañas web**, navegando a través de las páginas y analizando su estructura y contenido. Las arañas solo analizan un número determinado de páginas (o se detienen un tiempo máximo) dentro del sitio, antes de pasar al siguiente. Los motores de búsqueda recorren cada sitio de forma periódica para mantenerse actualizados.

Una vez las arañas han analizado un sitio, lo indexan, clasificándolo según su contenido y relevancia. A partir de este índice, el motor de búsqueda va a poder mostrar la página en los resultados.

Además de este análisis, los buscadores **priorizan resultados que otros usuarios con un perfil similar les han resultado útiles**.

```

graph LR
    SW[Sitios Web] <--> AW[Arañas Web]
    AW --> BD[Base de Datos]
    BD <--> MS[Motor de Búsqueda]
    MS --> R[Resultados]
    R --> U[Usuario]
    U <--> B[Búsqueda]
  
```

Sitios Web      Arañas Web      Base de Datos      Motor de Búsqueda      Resultados      Usuario

Búsqueda

## Meta

Define información (metadata) sobre un documento HTML como el autor, la descripción de la página, la codificación de los caracteres, la configuración del viewport, etc.

Algunos valores y/o propiedades más usadas y conocidas dentro de esta etiqueta son las siguientes:

- **Description:** define una breve descripción de nuestra web y es la que usan los motores de búsqueda.

```
<meta name="description" content="HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies besides HTML are generally used to describe a web page.">
```

## HTML: Hypertext Markup Language | MDN

HTML (HyperText Markup Language) is the most basic building block of the Web. It defines the meaning and structure of web content. Other technologies ...

- **Keywords:** define palabras clave sobre la página web.

```
<meta name="keywords" content="HTML, CSS">
```

- **Viewport:** define el área visible de un usuario en una web. `device-width` adapta el ancho de la página al dispositivo, aunque también se podría asignar un tamaño fijo en píxeles (no recomendado). También tiene propiedades para limitar el zoom que puede hacer el usuario.

```
<meta name="viewport" content="width=device-width,  
initial-scale=1.0">
```

- **Charset:** define la codificación de los caracteres del documento.

```
<meta charset="utf-8">
```

## Link

Define la relación entre el documento actual y un recurso externo. La relación que tienen ambos se define con la propiedad rel (relationship), pero se enlazan a través de la propiedad href. La propiedad rel es obligatoria y sus valores más usados son stylesheet e icon, aunque tiene muchos más.

El uso más común de esta etiqueta es para enlazar hojas de estilos.

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

## Style

La etiqueta style se utiliza para aplicar una hoja de estilos simple a un documento HTML. Últimamente no se suele utilizar ya que los estilos se suelen definir en un fichero aparte del HTML, mientras que esta etiqueta sirve para definir el CSS dentro del documento HTML. Por ejemplo:

```
<html>
  <head>
    <style>
      h1 {color:red;}
      p {color:blue;}
    </style>
  </head>
  <body>
    <h1>A heading</h1>
    <p>A paragraph.</p>
  </body>
</html>
```

## Etiquetas para el <body>

### Div

Estas etiquetas se utilizan a modo de contenedores ya sea para texto, para otras etiquetas o cualquier otro tipo de contenido. Se utilizan de la siguiente manera:

```
<div>Este es un contenido dentro del div</div>
```

Esta etiqueta no es semántica y no es recomendable abusar de ella. Cuando sea posible, se debería sustituir por otras etiquetas que sí sean

semánticas.

## Span

El span es un contenedor inline, es decir, se utiliza para envolver parte de un texto o un documento. Esto puede ser para darles a esos elementos un estilo distinto (aplicando una clase o id) o porque comparten un atributo, por ejemplo, lang (lenguaje). Es básicamente como un div, con la diferencia de que div es un elemento a nivel de bloque, mientras que span es inline. Este es un ejemplo de cómo usarlo:

```
<p>
  En esa empresa hay <span class="workers-number"> 25 </span>
  trabajadores
</p>
```

Y en la clase de CSS “workers-number” tendríamos los estilos que le queremos dar únicamente al número que representa la cantidad de trabajadores.

**Block vs. Inline vs. Inline-block**

**autentia**

**¿Qué son?**

Las propiedades block, inline e inline-block **modifican cómo el cuadro de un elemento HTML se muestra** en la página. Cada elemento HTML tiene un valor de display por defecto, aunque se puede sobreescribir.

**DIFERENCIAS**

La propiedad CSS **display** tiene dos funciones: cómo el cuadro del propio elemento se muestra y cómo se muestran los hijos del elemento. Para la primera de ellas, tenemos los siguientes valores:

- **Block:** un elemento block siempre empieza en una nueva línea y toma todo el ancho disponible. Un ejemplo de una etiqueta block es div.
- **Inline:** un elemento inline no empieza en una nueva línea y solo trata de ocupar el ancho necesario. Un ejemplo de una etiqueta inline es span.
- **Inline-block:** la diferencia con inline es que inline-block permite establecer un ancho y altura en el elemento. Además se respetan los valores de margin/padding del elemento. La diferencia con block es que no se añade un salto de línea después del elemento, de forma que puede tener elementos a continuación.

Otro valor común es **none**, que hace que el elemento no se muestre y no ocupe espacio en la página.

## Paragraph

Esta etiqueta se utiliza a la hora de mostrar un párrafo y este sería un ejemplo de uso:

```
<p> Este es un párrafo </p>
```

## Section

Su uso es bastante directo: definir una sección. Un ejemplo de una sección podría ser este:

```
<section>
  <h2> Este sería el título de la sección </h2>
  <p> Este es un párrafo </p>
</section>
```

## Aside

Esta etiqueta define cierto contenido aparte del contenido en el que se coloca. Este contenido aparte debería estar indirectamente relacionado con el contenido que le rodea:

```
<section>
  <p>El verano pasado fuimos a una playa increíble en Cádiz.</p>
  <aside>
    <h4>Playa de Cortadura</h4>
    <p>Un texto explicativo sobre la playa</p>
  </aside>
  <p>Aparte de la playa, fuimos a cenar a unos cuantos sitios, a
    las ferias....</p>
</section>
```

## Main

Con esta etiqueta se define el contenido principal del documento o página. Lo que contenga esta etiqueta debe ser único en el documento y no tener elementos que se repitan en otros, como puede ser una barra de navegación, un menú lateral, etc.

```
<main>
  Aquí iría el contenido
</main>
```

## Header y Footer

Estas etiquetas, como su propio nombre indica, se usan para definir la cabecera y pie de de página respectivamente:

```
<header>Esta es una cabecera</header>
<footer>Esto es un pie de página</footer>
```

## Headings

Los encabezamientos, comúnmente utilizados como títulos, se representan con las etiquetas de la `<h1>` a la `<h6>`, siendo la primera la correspondiente al formato de encabezado con mayor importancia y, por lo general, con mayor tamaño de fuente, entre otras cosas. Los encabezados `<h6>` son los más pequeños. Para utilizar estas etiquetas solo hay que envolver un texto entre ellas:

```
<h1>Esto es un título</h1>
```

## Strong

La etiqueta `<strong>` se usa para definir un texto (o una parte de él) que sea importante. Tradicionalmente, el navegador aplicará negrita a ese texto, aunque el efecto de esta etiqueta se puede cambiar por medio de CSS. Su uso es muy sencillo:

```
<p>Así es como se define la parte <strong>importante<strong/> de  
una frase<p/>
```

## Em

Esta etiqueta se utiliza para enfatizar texto. El texto dentro de esta etiqueta `<em>` normalmente se muestra en itálicas aunque, al igual que con la etiqueta `strong`, se puede cambiar este comportamiento con CSS. Su uso es exactamente igual que `strong`.

## Button

Esta etiqueta se utiliza para crear un botón en el que podemos escuchar al evento del click y decirle a qué función debe llamar en ese caso:

```
<button onclick="handleonClick()">Click me</button>
```

## Input

El input se usa para crear un campo en el que un usuario pueda ingresar información, como por ejemplo un texto. Entre las muchas propiedades que tiene el input, se puede especificar el tipo de input que es con type (texto, número, fecha...) o definir un texto que se mostrará si el input está en blanco con placeholder. Input es un **elemento de tipo Void**, por lo que este elemento no debe cerrarse con otra etiqueta, como en otros casos.

Este sería un ejemplo de un input en el que sólo se pueden introducir números y con un placeholder:

```
<input type="number" placeholder="Introduzca un número">
```

## Image

Este sería un ejemplo de cómo representar una imagen:

```

```

En src también se puede poner la url de una imagen de Internet. Alt es un texto alternativo que se muestra cuando la imagen no se ha podido mostrar por alguna razón.

