

SISTEMAS WEB I

TEMA 1 – INTRODUCCIÓN A LAS APLICACIONES DE INTERNET

¿QUÉ ES UNA APLICACIÓN PARA INTERNET?

Las aplicaciones web reciben este nombre porque se ejecutan en Internet. Es decir, los datos o los archivos en los que se trabaja son procesados y almacenados dentro de la web. Estas aplicaciones, por lo general, no necesitan ser instaladas en el ordenador.

Las aplicaciones web son un tipo de software que se codifica en un lenguaje soportado por los navegadores web y cuya ejecución es llevada a cabo por el navegador en Internet o una intranet.

Cliente – Servidor:

Modelo compuesto por:

- Servicios o funciones que requieren acceso a información difícil de distribuir, equipamiento especial o capacidad de cómputo
- Servidores que ejecutan los servicios bajo petición
- Clientes, que piden la ejecución de un servicio, con capacidad de cómputo reducida y presentan la respuesta
- Protocolo de petición – respuesta
- Red que soporta las interacciones.

Sistema Web: infraestructura o sistema que permite funcionar a una aplicación web.

Aplicación Web: aplicación distribuida que realiza una función (de negocio) basada en las tecnologías de la web. Consta de recursos específicos de la web. Características:

- Evolución rápida.
- A gran escala y fuertemente distribuidas.
- Multiusuario, lenguaje cultural.
- Seguridad y confidencialidad.
- Diferentes medios de acceso y agentes de usuarios.
- Gran volumen de información, varios formatos y procesos.

VENTAJAS DE LAS APLICACIONES WEB

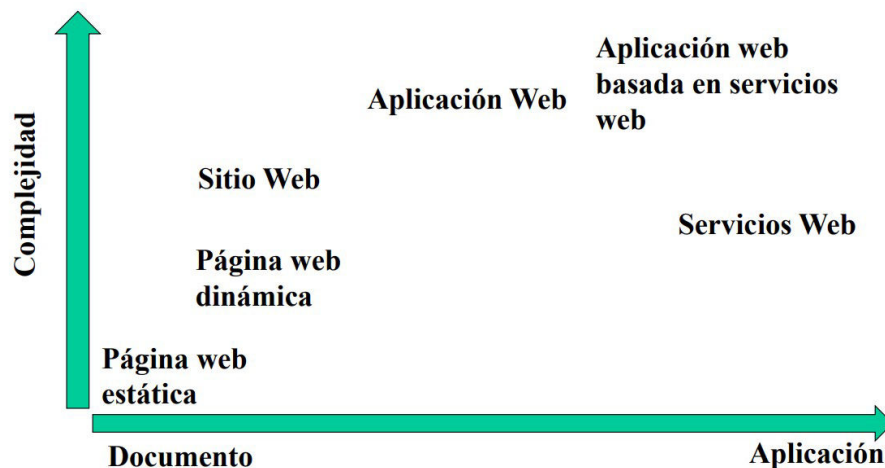
1. **Compatibilidad Multiplataforma:** una aplicación web se puede ejecutar sin problemas en múltiples Sistemas Operativos
2. **Acceso desde Internet:** a la aplicación web se accede a través de internet, por lo que son especialmente interesantes para desarrollar aplicaciones multiusuario y así compartir información
3. **Aplicaciones Ligeras:** las aplicaciones se ejecutan desde el navegador, por lo que el usuario no necesita tener un ordenador de grandes prestaciones para trabajar, al no tener que instalar nada
4. **Fáciles de Actualizar:** las aplicaciones web siempre se mantienen actualizadas y el usuario no tiene que descargar e instalar actualizaciones. Siempre se trabaja con la última versión
5. **Acceso Inmediato:** las aplicaciones web no necesitan ser descargadas, instaladas y configuradas. Además, se puede acceder desde cualquier ordenador conectándose a Internet.
6. **Compatibilidad de Versiones:** no hay problemas de incompatibilidad entre las versiones que utilizan los usuarios, porque todos trabajan con la misma.

Otras Ventajas: ahorran tiempo de instalación y despliegue, no hay problemas de compatibilidad, no hay necesidad de instalarlas y no ocupan espacio, actualizaciones inmediatas, consumo de recursos bajo, multiplataforma portables, alta disponibilidad, difícil de atacar mediante virus, ...

DESVENTAJAS DE LAS APLICACIONES WEB

- Menor funcionalidad que las aplicaciones de escritorio
- Limitaciones del navegador
- Disponibilidad supeditada al proveedor de red
- Alta dependencia del servidor
- Limitación por el protocolo HTTP utilizado
- Cuello de botella en el ancho de banda para grandes cantidades de datos,
- Necesidad de encriptar los datos
- Necesidad de software adicional
- Versiones específicas de navegadores.

MAPA DE TÉRMINOS



SITIOS WEB vs APLICACIÓN WEB

Sitio Web (Website):

- Conjunto de páginas web en un mismo dominio
- Páginas informativas del negocio
- Requieren poca o nula interacción por parte del usuarios.
- Se pueden crear con un CMS (Content Management System)

Aplicación Web (web application):

- Suele ser el propio negocio
- Requieren interacción por parte del usuario y normalmente registrarse.

APLICACIÓN WEB vs SERVICIOS WEB

Aplicación Web:

- Diseñadas para humanos
- Tienen interfaces
- Human To Machine
- Tienen usuarios
- Información y presentación (HTML)

Servicios web

- Diseñados para máquinas
- Exponen APIs
- Machine To Machine
- Tienen Aplicaciones

- Información sin presentación (XML, JSON)

ARQUITECTURA

Se compone de tres capas:

- Capa de acceso (capa del navegador)
- Capa del servidor
- Capa de persistencia (almacenamiento)

Arquitectura Física

Modelo básico:

- Un ordenador dedicado ejecutando un servidor
- Conexión a una red para recibir peticiones y devolver respuestas
- Si la red es interna a una organización – Intranet

Modelo con separación de funciones:

- Un ordenador dedicado ejecutando un servidor
- Conexión a una red para recibir peticiones y devolver respuestas
- Conectividad a otros servidores (de BBDD, por ejemplo).

Modelo con separación de redes:

- Un ordenador dedicado ejecutando un servidor
- Conexión a una red para recibir peticiones y devolver respuestas
- Conectividad a otros servidores en una red interna a través de un cortafuegos.

Modelo de separación completa de funciones:

- Ordenadores para servir contenidos estáticos (páginas HTML).
- Ordenadores diferentes para generar contenidos dinámicos.
- Ordenadores diferentes para bases de datos.
- Ordenadores diferentes para gestión, seguridad, etc.

Modelo de alto rendimiento:

- Varios ordenadores dedicados ejecutando el mismo servidor
- Conexión a una red a través de un balanceador de carga para recibir peticiones y devolver respuestas
- Conectividad a otros servidores (de bases de datos, por ejemplo)

Modelo para una alta disponibilidad:

- El repartidor de carga envía la petición a dos o más servidores
- Todos los elementos están duplicados

FRONT END Y BACK END

Front End

1. Web Performance

- Reducir los tiempos de carga
- Conseguir que el sitio sea utilizable lo antes posible
- Suavidad e interactividad
- Medidas de funcionamiento.

2. Compatibilidad entre Navegadores

Se debe comprobar en diferentes navegadores la funcionalidad, el rendimiento y la accesibilidad.

3. End-to-End Testing

Se debe asegurar que la aplicación funciona como debe, para lo que se replican

escenarios reales (validación de integración y de integridad de datos).

4. Build Automations

Empaquetado, minificación de archivos (minify) y reducción del tamaño de imágenes.

5. Accesibilidad

- Lograr que las páginas sean utilizables para el mayor número de personas independientemente de sus habilidades.
- Web universal y accesible.
- Más relacionado con la parte técnica (texto alternativo en las imágenes)

6. Usabilidad

Mejorar la experiencia, evitar ambigüedades, colocación de los elementos y calidad de la experiencia.