## Figure

Sirve para identificar contenido como diagramas, ilustraciones, etc. Hace uso de `<figcaption>` para determinar el texto identificativo. Así es como se utilizaría

```
<figure>
  
  <figcaption>Fig. 1 - Gráfica progreso mensual<figcaption/>
</figure>
```

## Nav

Esta etiqueta se usa para definir una serie de links navegables. No todos los links de una página web deben estar en un nav pero si hay una serie de links juntos, deberían contenerse dentro de un nav:

```
<nav>
  <a href="link.com">Link 1</a> |
  <a href="some-link.com">Link 2</a> |
  <a href="another-link.com">Otro link</a> |
  <a href="yet-another-link.com">Random Link</a>
</nav>
```

## Details y Summary

La etiqueta details crea un desplegable interactivo que el usuario puede abrir y cerrar. Por defecto, comienza cerrado. El texto o elemento que se mostrará, incluso cuando el desplegable esté cerrado, deberá estar entre las etiquetas `<summary>` y éste debe ser el primer elemento dentro de `<details>`:

```
<details>
  <summary>Playa de Cortadura</summary>
  <p>Un texto explicativo sobre la playa</p>
</details>
```

En este caso, “Playa de Cortadura” se mostrará siempre y el párrafo de debajo se mostrará sólo cuando el elemento esté desplegado.

## Blockquote

Esta etiqueta se usa para citar un texto o sección de otra fuente. Hay que especificar la fuente en la propiedad cite:

```
<blockquote cite="https://www.typescriptlang.org/">
    TypeScript is an open-source language which builds on
    JavaScript, one of the world's most used tools, by adding
    static type definitions.
</blockquote>
```

## Article

Esta etiqueta especifica contenido independiente y auto-contenido. Este contenido debería tener sentido por su cuenta y poder ser distribuido de forma independiente al resto de la página. Las etiquetas article no hacen que su contenido se renderice de forma distinta, sino que aporta meramente un significado contextual. Un pequeño ejemplo puede ser el siguiente:

```
<article>
    <h2>Algún título</h2>
    <p>Un texto cualquiera</p>
</article>
```

## Anchor

La etiqueta `<a>` define un hipervínculo que se utiliza para enlazar a otra sección dentro del mismo documento, de una página a otra, dentro del mismo sitio o a un sitio diferente. El atributo más importante del elemento `<a>` es el atributo `href` que indica el destino del enlace.

```
<a href="https://www.autentia.com">Visita nuestra página</a>
```

El atributo href se utiliza para señalar la URL a la que apunta el link o enlace. Estos links pueden contener URLs relativas o absolutas. Una URL absoluta es aquella que contiene toda la información necesaria para localizar un recurso, mediante el formato *scheme://server/path/resource*, mientras que una URL relativa generalmente consiste del *path* y, opcionalmente, el *resource*, pero sin *scheme* o *server*, ya que estos los toma de la URL base desde la que está partiendo (en este caso, la del documento HTML). Pero los links no están restringidos a URLs basadas en el protocolo HTTP, si no que se puede utilizar cualquier esquema URL soportado por los navegadores.

Por ejemplo:

- Secciones de una página mediante fragmentos de URLs. Un fragmento de URL es una cadena de caracteres que refiere a un recurso que está subordinado o incluido en otro. Estos fragmentos son opcionales y se suelen marcar precediéndolos mediante el carácter #.

```
<a href="https://www.autentia.com#ejemplos">Visita nuestros  
ejemplos</a>
```

Estos fragmentos también se pueden utilizar para enlazar a secciones dentro del mismo documento mediante el uso de otro tag `<a>` con el atributo id definido:

```
<a id="top">Bienvenidos</a>  
<a href="#top">Volver arriba</a>
```

- Índices temporales específicos o segmentos dentro de ficheros de

medios mediante fragmentos de medios. Estos fragmentos de URL se utilizan para enlazar a una posición específica o a un fragmento de tiempo, a un fichero de audio o vídeo o a una porción de una imagen, etc.

Por ejemplo, el siguiente enlace nos llevará al minuto 30 del vídeo enlazado:

```
<a href="https://youtu.be/0QtS400I8m8?t=1800">Desarrollo de Juegos  
en JAVA</a>
```

- Números telefónicos mediante el uso del esquema `tel:` en la URL.

```
<a href="tel:+34916753306">Llámenos!</a>
```

- Direcciones de email con el esquema `mailto:` en la URL.

```
<a href="mailto:info@autentia.com">Contáctenos!</a>
```

## Etiquetas especiales

Estas etiquetas tienen la particularidad de que pueden incluirse tanto dentro de la etiqueta `<body>`, como del `<head>` y algunas, incluso también dentro de la etiqueta `<html>`.

### Script

La etiqueta `<script>` permite integrar código ejecutable en el lado del cliente (normalmente JavaScript).

```
<!DOCTYPE html>  
<html>
```

```
<body>
  <p id="demo"></p>
  <script>
    document.getElementById("demo").innerHTML = "Hola Mundo";
  </script>
</body>
</html>
```

Es importante destacar que, si bien es posible añadir todo el código JavaScript dentro de esta etiqueta (embebido), es preferible hacerlo a través de un fichero externo con extensión .js. Normalmente, esta etiqueta suele situarse al final del body del documento HTML. ¿Por qué? Porque de este modo, se impiden bloqueos al parsear el HTML. En caso de no hacerlo así, el usuario tendría que esperar a que se descarguen y ejecuten todos los scripts y esto no ofrece una buena experiencia de usuario.

Existen alternativas a lo nombrado anteriormente, que permiten la descarga y ejecución del script de una forma diferente.

**<script async> vs. <script defer>**

## ¿Qué problema soluciona?

Los elementos <script> bloquean el análisis y renderizado del HTML de la página. Cuando el navegador encuentra un recurso de este tipo, detiene el análisis de HTML, descarga el recurso, lo ejecuta y prosigue donde lo dejó. HTML5 nos ofrece **async** y **defer** para evitar bloqueos antes de renderizar la página.

### RENDERIZADO Y CARGA DE SCRIPT

Antes de la aparición de estos atributos, se recomendaba colocar los elementos <script> al final del HTML, para que cuando el analizador se los encontrase, ya hubiese analizado y renderizado todo el documento. Los atributos **async** y **defer** nos ofrecen una solución a este problema, sin forzarnos a recolocar los scripts, con algunas diferencias entre ellos, que son:

- <script> (normal): bloquea el análisis y lo reanuda una vez ejecutado.
- <script async>: el script se descarga de forma asíncrona pero se sigue bloqueando al ejecutarse. No garantiza la ejecución de los scripts asíncronos en el mismo orden en el que aparecen en el documento.
- <script defer>: el script se descarga de forma asíncrona, en paralelo con el análisis HTML, esperando a que termine el análisis HTML para realizar la ejecución. No hay bloqueo en el renderizado HTML. La ejecución de todos los scripts se realiza en orden de aparición.

### ¿CUÁL USO Y CUÁNDO?

- **defer** parece la mejor opción de forma general. Siempre que el script a ejecutar no manipule o interaccione con el DOM antes de que se renderice. También es la mejor opción si el script tiene dependencias con otros scripts e importa el orden de ejecución.
- **async** se recomienda para scripts que manipulan o interaccionan con el DOM antes de su carga y/o no tienen dependencias con otros scripts.
- **El uso normal, sólo si el script es pequeño**, ya que la parada del análisis HTML será insignificante en comparación a la descarga del script.

Soporte	8.0	10.0	3.6	5.1	15.0
Versión					

- **<script async>**: permite descargar el script de forma asíncrona pero bloquea en análisis del HTML al ejecutarse. Además, no garantiza que los scripts se vayan a ejecutar en el mismo orden en el que están enlazados en el documento HTML.
- **<script defer>**: el script se descarga de forma asíncrona, en paralelo con el análisis del HTML. Cuando el análisis del HTML finaliza, los scripts se ejecutan en el orden en el que están.

Debemos tener cuidado con los problemas de compatibilidad entre navegadores. Por ejemplo, hay versiones de Gecko (motor de Mozilla) que difiere en el uso del atributo **defer**, ya que sólo funcionará cuando la etiqueta `script` tenga el atributo `src` y no funciona con código embebido en el `html`. En cualquier caso, podemos comprobar siempre la compatibilidad entre navegadores con herramientas como [Can I Use](#).

A
utentia



## Compatibilidad entre navegadores

### ¿En qué consiste?

Cada navegador implementa los estándares de manera distinta. Esto puede dar lugar a que los usuarios tengan una experiencia diferente en función del navegador que están usando.



**CONCEPTO**

La compatibilidad entre navegadores es un concepto a tener en cuenta a la hora de desarrollar aplicaciones web, debido a que para una misma web, **usar distintos navegadores puede resultar en una experiencia distinta**. Puede darse el caso extremo en el que exista incompatibilidad con un navegador, quedando limitada la audiencia de esa web.

Cuando el diseño se desajusta entre navegadores, puede ocurrir que el texto no quepa en la pantalla, que no sea visible la barra de scroll o que cierto código en JavaScript no se ejecute, etc. Como es inviable comprobar la compatibilidad entre todos los navegadores del mercado, merece la pena asegurarlo entre los que tienen más cuota de mercado, como Chrome, Firefox y Safari.

Hay distintas acciones que nos ayudan a asegurar esta compatibilidad, entre las que se encuentran:

- **Validar tanto el HTML como el CSS** de la web para que cumplan el estándar.
- **Resetear los estilos CSS**. Cada navegador tiene unos valores por defecto para ciertas propiedades, haciendo que algunos elementos se vean distintos.
- **Usar técnicas soportadas**. La web de [Can I Use](#) muestra la compatibilidad de funciones de la API de JavaScript para distintos navegadores.

**DIFERENCIAS ENTRE NAVEGADORES**

Hay dos piezas fundamentales en un navegador: por un lado el motor de renderizado (que analiza el código HTML y CSS) y por otro el motor de JavaScript.

Cada **navegador utiliza un motor distinto** que implementa los estándares con pequeñas diferencias. Además, esta implementación puede cambiar con las versiones y con el sistema operativo.

Es por esto que no todos los navegadores interpretan el HTML, CSS y JavaScript igual. Aunque a día de hoy las diferencias sean pequeñas, pueden hacer que un usuario no pueda ver correctamente la página.

Una vez vistas las diferentes alternativas, se podría concluir que usar el atributo defer y emplazar el elemento script dentro de la etiqueta *head* del HTML es la mejor opción para enlazar ficheros externos, ya que nos permite obtener el mismo comportamiento que al ubicarla al final del *body* pero sin mezclar contenido y scripts. En caso de no querer usar esta propiedad, se podría situar el script al final del documento HTML, obteniendo esta misma experiencia de usuario.

## Noscript

La etiqueta `<noscript>` permite mostrar un contenido distinto en caso de que el usuario deshabilite los scripts en el navegador o sea el propio navegador el que no los soporte.

```
<!DOCTYPE html>
```

```
<html lang="en">
  <head>
    <title>demo autentia</title>
  </head>
  <body>
    <noscript>
      <strong>Lo sentimos, pero `demo autentia` no funciona
      correctamente si JavaScript está deshabilitado. Por favor
      habilítelo para continuar.</strong>
    </noscript>
  </body>
</html>
```

Podemos usar esta etiqueta tanto en la cabecera como en el body del documento HTML. Si la usamos en la cabecera, `<noscript>` solo puede contener las etiquetas `<link>`, `<style>`, y `<meta>`.

# CSS

El CSS (Cascading Style Sheets) describe cómo se tienen que mostrar **visualmente** los elementos de HTML. Se utiliza para definir el estilo de una página web, incluyendo el **diseño, layout y variaciones** en la **interfaz** para distintos dispositivos y tamaños de pantalla. En resumen, el **HTML** aporta la **estructura** y los elementos de una web (esqueleto) y el **CSS** aporta la **capa visual** a los elementos del HTML.

El CSS hace referencia a elementos de HTML mediante varias cosas como puede ser el nombre de la etiqueta, la clase o el id. Aquí tenemos un ejemplo de un div con un fondo rojo:

HTML

```
<div class="class-example">
  <p> Este es el primer párrafo </p>
  <p> Este es el segundo párrafo </p>
</div>
```

CSS

```
.class-example {
  background-color: red;
}
```

## Selectores

Los selectores sirven para definir los elementos sobre los que se van a aplicar reglas CSS. Hay distintos tipos de selectores que además se pueden combinar utilizando unos operadores (llamados combinadores) para hacer selecciones más complejas.

## Selector de tipo

Selecciona todos los elementos del tipo que se ha definido en el selector.

```
div {  
    // El estilo se aplicará a los elementos div.  
}
```

## Selector de clase

Selecciona todos los elementos que tienen la clase del selector.

```
.example {  
    // El estilo se aplicará a todos los elementos que tengan la  
    // clase 'example'.  
}
```

## Selector de id

Selecciona el elemento que tenga el id definido en el selector. Solo se selecciona un elemento porque el id es único en un documento HTML.

```
#example {  
    // El estilo se aplica al elemento con el id 'example'.  
}
```

## Selector universal

Selecciona todos los elementos. Se puede utilizar junto con combinadores para hacer selecciones más complejas como seleccionar todos los hijos de un tipo de elemento.

```
* {  
    // El estilo se aplica a todos los elementos.  
}
```

## Selector de atributo

Selecciona los elementos que tengan el atributo definido en el selector. También se puede poner el atributo con un valor para seleccionar solo los elementos que tengan el atributo y el valor.

```
[attr] {  
    // El estilo se aplica a todos los elementos con ese atributo.  
}  
[attr=value] {  
    // El estilo se aplica a todos los elementos con ese atributo y  
    // valor.  
}
```

Con los selectores anteriores se pueden hacer selecciones más complejas usando **combinadores**. Los combinadores son operadores que se utilizan entre selectores.

## Combinador de hermanos adyacentes

Selecciona hermanos adyacentes, dos elementos que comparten padre y uno se encuentra al lado del otro. El combinador es el símbolo **+**.

```
h1 + p {  
    // El estilo se aplica a todos los p que estén inmediatamente a  
    // continuación de un h1.  
}
```

## Combinador general de hermanos

Selecciona hermanos, sin necesidad de que uno siga a otro. El combinador es el símbolo **~**.

```
h1 ~ p {
```

```
// El estilo se aplica a los p que son hermanos del h1 que les
// precede.
}
```

## Combinador de hijo

Selecciona los hijos directos del primer elemento. El combinador es el símbolo >.

```
div > p {
    // El estilo se aplica a los elementos p que son hijos de un div.
}
```

## Combinador de descendientes

Selecciona todos los elementos descendientes del primer elemento. El combinador es un **espacio**.

```
div li {
    // El estilo se aplica a los li que estén dentro de un div, da
    // igual si son hijos directos o no.
}
```

## Unidades de longitud

En CSS existen **diversas unidades** para expresar una longitud.



## Unidades de medida

**autentia**

### ¿Qué son?

Propiedades en CSS que se emplean para **establecer la disposición de los elementos en el documento**. Definen la altura, anchura y margen de los elementos a través de un valor numérico (entero o decimal), seguido de una unidad de medida.

 **UNIDADES ABSOLUTAS**

Su valor real es directamente el valor indicado, por lo que no dependen de otros componentes para situar los elementos.

- **px** (píxeles).
- **cm** (centímetros).
- **mm** (milímetros).
- **in** (pulgadas): equivalente a 25,4 milímetros.
- **pt** (puntos): equivalente a 0,35 milímetros.
- **pc** (picas): equivalente a 4,23 milímetros.

No se recomienda utilizar unidades absolutas si queremos un diseño **responsive**, ya que al ser unidades fijas, no se adaptan de igual manera a todas las pantallas.

 **UNIDADES RELATIVAS**

Su valor real está relacionado con otro elemento. Son las más utilizadas en el diseño web por la flexibilidad que ofrecen ya que se adaptan a diferentes pantallas si se usan correctamente.

- **em**: unidad relativa a la propiedad `font-size` del elemento padre.  
`html { font-size: 13px }` // `13px * 2 = 26px`  
Si tuviésemos el elemento hijo 'span' dentro de h1, al ser h1 el padre, `1em = 26px`  
`span { font-size: 1.5em } // 26px * 1,5 = 39px`
- **rem**: igual que la anterior, pero esta es relativa con respecto al `font-size` del elemento `root` (`root` es la etiqueta `html`). **En caso de no definirlo, toma el valor por defecto que son 16px**. En el ejemplo anterior, `1rem = 13px`. Esto es muy útil porque si algún usuario decide cambiar el tamaño de letra por defecto del navegador, todos los elementos de nuestro árbol serán flexibles al cambio.
- **ch**: relativa al ancho del cero (0).
- **vw**: relativa al 1% del ancho del `viewport`. El `viewport` es el tamaño de la ventana del navegador.
- **vh**: igual que la anterior, pero relativa al 1% del alto del `viewport`.
- **%**: relativa al elemento padre.

## Unidades absolutas

En CSS **no se recomienda** utilizar unidades de **longitudes absolutas** porque los tamaños de pantalla varían mucho. Las distintas unidades absolutas son: **cm**, **mm**, **in** (pulgada), **px** (píxeles), **pt** y **pc**. La unidad más utilizada de estas son los píxeles.

## Unidades relativas

Las unidades relativas expresan una longitud relativa a otra propiedad de longitud. Este tipo de unidades **escalan mejor** a la hora de renderizar en distintos tipos de pantalla. Las unidades más utilizadas son:

- **em**: esta unidad es relativa a la propiedad `font-size` del elemento. 2em significa dos veces el tamaño de la fuente.

- **rem**: igual que la anterior pero esta es relativa al font-size del elemento root.
- **ch**: relativa al ancho del cero (0).
- **vw**: unidad relativa al 1% del ancho del viewport. El viewport es el tamaño de la ventana del navegador.
- **vh**: igual que la anterior pero relativa al 1% del alto del viewport.
- **%**: unidad relativa al elemento padre.