7. Herramientas de edición de Imágenes

GIMP, Photoshop, ...

8. Interfaz de Usuario

UX – UI

9. SEO (Search Engine Optimization)

Posicionamiento de la página web en los buscadores para que aparezca lo antes posible.

Tecnologías

- HTML
- CSS
- JavaScript
- Librerías: SaaS (Syntactically Awesome Style Sheets) o jQuery
- Frameworks: Bootstrap, React, Angular, vue.js

Back – End

1. Automated Testing

Comprobación del servidor, de las APIs y de la Base de Datos (propiedades ACID, operaciones CRUD, esquema, índices, duplicado de información, seguridad, rendimiento).

2. Application Data Access – Acceso a la Información

Roles asignados y Seguridad

3. Lógica de Negocio

También llamado “domain logic”. Reglas: cómo se puede crear, almacenar y cambiar la información.

4. Gestión de Bases de datos

- Funcionamiento eficiente de la DB.
- Gestión de recursos (Memoria, Disco, Red)
- Modificación de la estructura
- Perfiles de usuarios
- Backup

5. Escalabilidad

Diseño pensando en la escalabilidad, control de consumo y rendimiento.

6. Alta Disponibilidad

Despliegue en diferentes regiones con bajo tiempo de recuperación. Está relacionado con la escalabilidad.

Downtime:

- 99% → algo más de 3 días
- 99.9% → casi 9 horas
- 99.99% → 53 minutos
- 99.999% → 5.2 minutos

7. Seguridad

8. Arquitectura del Software

Se debe diseñar el sistema, la autenticación, autorización, la base de datos, logs y peticiones.

9. Transformación de Datos

- Conversión entre formatos (desestructurados, semiestructurados – JSON, XML, ..., y estructurados (DB).
- Validación.

10. Backups

Se debe asegurar la periodicidad de los backups y guardarlos en lugares diferentes, teniendo en cuenta la cantidad de los datos.

Tecnologías

- Scripting: PHP, Python, Rubi, Node.js
- Compilados: C#, Java, Go
- Bases de Datos

TEMA 2 – HTTP

INTRODUCCIÓN

HTTP (HyperText Transfer Protocol) es un protocolo de capa de aplicación para transmitir documentos hipermedia como HTML.

Se da una comunicación entre clientes y servidores (aunque se puede usar con otros propósitos). Es un protocolo de transacciones (funciona en modo petición – respuesta) sin estado.

Se suele utilizar con TCP/IP, aunque se puede usar con cualquier capa de transporte fiable.

El funcionamiento es el siguiente:

- El cliente abre una conexión (ej. TCP).
- El cliente inicia una petición HTTP (request)
- El servidor devuelve la respuesta HTTP (response)
- Se cierra la conexión

Evolución

- **HTTP 1.0 vs HTTP 1.1:** En HTTP 1.1 se pueden realizar diferentes peticiones antes de cerrar la conexión.
- **HTTP 1.1 vs HTTP 2:** En HTTP 2 se puede solicitar varios elementos en paralelo, mientras que en 1.1 se debe esperar a recibir la respuesta antes de hacer otra petición.

A día de hoy la versión más utilizada es HTTP 2, la versión HTTP 3 comienza a ser utilizada.

HTTP/3

HTTP/3 es una versión draft, antes conocido como HTTP over QUIC.

QUIC es un protocolo de capa de transporte inicialmente conocido como Quick UDP Internal Conections, que usa UDP en lugar de TCP. A veces se conoce como TCP/2.

HTTP/3 está soportado por defecto en la mayoría de los navegadores. En algunos no está activado por defecto pero se puede activar.

2. CONCEPTOS

User-agent

Agente de usuario. Son programas que representan a una persona, el cliente (por ejemplo, el navegador).


<http://webaim.org/blog/user-agent-string-history/>


NCSA_Mosaic/2.0
(Windows 3.1)


NETSCAPE
Mozilla/1.0 (Win3.1)


Mozilla/1.22 (compatible;
MSIE 2.0; Windows 95)


Mozilla/5.0 (Windows; U;
Windows NT 5.1; sv-SE;
rv:1.7.5)
Gecko/20041108
Firefox/1.0


Mozilla/5.0 (compatible;
Konqueror/3.2;
FreeBSD) (KHTML, like
Gecko)


Mozilla/5.0 (Macintosh; U;
PPC Mac OS X; de-de)
AppleWebKit/85.7
(KHTML, like Gecko)
Safari/85.5

Stateless

HTTP es un protocolo sin estado, es decir, no se guarda información sobre conexiones anteriores. La respuesta del servidor es la misma para un cliente previo que para uno nuevo.

Ventajas:

- Escalabilidad.
- Menos complejidad.
- Mayor rendimiento.
- Se pueden cachear los recursos.

Desventajas:

- Complica la interacción con el usuario.
- Hace falta información extra para mantener la sesión.
- Susceptible a ataques como DDoS (Distributed Denial of Service).

URL

Una **URL** (Uniform Resource Locator) es un **localizador de recursos** uniforme. **Un recurso** es un elemento que solicita el cliente (ejemplo: documento HTML, imagen vídeo, ...). La URL referencia a un recurso web especificando la localización en una red y un mecanismo para obtenerlo.

Las URL son únicas, cada una identifica unívocamente un recurso.

Formato

scheme://host[:port]/path[?query][#fragment]

scheme: protocolo (forma de obtener el recurso → http, https, ftp, ...).

host: nombre o la IP del servidor al cual nos queremos conectar.

Port: puerto al cual nos queremos conectar (opcional). Por defecto:

- http: 80
- https: 443
- ftp: 21

Path: ruta en el sistema de archivos de la máquina remota donde se encuentra el recurso.

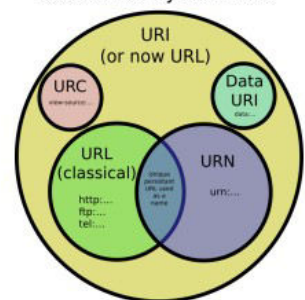
La ruta es relativa al directorio raíz de la web.

Query: parámetros extra, normalmente con el formato "q=text".

Relacionados con este concepto encontramos:

- **URI:** Uniform Resource Identifier, engloba los demás conceptos.
- **URL:** Uniform Resource Locator
- **URN:** Uniform Resource Name
- **URC:** Uniform Resource Characteristic (**metadatos**)

Venn diagram of URIs as defined by the W3C



3. PETICIÓN HTTP

El **formato** de una petición es el siguiente:

Método SP URL SP Versión Http CRLF

(nombre-cabecera:valor-cabecera(,valor-cabecera)*CRLF)*

CRLF

Cuerpo del Mensaje

Donde:

- **SP:** espacio en blanco
- **CRLF:** retorno de carro
- **():** opcional
- *****: se puede repetir
- **Método** que se emplea, habitualmente GET o POST.
- **URL del recurso** que se está pidiendo

- **Versión** del protocolo HTTP que se está empleando.
- Una o más **cabeceras**
 - nombre:valor(,valor)*
 - Los espacios en blanco forman parte del valor, no son separadores.
 - Cada cabecera va en una línea distinta.
- El **mensaje** puede ser vacío (es el caso del método GET) o no (es el caso del método POST).