## Layouts

El layout es básicamente la disposición de los elementos en la pantalla. En este apartado se explicará justamente eso: cómo colocar los elementos HTML donde queremos. Antiguamente, para maquetar una página web se utilizaban tablas pero hace un tiempo, surgieron herramientas como flexbox o grid como una solución mucho más práctica.

### Flexbox

Flexbox es un método de layout **uni-dimensional** utilizado para colocar elementos en **filas** o **columnas**. De esta forma, se consigue que los elementos se **expandan** o se **contraigan** de forma **dinámica** según el ancho o alto de la pantalla. Aunque en este apartado se resumen algunas de las características más importantes de flexbox, es muy recomendable echar un ojo a [esta página](#), ya que explica muy bien la herramienta de principio a fin.



## Flexbox

**autentia**



### ¿Qué es?

Módulo de CSS (layout mode) unidimensional utilizado para **distribuir el espacio de los elementos en filas o columnas de una forma dinámica y sencilla** y que permite desarrollos responsive gracias a elementos flexibles que se adaptan automáticamente al contenido.

 PREVIO A FLEX

Antes de Flex, se usaban distintos modos para la disposición de los elementos (ojo, todavía se usan):

- **En línea** (display:inline).
- **En bloque** (display:block).
- **En tabla** (display:table).
- **Position** (static, relative, absolute, etc.).
- **Float** (right, left, inherit, etc.).

Flex es una mezcla de estas propiedades en cuanto a cómo afecta a la disposición de los elementos contenidos en un contenedor. Un diseño Flexbox consiste en un **flex container** que contiene elementos flexibles (**flex items**).

 PROPIEDADES MÁS USADAS

Para que un contenedor sea flexible, se usa la propiedad **display: flex**.

Al usar flexbox, el eje principal es el horizontal en caso de que **flex-direction** sea una fila, y vertical en caso de que sea una columna. El eje secundario será el perpendicular al principal.

- Alineamiento a lo largo del eje principal: **justify-content**: flex-start | flex-end | center, etc.
- Alineamiento a lo largo del eje secundario: **align-items**: flex-start | flex-end | center, etc.
- Dirección de los elementos (de izquierda a derecha, de arriba a abajo, etc.): **flex-direction**: column | row | row-inverse, etc.
- Por defecto, Flex intenta ajustar los elementos en una fila pero esto se puede modificar: **flex-wrap**: wrap | no-wrap | wrap-inverse, etc.

flex-start	space-between	Original	wrap
flex-end	space-around	nowrap	wrap-reverse
center	space-evenly		

Al utilizar flexbox, habrá ciertas **propiedades** que se podrán aplicar al elemento **padre** y otras que se podrán aplicar al **hijo**. En esta explicación sólo mencionaremos las propiedades aplicables al padre. Veamos un ejemplo:

Lo primero es definir, en el CSS, **cuál** será el componente **padre**. Por ejemplo:

HTML

```
<div class="wrapper">
  <p> Este es el primer párrafo </p>
  <p> Este es el segundo párrafo </p>
</div>
```

---

## CSS

```
.wrapper {  
    display: flex;  
}
```

Esto, lo que hará, será indicar que el **elemento** con la **clase wrapper** pasa a tener un **display** de tipo **flex** y todos los elementos que sean **hijos directos** pasarán a ser **flex items**, poniendo los dos párrafos en una **fila** (esta es la dirección que aplica flexbox por defecto). Podemos cambiar la dirección para que los elementos formen una **columna** utilizando la propiedad **flex-direction** y asignándole el valor **column**.

Si queremos modificar la **alineación** de los elementos en el **eje principal** (por ejemplo, el **eje horizontal** si es una **fila** y el **vertical** si se trata de una **columna**), tendremos que usar la propiedad **justify-content**, con la que podremos situar los objetos al principio, mitad o final del eje, o determinar la separación entre los objetos de forma simétrica con respecto al centro del eje:

- **justify-content: flex-start**



- **justify-content: center**



- `justify-content: flex-end`



- `justify-content: space-evenly`



- `justify-content: space-around`



- `justify-content: space-between`



Si queremos modificar la alineación de los elementos en el **eje secundario** (por ejemplo, el eje vertical si es una fila), deberemos usar la propiedad `align-items`.

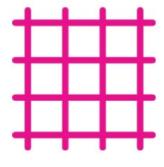
## Grid

CSS Grid Layout es un sistema de layout **bi-dimensional** que permite

disponer el contenido en **filas y columnas** (grid significa cuadrícula). Para comenzar a definir un grid hacemos algo muy parecido que con flexbox: utilizamos la propiedad de CSS display.

```
.wrapper {  
    display: grid;  
}
```

## Grid Layout


**¿Qué es?**
autentia

**CONCEPTOS BÁSICOS**

- Grid Layout se compone de líneas horizontales (para las filas) y verticales (para las columnas).
- El espacio delimitado entre dos líneas consecutivas se le llama **track**.
- Una vez que especificamos el número de filas y columnas, Grid Layout **numera las líneas automáticamente**.
- Una **celda** es el espacio que define la intersección de las líneas verticales y horizontales, teniendo el tamaño 1x1 en nuestra rejilla.
- Un **Grid** es el espacio que ocupa más de una celda en nuestra rejilla.

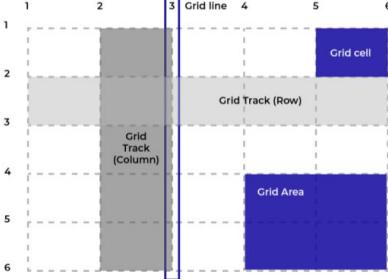


Diagrama que ilustra la estructura de una rejilla de 6 columnas y 6 filas. Se muestra un cuadro gris que abarca las columnas 1 y 2, y las filas 1 y 2. Se etiqueta como "Grid Track (Column)". Una línea vertical azul se extiende de la fila 1 a la fila 2 y se etiqueta como "Grid line". La zona sombreada entre la columna 1 y la columna 2, y entre la fila 1 y la fila 2, se etiqueta como "Grid Track (Row)". La celda central de este cuadro gris se etiqueta como "Grid cell". La celda completa que abarca las columnas 1 y 2 y las filas 1 y 2 se etiqueta como "Grid Area".

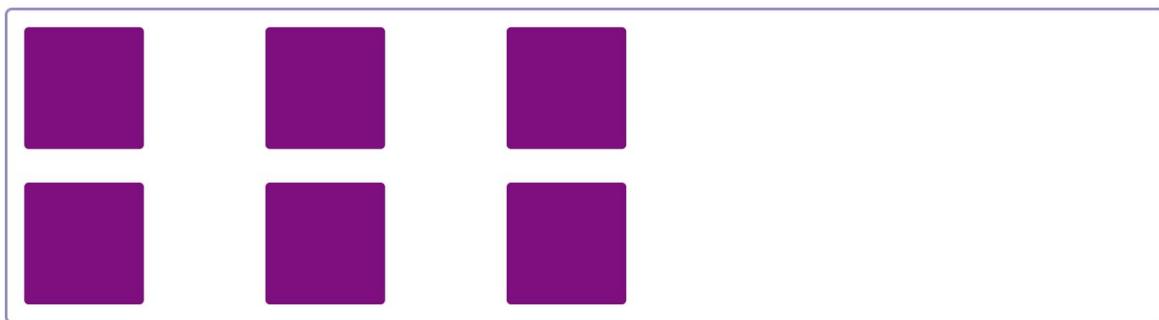
**CARACTERÍSTICAS**

- Es parte de la especificación de CSS, por lo que **no hay problemas de incompatibilidades**.
- Nos permite **colocar los ítems sin tener que hacer trucos como** (margin:auto, position, etc.), ya que flexbox solo tiene una dimensión (columnas o filas).
- Los ítems cuya posición no se especifique **se colocaran solos** (de manera automática), gracias al algoritmo de **auto placement**, ya que Grid es una rejilla, **no es una tabla**.
- Nos ofrece una **sintaxis muy extensa** en su especificación.
- Nos facilita la creación de diseños complejos con layouts.
- **Grid Layout y Flexbox se pueden combinar.** Permitiéndonos contener dentro de un Grid una estructura hecha en Flexbox que sólo crece en una dirección.
- Un contenedor en Flexbox es el conjunto de ítems en una dirección, a diferencia de CSS Grid en el que **cada Grid es un contenedor**.

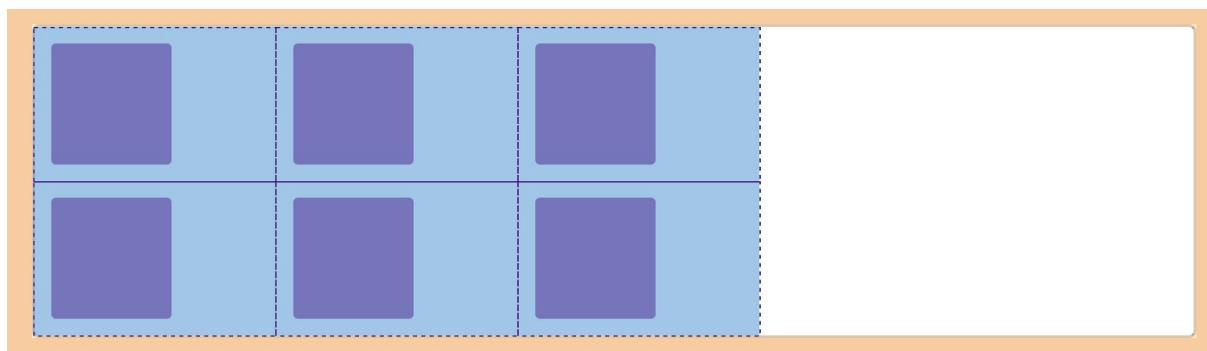
Al contrario que con flexbox, tendremos que seguir algún otro paso más antes de notar la diferencia visual. Esto es porque tras aplicar la propiedad anterior, el elemento de clase wrapper pasará a ser una cuadrícula de **una sola columna** y por tanto, no habrá diferencia visual. Para añadir más columnas, tendremos que añadirle la propiedad grid-template-columns al parent:

```
.wrapper {  
    display: grid;  
    grid-template-columns: 100px 100px 100px;  
}
```

Al introducir esta línea, los hijos directos del elemento con clase wrapper se organizarán en 3 columnas de 100 píxeles cada una.



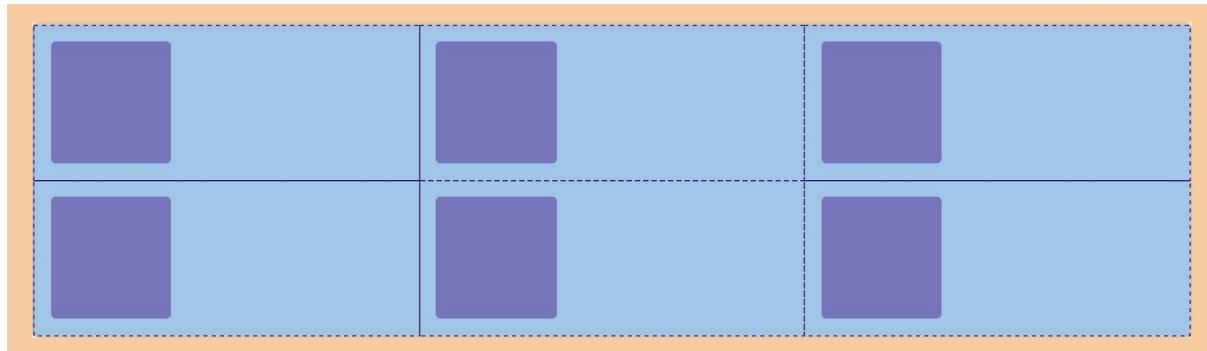
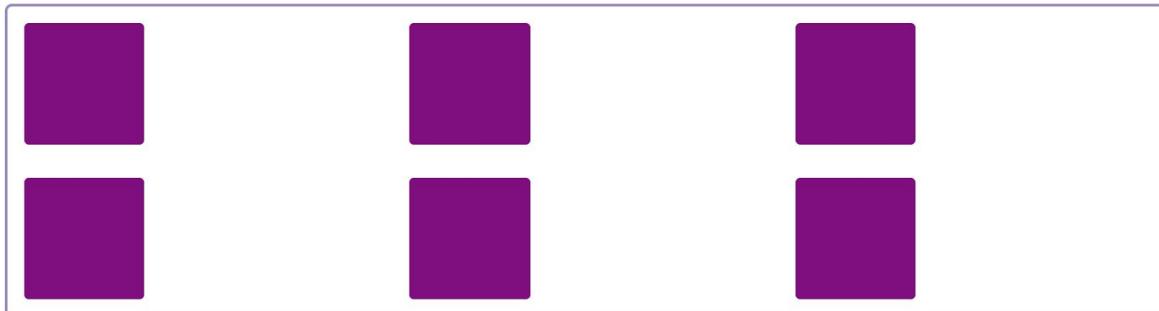
Si analizamos la cuadrícula con el inspector del navegador, se ve de la siguiente forma:



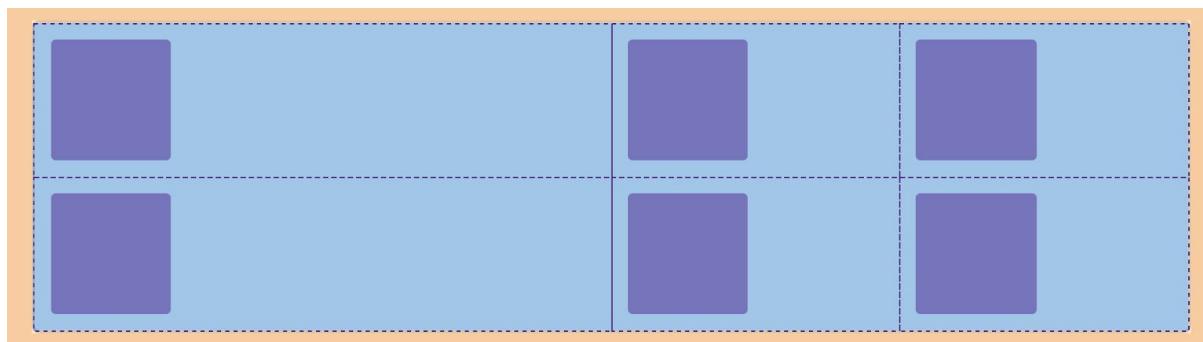
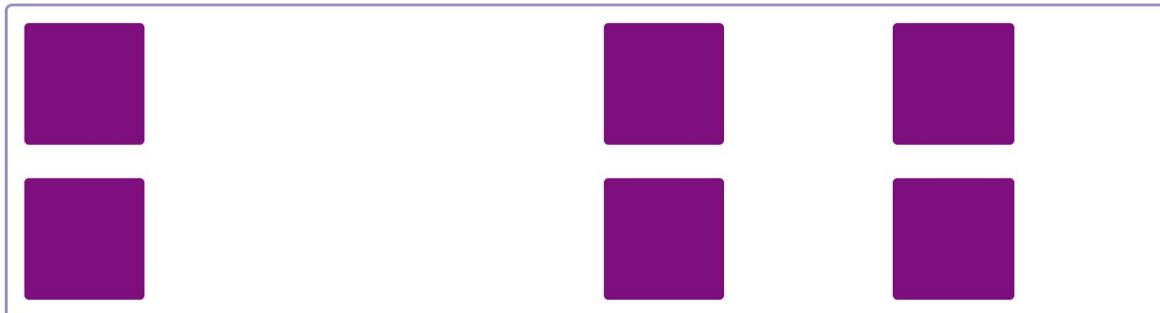
Además de poder crear grids usando longitudes y porcentajes, podemos usar fr para flexibilizar el tamaño de las filas o columnas. La unidad fr representa una **fracción** del **espacio libre** en el contenedor de la cuadrícula. Si tomamos el ejemplo anterior y cambiamos las unidades por estas, quedaría así:

```
.wrapper {
```

```
display: grid;  
grid-template-columns: 1fr 1fr 1fr;  
}
```



Esto, lo que hará, es coger el espacio (en este caso horizontal) restante y repartirlo en tres partes iguales que serán ocupados por cada una de las columnas. Esto funciona de forma **proporcional**. Es decir, si hubiéramos usado “grid-template-columns: 2fr 1fr 1fr;” se hubiera dividido el espacio en **cuatro fracciones, dos** de las cuales hubieran sido ocupadas por la **primera columna** que sería el doble de grande que las otras dos.



También podemos generar columnas, si son del mismo tamaño, usando repeat. De esta forma, estas dos expresiones producirían el mismo efecto:

- `grid-template-columns: 1fr 1fr 1fr`
- `grid-template-columns: repeat(3, 1fr)`

Para cambiar el **espaciado** entre las celdas de la cuadrículas se puede usar gap. Esta propiedad acepta unidades de distancia y porcentajes pero no una unidad fr.

También se puede especificar a qué columna y/o fila pertenece un elemento si a ese elemento hijo se le añade la propiedad grid-column y/o grid-row. Todo esto y otras muchas funcionalidades de grid están explicadas con más en detalle en [esta página](#).

## Custom properties

Las custom properties en CSS son, básicamente, **variables** en las que se almacenan **valores específicos** que se pueden **reutilizar** a lo largo de la página web. La adición de esta funcionalidad en CSS es algo muy importante ya que normalmente, en las páginas web hay mucho CSS y con muchos valores repetidos.