Ejemplos:

```
GET /en/html/dummy?name=MyName&married=not+single&male=yes HTTP/1.1
Host: www.explainth.at
User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko/20070312
Firefox/1.5.0.11
Accept: text/xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.explainth.at/en/misc/httpreq.shtml
```

POST equivalente al GET anterior, donde el contenido del mensaje es sólo la última línea:

```
POST /en/html/dummy HTTP/1.1
Host: www.explainth.at
User-Agent: Mozilla/5.0 (Windows;en-GB; rv:1.8.0.11) Gecko/20070312 Firefox/1.5.0.11
Accept:text/xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5 Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.explainth.at/en/misc/httpreq.shtml
Content-Type: application/x-www-form-urlencoded
Content-Length: 39

name=MyName&married=not+single&male=yes
```

En el formulario había tres campos, uno con nombre “name”, otro con nombre “married” y el último con nombre “male”. En la respuesta, cada uno de los tres campos está separado por el símbolo “&”. Cada campo va seguido del valor ue el usuario ha introducido en el formulario ara dicho campo, y separando el nombre del campo y su valor va el signo “=”.

Métodos

También llamados verbos, **representan la acción** que se desea efectuar sobre el recurso indicado en la petición.

- Ese recurso podría ser un archivo que reside en un servidor, o podría ser un programa que se está ejecutando en dicho servidor.

Métodos:

- **GET**: permite **obtener un recurso**. **No modifica nada en el servidor**. Es un método **obligatorio**.
- **HEAD**: **Obtiene la cabecera de una petición GET** sin el contenido, **para obtener la meta-información del recurso** (conocer tamaño/versión, ...). Es un método **obligatorio**.
- **POST**: **envía datos al servidor** para que sean procesados por el recurso especificado en la petición. **Los datos se incluyen en el cuerpo de la petición**. Podría crear un nuevo recurso en el servidor, o actualizar un recurso ya existente.
- **PUT**: **envía un recurso determinado** (un archivo) **al servidor**. **A diferencia de POST, crea una nueva conexión** (socket) y la emplea **para enviar el recurso**, lo que resulta más eficiente que enviar dentro del cuerpo del mensaje.

- **DELETE**: elimina el recurso especificado.
- **TRACE**: pide al servidor que le envíe un mensaje de respuesta. Se suele emplear para diagnosticar posibles problemas en la conexión.
- **OPTIONS**: pide al servidor que le indique los métodos HTTP que soporta para una determinada URL.
- **CONNECT**: se emplea para transformar una conexión ya existente encriptada (https).
- **PATCH**: modifica parcialmente un recurso ya existente en el servidor.

Se define como **safe methods** aquellos que no deberían cambiar el estado del servidor, sólo recuperan información. Aunque el método GET en principio no debería cambiar nada, en la práctica puede utilizarse para enviar comandos al servidor (no recomendado).

Los **métodos idempotentes** son aquellos para los que múltiples peticiones deberían tener el mismo resultado. Los **safe methods**, por definición, deberían ser también idempotentes. Una mala implementación puede saltarse estas prácticas.

HTTP method	RFC	Request has Body	Response has Body	Safe	Idempotent	Cacheable
GET	RFC 7231	Optional	Yes	Yes	Yes	Yes
HEAD	RFC 7231	Optional	No	Yes	Yes	Yes
POST	RFC 7231	Yes	Yes	No	No	Yes
PUT	RFC 7231	Yes	Yes	No	Yes	No
DELETE	RFC 7231	Optional	Yes	No	Yes	No
CONNECT	RFC 7231	Optional	Yes	No	No	No
OPTIONS	RFC 7231	Optional	Yes	Yes	Yes	No
TRACE	RFC 7231	No	Yes	Yes	Yes	No
PATCH	RFC 5789	Yes	Yes	No	No	No

4. RESPUESTA HTTP

Una respuesta del servidor en el protocolo HTTP sigue la siguiente estructura:

Version-http SP código-estado SP frase-explicación CRLF
 (nombre-cabecera: valor-cabecera (“,” valor-cabecera)* CRLF)*
 CRLF

Cuerpo del mensaje

Se compone de:

- **Código de estado**: indica si la petición ha tenido éxito o ha habido algún error con ella.
- **Frase**: explicación del código.
- **Cabeceras**: misma estructura que en las peticiones.
- **Cuerpo**: la respuesta del servidor.

Ejemplo:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length

<html>
<head> <title> Título de nuestra primera página </title> </head>
<body>
¡Hola mundo!
</body>
</html>
```

Códigos de Estado

Números de tres dígitos que forman parte de las respuestas HTTP. Explican qué ha

sucedido al intentar llevar a cabo una petición. Hay varias categorías:

- **1xx: Mensajes**
 - 100 Continue: conexión aceptada, continúa la petición.
 - 101 Switching Protocols: cambiando protocolos.
 - 102 Processing: procesando todavía la petición.
- **2xx: Operación realizada con éxito.**
 - 200 OK: la petición se ha realizado con éxito.
 - 201 Created: petición OK y se ha creado un nuevo recurso.
 - 202 Accepted: petición aceptada pero el proceso no ha terminado todavía.
 - 204 No Content: sin contenido.
 - 205 Reset Content: le indica al user agent que recargue el documento.
 - 206 Partial Content: contenido parcial
- **3xx: redirección:**
 - 301 Moved Permanently: contiene la nueva URL.
 - 302 Found: el cambio es temporal.
 - 304: not modified: no modificado, se puede usar la versión cacheada.
 - 307 Temporary Redirect: parecido a 302.
 - 308 Permanent Redirect: parecido a 301
- **4xx: Error por parte del cliente.**
 - 400 Bad Request: petición malformada o inválida
 - 401 Unauthorized: el cliente tiene que registrarse.
 - 403 Forbidden: el cliente no tiene los derechos de acceso.
 - 404 Not Found: no existe el recurso indicado.
 - 405 Method Not Allowed: ese método HTTP no está permitido.
 - 409 Conflict: ejemplo – múltiples ediciones simultáneas.
 - 410 Gone: el recurso ha sido borrado permanentemente.
- **5xx: Error del servidor**
 - 500 Internal Server Error: el servidor no sabe gestionar la situación.
 - 502 Bad Gateway
 - 503 Service Unavailable: ejemplo – sobrecargado o en mantenimiento.
 - 504 Gateway Timeout.

5. HTTPS (HTTP Secure)

HTTPS es un protocolo de aplicación basado en el protocolo **HTTP** destinado a la transferencia segura de datos. Utiliza un cifrado basado en **SSL/TLS** (puerto 443).

TEMA 3 – HTML

LENGUAJES DE MARCADO

Los lenguajes de marcado o “markup languages” son un sistema para anotar documentos distinguible de texto (contenido). No se muestra cuando se procesa, sirve para dar formato al texto. Importa el orden.

Ejemplos: TeX, HTML, SGML, ...

INTRODUCCIÓN

HTML (HyperText Markup Language – Lenguaje de Marcado de Hipertexto) es el lenguaje en el que se escriben las páginas web. Se define como **hipertexto** al texto con referencias a otro texto (hyperlinks, hiperenlaces).

Se compone de etiquetas HTML (tags). Fue publicado en Internet por Tim Berners-Lee, científico del CERN, para facilitar la navegación entre diferentes documentos.

Evolución:

- 1995: La primera versión de HTML que se convirtió en un estándar fue la 3.0, cuando el World Wide Web Consortium (W3C) lo aceptó como uno de sus estándares.
- 1999: Llega HTML 4.0
- 2008: Comienzo de la definición de la versión 5.
- En la actualidad, se conoce simplemente como HTML y es un “Living Standard”.

ETIQUETAS

Sintaxis:

```
<nombre_etiqueta [atributo1=“valor1”, ...]>
```

```
[contenido de la etiqueta]
```

```
[</nombre_etiqueta>]
```

Atributos: algunos no tienen un valor asociado (atributos compactos). Si tienen valor tiene que ir entre comillas.

Contenido: texto, otras etiquetas o vacío.

Normalmente, las etiquetas se cierran.

<!DOCTYPE html>

- Declaración de tipo de documento. No es una etiqueta HTML.
- Se coloca al principio del documento HTML.

<html [lang=“en”]> </html>

- Elemento raíz del documento.
- Contenedor para todos los elementos HTML (excepto doctype)
- Recomendable usar el atributo **lang**.

<head> </head>

- Contenedor para meta información. No se renderiza.
- Es usado por navegadores, buscadores (SEO) y otros servicios.
- Se coloca dentro de <html> antes de <body>.

<title> </title>

- Dentro de <head>.
- Contiene el título que el navegador muestra en la barra.
- El contenido sólo puede ser texto.
- Es una etiqueta obligatoria y única → sólo puede haber una etiqueta <title>

<meta>

- **Añaden metainformación al documento** (atributos name y content):
 - `<meta charset="UTF-8">`
 - `<meta name="description" content="Descripción de la web">`
 - `<meta name="keywords" content="HTML, CSS, JavaScript">`
 - `<meta name="author" content="John Doe">`
 - `<meta name="viewport" content="width=device-width, initialscale=1.0">`
- **Siempre van dentro de <head>.**

Favicon

- **Se define en la etiqueta <link> dentro de head.**
- El nombre estándar es **favicon**.
- **El valor de type dependerá del formato del icono:**
 - `image/x-icon` → `.ico`
 - `image/png` → `.png`
 - `image/svg` → `.svg`
- Ejemplo:
 - `<link rel="icon" type="image/x-icon" href="/images/favicon.ico">`

<body> </body>

- **Contenido de la página web.**
- Todos los elementos visibles en la página web.
- **Sólo puede haber una etiqueta body.**

<!-- -->

- Para añadir **comentarios**, se puede usar en múltiples líneas.
- Lo ignora el navegador web.

Encabezados

- `<h1>` Encabezado 1 `</h1>`
- `<h2>` Encabezado 2 `</h2>`
- `<h3>` Encabezado 3 `</h3>`
- `<h4>` Encabezado 4 `</h4>`
- `<h5>` Encabezado 5 `</h5>`
- `<h6>` Encabezado 6 `</h6>`

Párrafos

- `<p>` Párrafo `</p>`

Salto de línea (line break)

- `
`

Estilos de texto

- `` Negrita (bold) ``
- `<i>` Cursiva (italic) `</i>`
- `` Énfasis (emphasis) `` `` Más énfasis ``
- `<small>` Tamaño reducido `</small>`
- `<u>` Subrayar (underline) `</u>`

Lista ordenada (ordered list)

``

```
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
```

```
</ol>
```

Lista sin ordenar (unordered list)

```
<ul>
```

```
<li>Coffee</li>
<li>Tea</li>
<li>Milk</li>
```

```
</ul>
```

Lista de descripción (description list)

```
<dl>
```

```
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink</dd>
```

```
</dl>
```

Imágenes

```
 </img>
```

Para añadir una imagen al texto. Los atributos usados son:

- **src:** ruta de la imagen.
- **alt:** texto alternativo por si no puede mostrarse la imagen (obligatorio).
- **Width y height:** ancho y alto de la imagen en píxeles.
- No se debe usar con / de cierre:

Tablas <table> </table>

```
<table>
```

```
<tr>
  <th>Month</th>
  <th>Savings</th>
</tr>
<tr>
  <td>January</td>
  <td>$100</td>
```

```
</tr>
```

```
</table>
```

Enlaces (anchor → <a>)

```
<a href="direccion">Texto del enlace</a>
```

División <div> </div>

- División o sección dentro del documento.
- Contenedor de otros elementos HTML.

```
<div class="myDiv">
```

```
<h2>This is a heading in a div element</h2>
```

```
<p>This is some text in a div element.</p>
```

</div>

Span

- Contenedor de línea.
- Permite remarcar una parte de un texto o documento.

<p>My mother has blue eyes.</p>

FORMULARIOS

<form action="/action_page.php" method="get"> </form>

- Etiqueta que define un formulario para que el usuario introduzca información.
- **action**: URL.
- **method**: get/post (escrito en minúsculas).

<label for="element_id" form="form_id"> </label>

- Etiqueta para los elementos del formulario.
- **form**: id del formulario al que pertenece la etiqueta.
- **for**: id del elemento asociado a la elemento.
- También se pueden incluir dentro los elementos asociados.

<input>

Campo de entrada en un formulario.

- **Text**: <input type="text" placeholder="Introduce el texto">
- **Radio**: <input type="radio">
- **Checkbox**: <input type="checkbox">
- **Button**: <input type="button">
- **Password**: <input type="password">
- **Hidden**: <input type="hidden">
- **Números**: <input type="number">
- ...

Acepta además los siguientes atributos:

- **required** → marca el campo como obligatorio
- **minlength** y **maxlength**.

<select> </select>

- Se emplea para crear una lista de elementos (drop-down list).
- Necesita el atributo id asociado a una **label**.
- Necesita el atributo **name** para cuando se envía el formulario.

<label for="cars">Choose a car:</label>

<select name="cars" id="cars">

<option value="volvo">Volvo</option>

<option value="saab">Saab</option>

<option value="mercedes">Mercedes</option>

<option value="audi">Audi</option>

</select>

Canvas <canvas> </canvas>

- Contenedor de gráficos.
- Se usa junto con JavaScript.

Metadatos <meta>

- Van dentro de <head>.
- Añaden información sobre los datos del documento.
- Los metadatos no aparecen en pantalla pero las máquinas los analizan.

ESTILO

Consideraciones generales de estilo:

- Los nombres de etiqueta y atributo siempre en minúscula
- Aplicar etiquetas de cierre
- Valores de atributos entre comillas
- Especificar los atributos alt, width y height de las imágenes
 - ``
- No usar espacios antes y después del signo igual
- Evitar líneas de código demasiado largas
- No añadir líneas en blanco ni aplicar sangrado sin necesidad
- Recomendable usar dos espacios para sangrar y no tabulador
- Usar siempre las etiquetas <html>, <head>, <body> y <title>

Caracteres especiales

En HTML hay algunos caracteres que tienen significado especial y por tanto no pueden emplearse directamente en el código HTML, sino que debe usarse una secuencia de escape equivalente:

- Empiezan con &.
- Terminan con ;.

La mayor parte de los navegadores modernos permiten escribir vocales con acento y caracteres como la ñ directamente en el HTML.

Secuencia de escape	Caracteres
<	<
>	>
&	&
"	"
á é í ó ú	á, é, í, ó, ú
ñ	ñ
¿	¿
 	(Espacio no separador)

Atributos

id="nombre_id"

- Nombre único para el elemento HTML.
- El nombre no puede contener espacios.
- Se usa con CSS y JS.

class="nombre_clase1 nombre_clase2"

- Uno o más nombres de clase para el elemento HTML.
- Puede tener múltiples nombres separados por espacios.
- Los nombres no pueden tener números en el primer carácter.
- Se usa con CSS y JS.

ROBOTS.TXT

Archivo que se puede incluir en el directorio raíz, con formato UTF-8 y que indica a los

web crawlers qué URLs pueden ser accedidas en el sitio.

Sirve para evitar sobrecargar el sitio con peticiones. No sirve para que un recurso no aparezca en los buscadores.

SECURITY.TXT

Archivo que se puede incluir en `/.well-known/security.txt`. Indica a alguien que haya encontrado una vulnerabilidad cómo proceder. Incluye información de contacto y puede incluir claves para cifrar la información.

XHTML

Versión **estricta** de HTML. No permite errores como etiquetas sin cerrar, etc.

HTML vs. XHTML Comparison	
HTML	XHTML
<pre><html> <header> <title>My Web Page </title> </header> <body> <p> Hello to my page!! </p> </body> </html></pre>	<pre><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml 1-transitional.dtd"> <html xmlns="http://www.w3.org/1999/xhtml"> <head> <meta http-equiv="Content-Type" content="text/html; charset=utf-8" /> <title>XHTML Sample Web Page</title> </head> <body> <p> Here is my content!! </p> </body> </html></pre>

TEMA 4 – CSS

INTRODUCCIÓN

CSS: Cascading Style Sheets (Hojas de estilo en cascada). Describe la presentación del documento, definiendo el *layout* (disposición), colores, fuentes, variaciones según el dispositivo, ...

Permite que múltiples documentos utilicen el mismo estilo. **Cascading**: prioridad para ver qué regla se aplica.

CSS es gestionado por el World Wide Web Consortium (W3C). Las versiones de CSS se conocen como **niveles** (levels). CSS1 se publicó en 1996, en la actualidad se usa CSS3.

Se va actualizando periódicamente.

Inserción en un documento:

- Como **archivo externo**: `<link rel="stylesheet" href="documento.css">`
- Como **CSS Interno**: `<style> ... </style>`
- **CSS – En línea**: `<p style="...">...</p>`

HTML nunca estuvo pensado para incluir etiquetas para dar formato, puesto que sirve sólo para describir el contenido. HTML3.2 introdujo etiquetas como ``, lo que empezó a complicar el código. A raíz de esto, el W3C creó CSS.

Nunca se debería insertar CSS en línea.

Comentarios

En CSS, los comentarios tienen la **sintaxis**: `/* ... */`, y pueden ser **multilínea**. Se pueden usar en el HTML dentro de la etiqueta `<style></style>`.

Sintaxis CSS

CSS funciona a base de reglas, es decir, declaraciones sobre el estilo de uno o más elementos:

```
h1 {color: red}
```

`h1` es el **selector**, indica los elementos de HTML que van a ser afectados.

`{color: red}` es el **bloque de declaración**. Cada bloque contiene una o más declaraciones separadas por `;`; `color` es el nombre de una propiedad, `red` es el valor.

Selectores

Los selectores pueden ser:

- **Simples**
- **Combinación**
- **Pseudo-clases**
- **Pseudo-elementos**
- **Atributos**

Selectores simples

- **Basados en el nombre de un elemento** (p, a, body, div, ...).
 - Sintaxis: `nombre_elemento {}`
- **Basados en el atributo class** de un elemento (puede estar repetido).
 - Sintaxis: `.nombre_clase {}`
 - Se puede especificar que solo afecte a un tipo determinado de elementos:
 - Sintaxis: `nombre_elemento.nombre_clase {}`
- **Basados en el atributo id** de un elemento (que tiene que ser único).
 - Sintaxis: `#nombre_id {}`
- **Selector universal** → selecciona todos los elementos del HTML.

- Sintaxis: `* {}`
- **Agrupación**
 - Sintaxis: `nombre_selector_1, nombre_selector_2 {}`

Selectores combinación

Llamados CSS Combinators. Relacionan selectores, pueden ser de los siguientes tipos:

- **Descendiente:** selecciona todos los elementos descendientes del elemento especificado (ejemplo: todos los `<p>` dentro de un `<div>`, ya sean descendientes de nivel 1, 2 o n).
 - Sintaxis: `selector_padre selector_descendiente {}`
- **Hijo:** Selecciona todos los elementos que son hijos del elemento especificado (ejemplo: todos los `<p>` dentro de un `<div>` que sean descendientes de nivel 1).
 - Sintaxis: `selector_padre > nombre_selector_hijo {}`
- **Hermano adyacente:** selecciona todos los elementos que son hermanos (tienen el mismo padre):
 - Sintaxis: `selector_simple_1 + selector_simple_2 {}`
- **Hermano general:** *General sibling*, selecciona todos los elementos que son hermanos y aparezcan después.
 - Sintaxis: `selector_simple_1 ~ selector_simple_2 {}`

Selectores pseudoclasas

Definen un estado especial de un elemento. Sintaxis: `selector:pseudo-class {}`

Ejemplos:

```
:hover → Selecciona al elemento que tiene el ratón encima
:focus → Selecciona al elemento que tiene el foco
:first-child → Selecciona al primer hijo
:last-child → Selecciona al último hijo
```

Selectores pseudoelementos

Identifican una parte especial de un elemento. Sintaxis: `selector::pseudo-element {}`

Ejemplos:

```
::after → Para insertar algo antes. Usar con la propiedad content
::before → Para insertar algo después. Usar con content
::first-letter → Selecciona la primera letra
::first-line → Selecciona la primera línea
::selection → La selección del usuario
```

Selectores de atributo

Selecciona un elemento con un atributo concreto. Puede estar asociado a un selector.

Sintaxis: `selector[attribute] {}`

También se puede seleccionar un **atributo** con un **valor determinado**.

Selecciona un elemento con un atributo y valores concretos. Puede estar asociado a un selector.

Sintaxis: `selector[attribute="value"] {}`

Otros selectores de atributo:

`[attribute~="value"]`

- Selecciona un elemento con un atributo cuyo valor contiene una palabra específica

`[attribute|= "value"]`

- Selecciona un elemento con un atributo cuyo valor empieza por una palabra

concreta

[attribute^=value]

- Selecciona un elemento con un atributo cuyo valor empieza de una forma concreta

[attribute\$="value"]

- Selecciona un elemento con un atributo cuyo valor termina de una forma concreta

[attribute*="value"]

- Selecciona un elemento con un atributo cuyo valor contiene un valor específico

Declaraciones

Se aplica el modelo de caja:

- **Contenido**: el contenido de la caja como texto o imágenes.
- **Padding**: área vacía alrededor del contenido. Es transparente.
- **Borde**: borde alrededor del padding y del contenido
- **Margen**: área vacía fuera del borde. Es transparente.

Margin

Espacio alrededor del elemento por fuera. Se definen margin-top, margin-right, margin-bottom, margin-left.

Valores (pueden ser negativos):

- **auto** (gestionado por el navegador)
- **longitud** (px, pt, cm, ...).
- **%** con respecto al ancho del elemento.
- **inherit**: heredado del padre

Se puede colapsar – Sintaxis:

- margin: top right bottom left;
- margin: top right bottom;
- margin: top+bottom right+left;
- margin: top+bottom+right+left;

Padding

Espacio alrededor del contenido del elemento y dentro de los bordes. Al igual que antes, existen padding-top, padding-right, padding-bottom, padding-left. Sus valores pueden ser:

- **longitud** (px, pt, cm, ...).
- **%** con respecto al ancho del elemento.
- **inherit**: heredado del padre

También se puede colapsar. Sintaxis:

- padding: top right bottom left;
- padding: top right bottom;
- padding: top+bottom right+left;
- padding: top+bottom+right+left;

width (ancho) y height (alto)

Ancho y alto del contenido. Además, se deberá sumar el padding, el borde y el margen. Valores:

- **auto**
- **longitud** (px, pt, cm, ...).
- **%** con respecto al ancho del elemento.
- **initial**.

- **inherit**: heredado del padre

max-width, max-height, min-width, min-height

Máximo/mínimo ancho y alto del contenido. Valores:

- **longitud** (px, pt, cm, ...).
- **%** con respecto al ancho del elemento.
- **none**: no hay máximo/mínimo.

text

Sobre el texto, **se puede definir**: **color, background-color, text-align**: left/right/center/justify; **direction, vertical-align, text-decoration, text-transform, text-indent, letter-spacing, line-height, word-spacing, white-space, text-shadow, ...**

Font

Respecto a la fuente, **se puede definir**: **font-family, font-style**: normal/italic/oblique; **font-weight**: normal/bold; **font-variant**: normal/small-caps; **font-size**;

Se puede colapsar → **font-style font-variant font-weight font-size/lineheight font-family**;

Position

Tipo de posicionamiento usado para el elemento:

- **static**: colocado **según el flujo normal de la página**. Por defecto.
- **relative**: **relativo a su posición por defecto**.
- **fixed**: **relativo al viewport, no se mueve al hacer scroll**.
- **absolute**: **relativo al ancestro más cercano**.
- **sticky**: **se queda "pegado" al hacer scroll**.

Para posicionar se usan: top, right, bottom, left y z-index.

Valores – Unidades

Las unidades pueden ser:

- **Absolutas**: cm, mm, in, px, pt
- **Relativas**: **em** (relativo al tamaño de la fuente), **rem** (relativo al tamaño de fuente del elemento raíz), **ch** (relativo al tamaño del cero), **%** (relativo al elemento padre)

Valores – Colores

Los colores se pueden especificar por:

- **nombres**: lista de colores
- **RGB**: **reg**(red, green, blue)
- **HEX**: #XXXXXX
- **HSL**: **hsl**(hue, saturation, lightness)
- **RGBA**: **reg**(red, green, blue, alpha)
- **HSLA**: **hsl**(hue, saturation, lightness, alpha)

Specifity

La especificación **se usa para la resolución de conflictos**. **Cuanto más específico sea un elemento, mayor prioridad tendrá**: **id > class > element**.

En caso de igualdad se aplica el que aparezca después en el archivo.

!important

Permite añadir más importancia a una propiedad o valor. Sobrescribe todas las reglas de esa propiedad.

Ejemplo:

```
p {background-color: yellow !important; }
```

RESPONSIVE WEB DESIGN

El diseño **responsive** es el que hace que la página web se vea bien en todos los dispositivos, haciendo uso sólo de HTML y CSS. Se debe **evitar** scroll horizontal.

Viewport

Se debe usar siempre la siguiente etiqueta para que se visualice bien el contenido en los móviles:

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

Media Queries

Permiten establecer breakpoints en el CSS según:

- Ancho y alto del dispositivo.
- Orientación
- Resolución

Se emplea la etiqueta **@media**:

```
@media [not|only] mediatype and (mediafeature and|or|not mediafeature) {  
    CSS-Code;  
}
```

Siempre se definen **al final** del CSS.

- **Mediatype**: normalmente será **screen**.
- **Mediafeature**: normalmente será **max-width** o **min-width**.

Se pueden enlazar diferentes CSS:

```
<link rel="stylesheet" media="screen and (min-width: 900px)"  
href="widescreen.css">  
<link rel="stylesheet" media="screen and (max-width: 600px)"  
href="smallscreen.css">
```

Ejemplo con 4 columnas:

```
.column {  
    float: left;  
    width: 25%;  
    padding: 20px;  
}  
  
@media screen and (max-width: 992px) {  
    .column { width: 50%; }  
}  
  
@media screen and (max-width: 600px) {  
    .column { width: 100%; }  
}
```

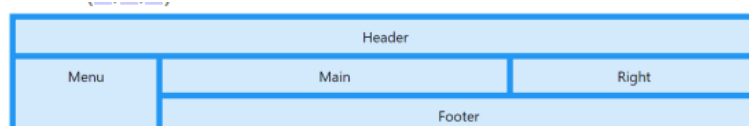
GRID

No es un estándar oficial. Define dos dimensiones (filas y columnas) que permiten dividir la página en **regiones**.

Es la evolución de la **table** (que nunca debería usarse para el layout).

No está soportado por Internet Explorer.

Para su uso, se debe declarar un contenedor como **display: grid**.



FLEXBOX

FlexBox (Flexible Box) define una dimensión (filas o columnas). Permite facilitar el layout, alineamiento y distribución de elementos.

Permite alterar el tamaño de los elementos para rellenar el espacio.

No está soportado por Internet Explorer.

Para su uso, se debe declarar un contenedor como **display: flex**;



SaSS

SaSS (*Syntactically Awesome Style Sheets*) es un preprocesador CSS para trabajar con proyectos grandes, basado en Ruby.

Permite:

- Variables
- Nesting
- Módulos
- Mixins
- ...

Less

Less (*Leaner Style Sheets*) es un preprocesador CSS inspirado en Sass y basado en JavaScript.

Permite:

- Variables
- Nesting
- Módulos

3. CSS FRAMEWORKS

Los frameworks son librerías que facilitan el desarrollo web. Hacen uso de CSS y pueden tener JS. Resetean la hoja de estilos, empleando un layout en forma de grid (responsive).

Permiten un diseño Mobile First, que incluyen tipografía, fuentes, iconos, ...

Entre sus **desventajas**: se debe aprender a trabajar con el framework, generan mucho código extra que no se utilizará, habrá una gran similitud entre todas las páginas web que lo usen y la aplicación tendrá una alta dependencia del framework utilizado.

4. BOOTSTRAP

Bootstrap es un framework CSS que responde al lema "Build fast, responsive sites". Se emplea para el prototipado rápido.

Ofrece un gran ecosistema, gestionado por Twitter y con una gran cantidad de componentes, como iconos. Tiene soporte para Less y Sass.

Uso

- Añadir dentro de head el enlace al CDN del CSS.
- Añadir etiqueta *meta viewport* para que sea responsive.
- Opcionalmente añadir los JS al final de body.

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1>
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.1/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
F3w7mX95PdgyTmZZMECAngseQB83DfGTowi0iMjiWaeVhAn4FJkqJByhZMI3AhiU"
crossorigin="anonymous">
</head>
```

Archivos

Archivos .min:

- Versión reducida del archivo original.
- Se eliminan saltos de línea y espacios innecesarios.
- Se usa en producción.
- Bootstrap.css (200KB) → bootstrap.min.css (159KB).

Archivos .map:

- Source maps.
- Para trabajar con herramientas de desarrollo.
- Conversión que se ha usado para generar el archivo minimizado.

Archivos rtl (Right to Left): soporte para escritura de derecha a izquierda.

COMPONENTES

Containers:

- Los **containers** (contenedores) son el elemento de layout más básico. Son un requisito para usar Grid:

```
<div class="container">
  <!-- Content here -->
</div>
```

Grid:

- Basado en Flexbox. Tiene clases predefinidas para trabajar con columnas.

```
<div class="container">
  <div class="row">
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
  </div>
</div>
```

Alerts

- Mensajes contextuales. Son el resultado de una acción del usuario.


```

<div class="alert alert-primary" role="alert"> Mensaje </div>
<div class="alert alert-secondary" role="alert"> Mensaje </div>
<div class="alert alert-success" role="alert"> Mensaje </div>
<div class="alert alert-danger" role="alert"> Mensaje </div>
<div class="alert alert-warning" role="alert"> Mensaje </div>
<div class="alert alert-info" role="alert"> Mensaje </div>
<div class="alert alert-light" role="alert"> Mensaje </div>
<div class="alert alert-dark" role="alert"> Mensaje </div>

```

Carousel

- Permite ciclar elementos (imágenes o texto).
- No debería usarse.

```

<div id="carouselExampleSlidesOnly" class="carousel slide" data-bs-
ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      
    </div>
    <div class="carousel-item">
      
    </div>
    <div class="carousel-item">
      
    </div>
  </div>
</div>

```

Forms:

- Formularios

```

<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Email
address</label>
    <input type="email" class="form-control"
id="exampleInputEmail1" aria-describedby="emailHelp">
    <div id="emailHelp" class="form-text">We'll never share your
email with anyone else.</div>
  </div>
  <div class="mb-3">
    <label for="exampleInputPassword1" class="form-
label">Password</label>
    <input type="password" class="form-control"
id="exampleInputPassword1">
  </div>
  <div class="mb-3 form-check">
    <input type="checkbox" class="form-check-input"

```

```

        id="exampleCheck1">
        <label class="form-check-label" for="exampleCheck1">Check me
        out</label>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>

```

Navbar:

- Cabecera de navegación.
- Colapsa con pantallas pequeñas y se puede desplegar.

```

<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <a class="navbar-brand" href="#">Navbar</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
    data-target="#navbarSupportedContent" aria-
    controls="navbarSupportedContent" aria-expanded="false" aria-
    label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
        (...)
    </div>
</nav>

```

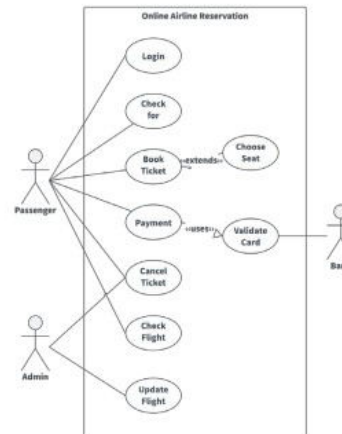
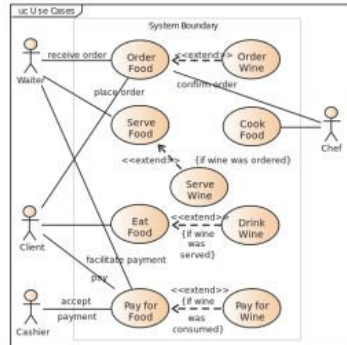
Otros

Hay muchos componentes más, como Popovers, barras de progreso, botones, iconos, ...

PROYECTO – PARTE 1

Diagramas de caso de uso

Muestran la interacción del usuario con el sistema. Hay diferentes tipos de usuarios:



Requisitos funcionales

Representan el comportamiento básico del sistema, son lo que el sistema hace o no hace. Representan características del producto, se centran en los requisitos del cliente.

Son sencillos. Se numeran de la manera RF1, RF2...RFN.

Requisitos no funcionales

Describen cómo se deberían aplicar los requisitos funcionales. No afectan a la funcionalidad básica del Sistema. Si no se cumplen, el sistema seguirá funcionando. Están relacionados con la usabilidad. Se numeran RNF1, RNF2, ..., RNFN.

TEMA 5 – JAVASCRIPT

1. INTRODUCCIÓN

JavaScript (JS) fue creado originalmente para que las webs estuvieran vivas. No está relacionado con Java.

Los programas son scripts, basado en la especificación ECMAScript (ES). Hace falta un JavaScript engine para ejecutarlo:

- V8 en Chrome y Opera
- SpiderMonkey en Firefox

Características principales:

- **Multiplataforma**
- **Orientado a objetos**
- Completa **integración con HTML / CSS**
- Las cosas simples son sencillas
- **Soporte de casi todos los navegadores**
- **Habilitado por defecto en los navegadores.**
- **Tipado dinámico y débil.**
- **Versión de servidor: node.js**

JS en el navegador no tiene acceso a funciones del SO:

Está muy limitada la escritura de archivos.

Diferentes pestañas / ventanas en general no se conocen entre ellas

“Same Origin Policy”

Una web puede comunicarse con el servidor del que vino:

Está muy limitada la recepción de datos de otros dominios.

Lenguajes transpiled: lenguajes que tienen otra sintaxis y se convierten a JS. Ejemplos: CoffeeScript, TypeScript, Flow, Dart o Brython.

Compatibilidad: no todas las funcionalidades están soportadas por todos los navegadores.

Consola de desarrollo: Chrome (Ctrl + Shift + I), Firefox, Edge (F12).

Se puede utilizar como:

- **Código incrustado en el HTML:**

```
<script>
    alert( 'Hello, world!' );
</script>
```
- **Script externo**

```
<script src="/path/to/script.js"></script>
```

Comentarios

De una línea: **con //**

//Esto es un comentario

Multilínea: con /* */ (No se pueden anidar)

/* Comentario multilínea */

Interacción

alert: muestra un mensaje al usuario:

```
alert("Hola");
```

prompt: pide al usuario que introduzca información:

SINTAXIS: `prompt(title, [default]);`

```
let result = prompt("¿Cuántos años tienes?", "");  
console.log(result);  
console.log(typeof(result)); //string
```

confirm: le solicita al usuario que confirme

SINTAXIS: `result = confirm(question);`

```
let str = confirm("¿Estás de acuerdo?");  
console.log(str) //true o false
```

console.log: imprime un mensaje por la consola del navegador:

```
//console.log(mensaje);  
console.log("Esto se imprime en la consola");
```

Statements

Las declaraciones se escriben separadas por ';' aunque se puede omitir en un salto de línea (no recomendable).

Variables

Las variables permiten almacenamiento de datos con nombre:

```
let variable = valor;  
let message;  
message = 'Hello!';  
console.log(message);
```

Puede cambiar el tipo de datos, aunque con mucho cuidado:

```
let message;  
console.log(typeof(message)) //undefined  
message = 'Hello!';  
console.log(typeof(message)) //string  
message = 1234  
console.log(typeof(message)) //number
```

Sintaxis:

- Solo pueden contener letras, dígitos, \$ o _
- El primer carácter no puede ser un dígito.
- Se recomienda usar lower **camelCase** para múltiples palabras.

Constantes

Las constantes son variables que no cambian de valor. Se debe asignar un valor al declararlas:

```
const constante = valor;
```

Para constantes conocidas recomendable que solo contengan mayúsculas y las palabras separadas por _

```
const COLOR_RED = "#F00";
```

Para otras constantes usar lower camelCase:

```
const pageLoadTime = /* time taken by a webpage to load */;
```

let vs var

En general → **no usar var**, ya que **var** no tiene scope de bloque, tiene scope de función.

Además, **var** se puede redeclarar, **let** no. Con **var** no hace falta poner var al declarar la variable, lo que permite que las variables se declaren después de su uso (**hoisting**).

Tipos de datos – Number

Tanto para integers como coma flotante.

Valores enteros entre $-(2^{53}-1)$ y $(2^{53}-1)$. Para valores numéricos fuera de este rango se usa **BigInt** (int normal, terminado en n):

```
const bigint = 1234567890123456789012345678901234567890n;
```

Valores numéricos especiales:

- **Infinity, -Infinity**
- **NaN** (Not a Number)

Tipos de datos – string

Declaración: entre comillas dobles o simples:

```
let str = "Hello";
```

Los **template literals** permiten introducir variables dentro del string. Usan ``` en lugar de comillas:

```
let phrase = `can embed another ${str}`;
```

Tipos de datos – boolean

Declaración: **true o false**:

```
let nameFieldChecked = true;
```

Pueden ser el resultado de una comparación

Tipos de datos – null

Declaración: se usa la palabra clave **null**.

```
let age = null;
```

No es una referencia a un objeto que no existe. Es un valor especial que representa nada, vacío, valor desconocido.

typeof devuelve object por compatibilidad histórica, aunque no lo sea.

Tipos de datos – undefined

Representa que no se ha asignado ningún valor. Declaración:

```
let age;
```

Tipos de datos – Conversiones

- Conversión a string: usando **String()**
- Conversión a Number: usando **Number()** o **+**
- Conversión a boolean: usando **Boolean()**
- Conversiones explícitas: al intentar operar con strings y números.

```
console.log("2" + 2); //22
console.log(4 + 5 + "px"); //9px
console.log("$" + 4 + 5); // $45
console.log("2" / "5"); //0.4
console.log(6 - "2"); //4
console.log(false + 34); //34
console.log(true + 34); //35
console.log(true + "34"); //true34
```

Operadores

- Suma / resta / multiplicación / división: **+, -, *, /**
- Resto: **%**

- Exponente: **
- Incremento/decremento: ++/--
- Operadores de bit: AND & – OR | – XOR ^ – NOT ~ – LEFT SHIFT << – RIGHT SHIFT >> – ZERO-FILL RIGHT SHIFT >>>

Comparaciones

- Mayor que >
- Menor que <
- Mayor o igual que >=
- Menor o igual que <=
- Igual ==
- Distinto !=

Strings: letra a letra, según el orden Unicode.