### Declarando una variable

Declarar una variable es tan sencillo como nombrarla con **dos guiones** antes del nombre y asignarle el valor deseado:

```
element {  
  --primary-color: brown;  
}
```

El elemento donde se declare la variable indicará el scope dentro del cual se puede utilizar. Una práctica común es declarar variables en el elemento root, para que estén disponibles a lo largo de toda la aplicación. Por ejemplo:

```
:root {  
  --primary-color: brown;  
  --secondary-color: purple;  
  
  --title-text-size: 22px;  
  --body-text-size: 18px;  
}
```

## Utilizando una variable

Siguiendo el ejemplo del anterior apartado, para utilizar la variable `primary-color`, habría que utilizar la palabra “var” de la siguiente forma:

```
element {  
    background-color: var(--primary-color);  
}
```

## Posicionamiento

La propiedad `position` se utiliza para posicionar un elemento en un documento. Hay distintos tipos de posicionamiento: `static`, `relative`, `absolute`, `fixed` y `sticky`. En este documento explicaremos los más utilizados.

Position



### ¿En qué consiste?

Es una propiedad de CSS llamada **position**, cuya función determina cómo se posicionarán los elementos en la página. Le acompañan además unas propiedades de desplazamiento que precisan con más detalle esta posición (*top*, *right*, *bottom*, *left* y *z-index*). Un elemento **posicionado** es aquel que tenga un *position* definido y cuyo tipo no sea *static*.

**autentia**

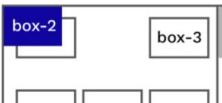
 **TIPOS**

- **STATIC:** valor por defecto. Posiciona los elementos de acuerdo al flujo normal del documento y los elementos a su alrededor. Las propiedades de desplazamiento **no tienen efecto**.

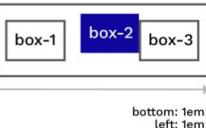
• **FIXED:** posiciona el elemento dentro del *viewport* inicial, fijándose en un sitio independientemente de la posición de los demás elementos.



box-1 box-2 box-3

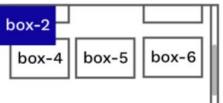


box-2 box-3



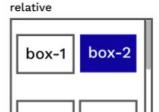
box-1 box-2 box-3

bottom: 1em; left: 1em;



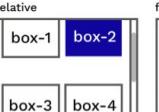
box-2 box-4 box-5 box-6

relative



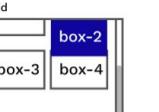
box-1 box-2

relative



box-1 box-2  
box-3 box-4

fixed



box-2 box-3  
box-4

## Static

Este es el valor **por defecto** de la propiedad *position* y posiciona el objeto de acuerdo con el flujo normal del documento (es decir, el elemento se sitúa de forma normal y depende de la posición del resto de elementos de su alrededor). En este caso, las propiedades CSS *top*, *right*, *bottom*, *left* y *z-index* no tienen efecto.

## Relative

Este valor posiciona el objeto de acuerdo con el flujo normal del documento pero modifica su offset con respecto a sí mismo, según los valores de *top*, *right*, *bottom* y *left*. El offset **no afecta a la posición de los elementos que le rodean**. La propiedad *z-index* define la capa o altura en la

que se encuentra un elemento.

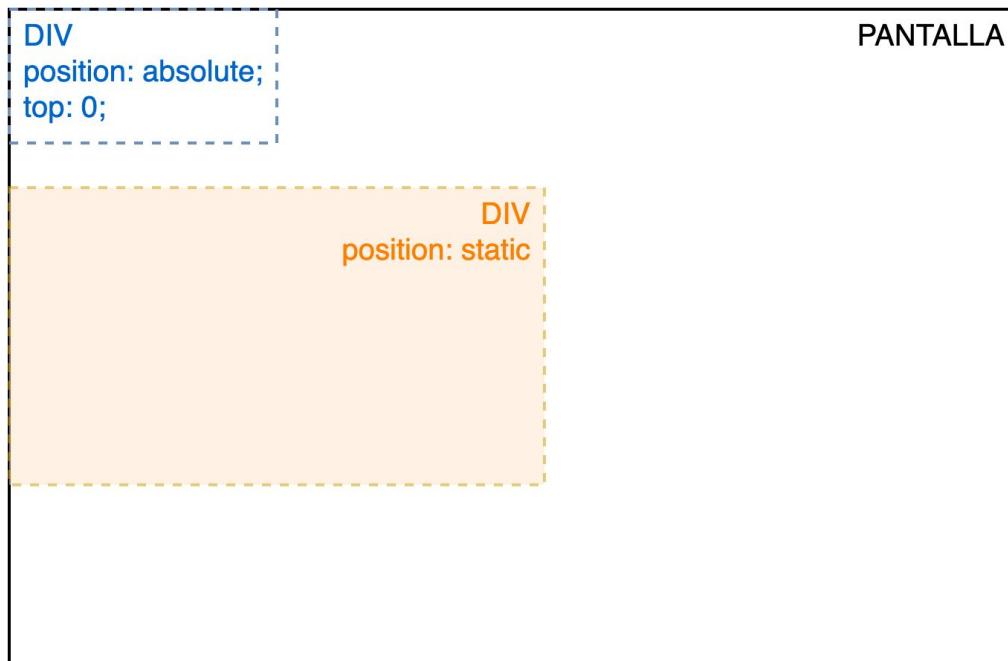
## Absolute

Si establecemos la propiedad position de un elemento a absolute, **se extraerá del flujo normal** del documento y no se reservará ningún espacio para éste en la interfaz. El objeto se posicionará con respecto a su ancestro posicionado más cercano (es decir, al padre más cercano al que hayamos asignado un valor a position, siempre y cuando, el valor no sea static). Si no hay ninguno, se utiliza el cuerpo del documento.

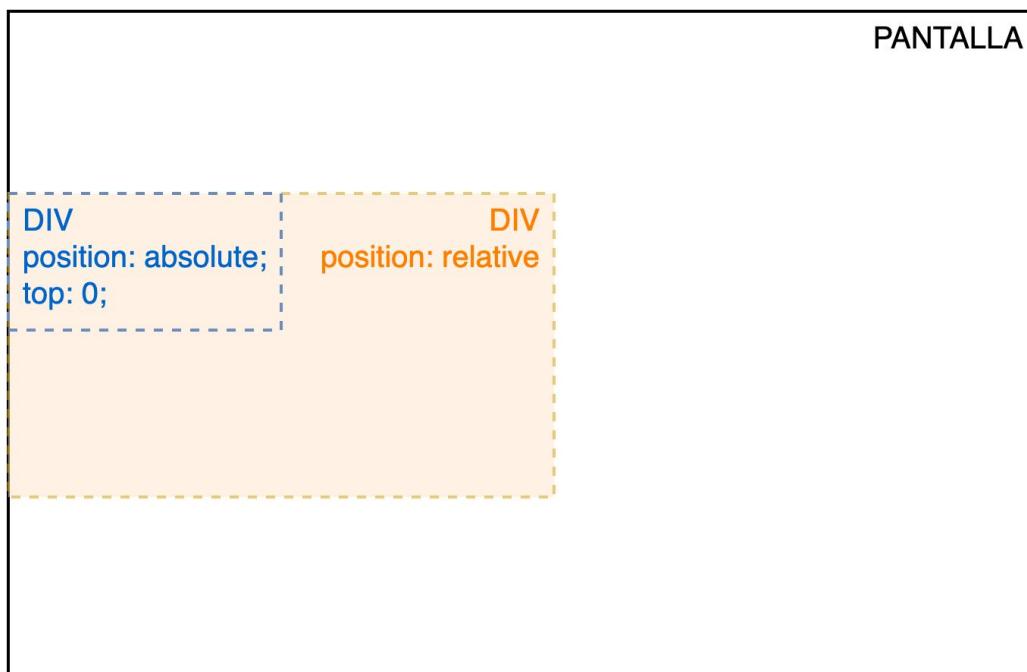
Por ejemplo, pongamos que tenemos un div y hay un elemento dentro al que aplicamos el siguiente CSS:

```
element {  
    position: absolute;  
    top: 0;  
}
```

Lo que ocurrirá, es que este elemento se pegará a la parte superior de la pantalla:



Sin embargo, si al div padre le añadimos la propiedad “position: relative” (o cualquier tipo de position que no sea static), éste se convertirá en un ancestro posicionado y por tanto, el elemento interno que tiene “position: absolute” se quedará contenido dentro de ese div:



## Diseño fluido

El diseño fluido se basa en la **proporcionalidad** a la hora de colocar los elementos a lo largo de la interfaz, por lo que estos ocupan siempre el **mismo porcentaje** del espacio en diferentes tamaños de pantalla. Esto quiere decir que cuando se utilizan unidades de medida en CSS, se utilizan porcentajes.

### Diseño fluido

#### ¿Qué es?

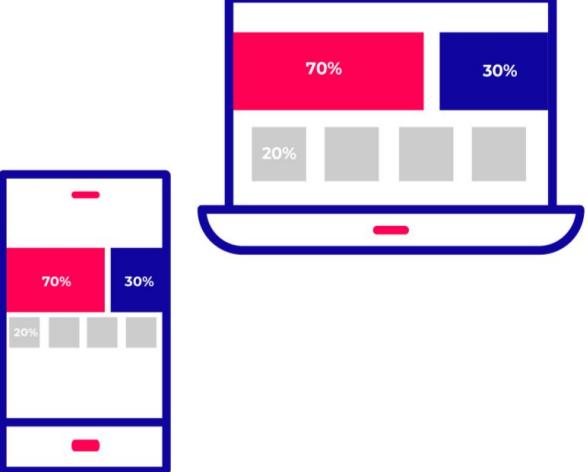
Se basa en la proporcionalidad a la hora de colocar los elementos a lo largo de la interfaz usando **porcentajes** o **em** en vez de píxeles, por lo que independientemente del tamaño de la pantalla, el porcentaje será igual para todos.

 **EN QUÉ CONSISTE?**

Con la finalidad de que en distintas pantallas se visualice la información de manera muy parecida, el diseño fluido hace uso de los porcentajes para que tanto en dispositivos móviles como en pantallas grandes, los elementos se muestren de igual forma y siempre se llene el ancho de la página.

Esto puede acabar con una experiencia de usuario bastante desagradable ya que si tenemos la misma disposición de los elementos en todos los dispositivos, lo que se vea bien en una pantalla grande, se puede ver muy pequeño en un móvil.

Por ejemplo, en un monitor muy grande, las imágenes se podrían ver muy estiradas, mientras que en un móvil, la letra pueda llegar a ser demasiado pequeña.



autentia

## Diseño responsive

A diferencia del diseño fluido, el diseño responsive usa **CSS Media Queries** para presentar **distintos layouts** dependiendo del tamaño o tipo de pantalla.

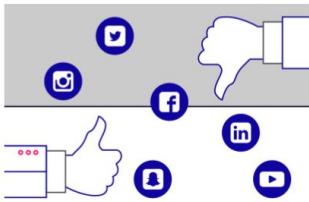
## Responsive Web Design

### ¿Qué es?

Es un **enfoque que se preocupa de desarrollar y diseñar sitios web** que puedan ajustarse a cualquier resolución, adaptando la fuente y las imágenes a cualquier dispositivo. Se intenta **que el usuario tenga una experiencia satisfactoria independientemente del dispositivo** que utilice para acceder.

#### ¿POR QUÉ LO NECESITAS?

- Mejorar la experiencia del usuario.** Según Google, los usuarios de móvil tienen una tasa de rebote del 61% frente al 67% de conversión, si tienen una buena experiencia.

- El acceso a contenidos desde el móvil está en auge.** El uso de Internet desde el móvil es de más del 75% a nivel global y sigue subiendo gracias a la mejora del ancho de banda y los dispositivos.



- Google favorece el posicionamiento** de los sitios con Responsive Design en sus búsquedas, ya que aumenta de forma natural el tráfico orgánico. Esta **actualización en el algoritmo de Google** recibió el apodo de **Mobilegeddon** (Mobile + Armagedón). Así que, si no lo haces por los usuarios, hazlo por tu posicionamiento SEO.



- Aumenta la velocidad de carga.** Ya que son más ligeros y optimizados para móviles que una versión desktop.
- Mejora la difusión por RRSS,** aumentando las ventas y la tasa de conversión.
- Responsive Web Design tiene un enfoque adaptativo, lo que nos **ofrece una ventaja competitiva al estar preparados para cualquier dispositivo.**

## Responsive Web Design

### ¿Cómo hacerlo?

Una aplicación o **sitio web Responsive** debe tener un diseño flexible y capaz de adaptarse a diferentes resoluciones de pantalla y dispositivos. Estas son **algunas técnicas que nos permiten adaptarnos mejor** a cada dispositivo para así ofrecer una experiencia satisfactoria a los usuarios finales.

#### TÉCNICAS BÁSICAS

- Cambiar el tamaño de la caja,** de box-sizing a border-box para evitar que cada elemento añada sus propiedades de tamaño.
- Usar la etiqueta meta** name="viewport" content="width=device-width initial-scale=1", **indica a la página el ancho de la pantalla en pixeles independientemente del dispositivo.** También se pueden establecer atributos como minimum-scale, maximum-scale, user-scalable.
- Hacer uso de CSS Layouts que nos permiten la creación de diseños fáciles y flexibles** como Grid Layout, Flexbox o Multicol.
- Definir puntos de ruptura.** Son expresiones condicionales que aplican diferentes estilos dependiendo del dispositivo. Esto se hace utilizando una herramienta llamada **Media Queries**.
- Usar imágenes vectoriales SVG.** La imagen no pierde calidad al redimensionarse.
- Envolver objetos en un contenedor.** Esto hace un diseño comprensible, limpio y ordenado.
- En el ciclo de desarrollo hay que tener presente que se va a acceder al sitio o la aplicación desde **diferentes dispositivos con distintos tipos de pantalla y resoluciones.**

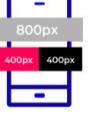


**Unidades relativas**





**Unidades Estáticas**



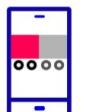
**Con Breakpoints**



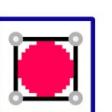
**Vectores**



**Sin Breakpoints**



**Imágenes**



Por ejemplo, pongamos que tenemos el siguiente CSS:

```
@media only screen and (max-width: 600px) {  
    body {  
        font-size: 40px;  
    }  
}
```

Lo que pasará, es que cuando el ancho de la pantalla sea **menor** de 600px, el **fondo de pantalla** pasará a ser de color azul claro. Este es un ejemplo muy simple pero esta herramienta es muy útil y se puede usar para, por ejemplo, reorganizar la interfaz en función del tamaño de pantalla.

# Accesibilidad

La accesibilidad hace que los sitios web puedan ser utilizados por el mayor número de personas posible.

El principal objetivo de la accesibilidad web es el de permitir a cualquier usuario, independientemente del tipo de discapacidad que tenga, el acceso a los contenidos del sitio y permitirle la navegación necesaria para realizar las acciones deseadas. Si nuestra página no es accesible sucederán dos cosas: **perderemos muchos usuarios** que no entenderán nuestra página y por otro lado, los que no perdamos **sufrirán** para poderse manejar con el contenido de la página. Además, aunque no lo parezca, una **gran cantidad de usuarios** tienen algún tipo de discapacidad.



## Accesibilidad Web

### Definición

La accesibilidad web significa que **personas con algún tipo de discapacidad podrán hacer uso de la Web** gracias a un diseño que va a permitir que estas personas puedan percibir, entender, navegar e interactuar con ella.

#### TIPOS DE ACCESIBILIDAD

Los tipos de accesibilidad se clasifican en función de la discapacidad que tenga el usuario:

- **Sensorial:** permite el uso de la web a personas con problemas de sordera, ceguera completa/parcial o para distinguir colores como el daltonismo.
- **Motriz:** mejora el uso para gente que no puede utilizar correctamente un ratón, tienen el control motor delicado o tiempo de respuesta lento.
- **Cognitiva:** reúne una serie de técnicas para permitir que usuarios con un lenguaje de compresión y entendimiento limitado utilicen la web.
- **Tecnológica:** permite el uso de la web a usuarios que no disponen de los recursos suficientes para acceder a la web de manera eficiente, como por ejemplo, conexión lenta a la web o acceso a través de móvil y tablets.



#### TÉCNICAS

Existen una serie de técnicas para conseguir que nuestras web sean accesibles y pautas que podemos seguir:

- **Fundamentales**
  - Elementos sonoros o gráficos con información textual alternativa.
  - Diseño de la web independiente del dispositivo.
  - Desactivar elementos visuales o sonoros para no interferir en la lectura.
  - Emplear un lenguaje sencillo.
  - Buena usabilidad de la Web.
  - Hardware y software actualizados.
- **CSS**
  - Diseño de páginas flexibles al tamaño de la interfaz, tamaño de fuente...
  - Color adecuado y alto contraste.
  - Tamaño de fuente grande y/o flexible.
  - Elementos de interacción fáciles de clicar.
- **HTML**
  - Descripciones detalladas para imágenes complejas.
  - Información alternativa para los marcos.
  - Tablas bien formadas (para su lectura secuencial).

## Soluciones tecnológicas (AT)

Las discapacidades que puede tener un usuario que acceda a nuestra web son **variadas**. Entre ellas podemos encontrar discapacidades visuales, auditivas, motrices, cognitivas... Estos usuarios utilizan las llamadas **tecnologías de apoyo** o assistive technologies (**AT**). Por lo general, una regla que se debería cumplir para funcionar correctamente junto con las AT es que los **controles** de la página sean **accesibles por el teclado** u otro tipo de dispositivo de asistencia, sin necesidad de usar el ratón.

¿Qué tipos de discapacidades hay y qué **herramientas** se utilizan según el tipo de discapacidad?

## Visuales

Las personas con discapacidades visuales utilizan un **lector de pantalla**. Lo que éste hace es leer el contenido de la pantalla a una **gran velocidad** y permitir al usuario navegar e interactuar con la página web.

Sin embargo, esto **depende** también de **cuán accesible sea la página web**, ya que si nuestra página no es accesible, el lector de pantalla tendrá más dificultades para leer el contenido y los elementos de la pantalla según el usuario vaya navegando y por tanto, le costará mucho más usarla.

## Auditivas

Realmente no hay AT específicos para las personas con este tipo de discapacidad que estén orientados al uso del ordenador/web. Aún así, hay **técnicas específicas** para ofrecer alternativas textuales a contenidos de audio que van desde **simples transcripciones**, hasta **subtítulos** que se pueden mostrar junto con el vídeo.

## Motrices

Las discapacidades motrices pueden implicar **problemas puramente físicos** (como la pérdida de una extremidad o la parálisis) o **trastornos neurológicos/genéticos** que conllevan la debilidad o pérdida de control en las extremidades.

Algunas personas, simplemente pueden tener dificultades a la hora de mover el ratón, mientras que otras podrían verse más gravemente afectadas, tal vez estén paralizadas y necesiten utilizar un puntero de cabeza para interactuar con el ordenador. Esto quiere decir que es probable que estos usuarios tengan **limitaciones en cuanto al hardware** (algunos usuarios podrían no tener un ratón).

## Cognitivas

Este tipo de discapacidad engloba una **amplia gama** de discapacidades, desde las personas que presentan capacidades intelectuales más limitadas, hasta toda la población que tiene problemas a la hora de recordar derivados de la edad, u otros.

Aunque dentro de este conjunto hay una amplia gama de discapacidades, todas ellas tienen un **conjunto común de problemas funcionales** en cuanto a páginas web se refiere que incluye dificultades a la hora de **entender** los contenidos, **recordar** cómo completar las tareas y **confusión** ante páginas web diseñadas de forma incoherente.

Una **buena base** de accesibilidad para personas con **deficiencias cognitivas** incluye:

- Proporcionar el contenido en más de un formato, como puede ser texto-a-voz o vídeo.
- Proporcionar contenidos fáciles de entender, como texto escrito con estándares de lenguaje sencillo.
- Centrar la atención en el contenido importante.
- Minimizar las distracciones, tales como contenidos innecesarios o anuncios.
- Proporcionar un diseño coherente de la página web y del sistema de navegación.
- Usar elementos ya conocidos, como los enlaces subrayados en azul cuando aún no se han visitado y en morado cuando sí.
- Dividir los procesos en pasos lógicos y esenciales con indicadores de progreso.
- Ofrecer un sistema de autenticación del sitio web de la forma más fácil posible sin comprometer la seguridad.

- Diseñar formularios fáciles de completar, con mensajes de error claros y de fácil solución.

## Normativas y estándares

En Europa y EE.UU. hay estándares que definen los requerimientos de accesibilidad para los productos/servicios de tecnologías de la información y comunicación. Estos son el estándar **EN 301 549** para Europa y la **Section 508** para EE.UU. Ambas se alinean con las pautas o guías que propone el WCAG (2.0 en el caso de EEUU y 2.1 en el caso de Europa).

### WCAG 2.1

Aunque tiene unas pautas muy extensas, se pueden resumir en ciertos objetivos generales. Tu página web ha de ser:

- **Perceptible:** la información y los componentes de la interfaz de usuario deben ser mostrados a los usuarios en formas que ellos puedan entender.
- **Operable:** los componentes de la interfaz de usuario y la navegación deben ser manejables, aunque sólo sea con el teclado u otros tipos de métodos de entrada.
- **Comprensible:** la información y las operaciones de usuarios deben ser comprensibles.
- **Robusta:** el contenido debe ser suficientemente robusto para que pueda ser bien interpretado por una gran variedad de agentes de usuario, incluyendo tecnologías de asistencia.

## Técnicas básicas de HTML y CSS

Hay **numerosas técnicas** para facilitar la accesibilidad, tanto para HTML como para CSS. Alguna de estas puede ser:

- Esconder mediante CSS texto que indique el objetivo de, por ejemplo, un link o similar de forma que se pueda disponer de esa información.
- Posicionar el contenido con CSS basándose en la estructura del HTML utilizado. Esto quiere decir que si el CSS no está disponible o no es legible por el dispositivo AT, el usuario debe poder seguir determinando el significado del contenido. Por ejemplo, si se utiliza el CSS para colocar cierto contenido en ciertas partes de la pantalla pero el HTML usado no concuerda estructuralmente con esta disposición, si el CSS no estuviera disponible sería extremadamente difícil comprender la página web y la organización o significado de su estructura y contenido, porque este estaría “desperdigado”. Por esto también es muy importante dar preferencia al HTML semántico frente al no semántico.
- Utilizar la propiedad alt en las imágenes.
- Usar notación de tablas en HTML para representar datos tabularmente (por ejemplo, no utilizar sólo divs y mostrar los datos como una tabla mediante CSS).

Hay **muchas otras técnicas** aparte de estas, y se pueden encontrar en estas páginas:

- [Técnicas de CSS](#)
- [Técnicas de HTML](#)

# Bibliografía

Estas son las fuentes que hemos consultado, o en las que nos hemos basado, para la redacción de este material:

- Documentación para desarrolladores de Mozilla.org:  
<https://developer.mozilla.org/en-US/docs/Web/HTML>
- Documentación para la comunidad de Digital Ocean:  
[https://www.digitalocean.com/community/tutorials?primary\\_filter=tutorial\\_series](https://www.digitalocean.com/community/tutorials?primary_filter=tutorial_series)
- Consorcio de la World Wide Web (W3C):  
<https://www.w3.org>
- Especificación del estándar HTML del Grupo de trabajo de tecnología de aplicación de hipertexto web (WHATWG por su nombre en inglés):  
<https://html.spec.whatwg.org>
- Pautas de accesibilidad al contenido web de la W3C:  
<https://www.w3.org/WAI/standards-guidelines/wcag/>
- Datos de la OMS sobre discapacidad visual y auditiva:  
<https://www.who.int/features/factfiles/blindness/es/>  
<https://www.who.int/features/factfiles/deafness/es/>

- Datos sobre daltonismo:

<https://www.color-blindness.com/>

- Distribución por edades de la población española:

<https://www.ine.es/jaxi/Tabla.htm?path=/t20/e245/p08/l0/&file=02002.px>

# Lecciones aprendidas con esta guía

---

Debido al auge que los frameworks web han experimentado a lo largo de los últimos años, los perfiles con estos conocimientos son cada vez más demandados. Pero para llegar hasta este conocimiento, es importante tener primero una buena base.

Este documento intenta ofrecer una guía para poder tener unos conocimientos generales sobre esta base, intentando poner el foco en la importancia de los siguientes conceptos:

y saber los distintos métodos que hay para **adaptarse a las necesidades de los dispositivos** de nuestros usuarios.

- **Ser inclusivos con el porcentaje de usuarios que accedan a nuestra web que no tengan las mismas facilidades** que el resto, construyendo nuestra web o aplicación de forma que les resulte más fácil disfrutar del contenido que queremos ofrecerles.

- **No sólo saber utilizar el HTML, sino saber usarlo de forma coherente** y semántica, entendiendo el porqué de usar ciertas etiquetas.
- Conocer los principios básicos a la hora de maquetar con CSS

En Autentia proporcionamos soporte al desarrollo de software y ayudamos a la transformación digital de grandes organizaciones siendo referentes en eficacia y buenas prácticas. Te invito a que te informes sobre los servicios profesionales de [Autentia](#) y el soporte que podemos proporcionar para la transformación digital de tu empresa.

¡Conoce más!

# Expertos en creación de software de calidad

Diseñamos productos digitales y experiencias a medida



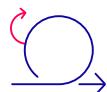
## SOPORTE A DESARROLLO

Construimos entornos sólidos para los proyectos, trabajando a diario con los equipos de desarrollo.



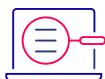
## DISEÑO DE PRODUCTO Y UX

Convertimos tus ideas en productos digitales de valor para los usuarios finales.



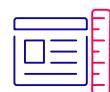
## ACOMPAÑAMIENTO AGILE

Ayudamos a escalar modelos ágiles en las organizaciones.



## AUDITORÍA DE DESARROLLO

Analizamos la calidad técnica de tu producto y te ayudamos a recuperar la productividad perdida.



## SOFTWARE A MEDIDA

Desarrollamos aplicaciones web y móviles. Fullstack developers, expertos en backend.



## FORMACIÓN

Formamos empresas, con clases impartidas por desarrolladores profesionales en activo.

[www.autentia.com](http://www.autentia.com)

[info@autentia.com](mailto:info@autentia.com) | T. 91 675 33 06

---

¡Síguenos en nuestros canales!