Tipos distintos: se convierten a números. Si no se puede convertir, devuelve false.

- Para que no se haga conversión, usar la igualdad estricta ===.
- Si los tipos son distintos, === devuelve false.

Condicionales – if

Se evalúa la expresión y se convierte a boolean.

```
if (...) {
  ...
} else if (...) {
  ...
}
else {
  ...
}
```

El operador ? implementa el operador ternario:

```
let accessAllowed = (age > 18) ? true : false;
```

Condicionales - Operadores lógicos

OR ||

- Se evalúa de izquierda a derecha.
- **Devuelve** el primer valor del primer operando que se evalúe a **true**.
- Si se llega al final, se devuelve el valor del último operando

```
let firstName = "";
let lastName = "";
let nickName = "SuperCoder";
console.log( firstName || lastName || nickName || "Anonymous");
```

AND &&

- Se evalúa de izquierda a derecha.
- **Devuelve** el primer valor del primer operando que se evalúe a **false**.
- Si se llega al final, se devuelve el valor del último operando.
- Tiene **más prioridad** que **OR**.

NOT !

- Convierte el operando a booleano y devuelve el valor inverso.

Nullish coalescing ??

- **a ?? b** → devuelve a si está definido (no es null ni undefined) y si no b.
- Provee un valor por defecto a una variable.
- No es compatible con todos los navegadores.
- Por motivos de seguridad, **no se puede utilizar junto con && o || sin usar paréntesis**.

Condicionales – switch

- Reemplaza múltiples if con igualdad estricta **===**.
- Empieza ejecutando desde la primera condición **true**.
- **break** para terminar la ejecución del **switch**.
- **default** equivalente al else.

Bucles

while se ejecuta mientras la condición sea **true**

```
while (..) {  
    ...  
}
```

do..while se ejecuta al menos una vez.

```
do {  
    ...  
} while(...)
```

for:

```
for (let i = 0; i < 3; i++) { console.log(i); }
```

break: fuerza la terminación del bucle.

continue: fuerza al bucle a seguir con la siguiente iteración.

etiquetas (labels): **identifican al bucle**. Sirven como referencia para **break / continue**. No se pueden usar para saltar a cualquier lado (no son goto).

outer:

```
for (let i = 0; i < 3; i++) {  
    for (let j = 0; j < 3; j++) {  
        let input = prompt(`Value at coords (${i},${j})`, '');  
        if (!input) break outer;  
        // do something with the value...  
    }  
}
```

Funciones

Las funciones son los **principales bloques de construcción de JavaScript**. **Permiten la reutilización del código**.

Las variables declaradas dentro no son accesibles fuera, aunque las variables exteriores son accesibles y modificables dentro.

Si se declara la misma variable dentro y fuera, la de dentro oscurece la de fuera (**shadowing**). Lo mismo sucede en condicionales y bucles.

```
function f() {  
    ...  
}
```


Parámetros

Se puede llamar a la función con menos parámetros, en cuyo caso se consideran `undefined`.

```
function f(arg1, arg2) {  
    ...  
}
```

Se puede incluir un valor por defecto, que puede ser a su vez una llamada a otra función:

```
function f(arg1 = "ejemplo", arg2=4) {  
    ...  
}
```

Se pueden pasar también múltiples parámetros a una función definida sin parámetros:

```
function foo() {  
    for (let i = 0; i < arguments.length; i++) {  
        console.log(arguments[i]);  
    }  
}
```

Otra sintaxis para múltiples parámetros puede ser la siguiente:

```
function my_log(x, ...args) {  
    console.log(x, args, ...args);  
}
```

return

La sentencia `return` se utiliza para devolver un resultado en la función. Si no existe, la función devuelve `undefined`. Se puede usar vacío (`return;`) para terminar la ejecución de la función.

Expresiones

Se puede asignar una función a una variable, añadiendo `;` al final:

```
let sayHi = function() {  
    console.log( "Hello" );  
};
```

```
console.log(sayHi); //se imprime el código de la función  
console.log(sayHi()); // Hello\nundefined
```

Una declaración de función se puede usar antes, una expresión no.

```
sayHi("John"); // Hello, John  
saludar("John"); // ReferenceError: saludar is not defined  
function sayHi(name) {  
    alert( `Hello, ${name}` );  
}  
let saludar = function(name) {  
    alert( `Hello, ${name}` );  
};
```

Arrows → lambda

Los arrows son los equivalentes a funciones lambda (funciones anónimas). Se trata de una forma concisa de declarar una función.

```
let sum = (a, b) => a + b;
```

Pueden utilizar múltiples líneas con **{}**, en ese caso necesitan **return**:

```
let sum = (a, b) => {  
  let result = a + b;  
  return result;  
};
```

Arrays

Para crear arrays se utiliza **[]**:

```
let fruits = ['Apple', 'Banana'];
```

Para acceder a un elemento de un array se usa **[índice]**

```
console.log(fruits[0]);
```

La longitud del array la contiene el atributo **length**:

```
console.log(fruits.length);
```

Para recorrer el array, se puede utilizar el bucle **forEach**:

```
fruits.forEach(function(item, index, array) {  
  console.log(item, index);  
});
```

Para añadir un elemento a un array, se usa la función **push**:

```
let newLength = fruits.push('Orange');
```

Para eliminar el último elemento se usa la función **pop**:

```
let last = fruits.pop();
```

Para buscar el índice de un elemento se usa la función **indexOf**:

```
let pos = fruits.indexOf('Banana');
```

Para eliminar un elemento, se puede usar la función **splice**:

```
let removedItem = fruits.splice(pos, 1);
```

Objetos

Almacenan colecciones de varios **datos**. Para crearlos, hay dos sintaxis posibles.

```
let user = new Object(); // "object constructor" syntax
```

```
let user = {}; // "object literal" syntax
```

Se pueden inicializar con datos mediante **clave: valor** separados por comas:

```
let user = {  
  name: "John",  
  age: 30  
};
```

Para acceder a la información:

```
console.log(user.name);  
console.log(user["name"]);
```

Se pueden añadir nuevas propiedades:

```
let user = {  
  name: "John",  
  age: 30  
};
```

```
user.isAdmin = true;
```

También se pueden borrar propiedades con **delete**:

```
delete user.age;
```

Una propiedad puede tener más de una palabra, pero entonces hay que usar comillas:

```
let user = {  
  name: "John",  
  age: 30,  
  "likes birds": true  
};
```

A esas propiedades se deberá acceder con []:

```
console.log(user["likes birds"]);  
console.log(user["age"]);
```

Para comprobar si existe una propiedad, comparamos con **undefined**:

```
console.log(user.noSuchProperty === undefined); // true
```

Se puede usar el operador **in**, que devuelve true si existe:

```
console.log("noSuchProperty" in user); // false  
let key = "age";  
console.log(key in user); // false
```

Para recorrer objetos se usan bucles **for**:

```
for (let key in user) {  
  alert(key); // name, age, isAdmin  
  alert(user[key]); // John, 30, true  
}
```

Se pueden anidar objetos:

```
let user = {  
  name: "John",  
  birthday: {  
    year: 1990,  
    month: "November"  
  }  
};
```

APIs

Hay múltiples interfaces y APIs predefinidas. Las más útiles incluyen:

- Propiedad **onload**: permite ejecutar algo cuando el recurso ha cargado:

```
window.addEventListener('load', (event) => {  
  init();  
});
```
- Propiedad **onclick**: ejecuta una acción cuando se hace click sobre un elemento.

```
var element = document.getElementById("boton");  
element.addEventListener("click", myScript);
```
- Para acceder al documento:

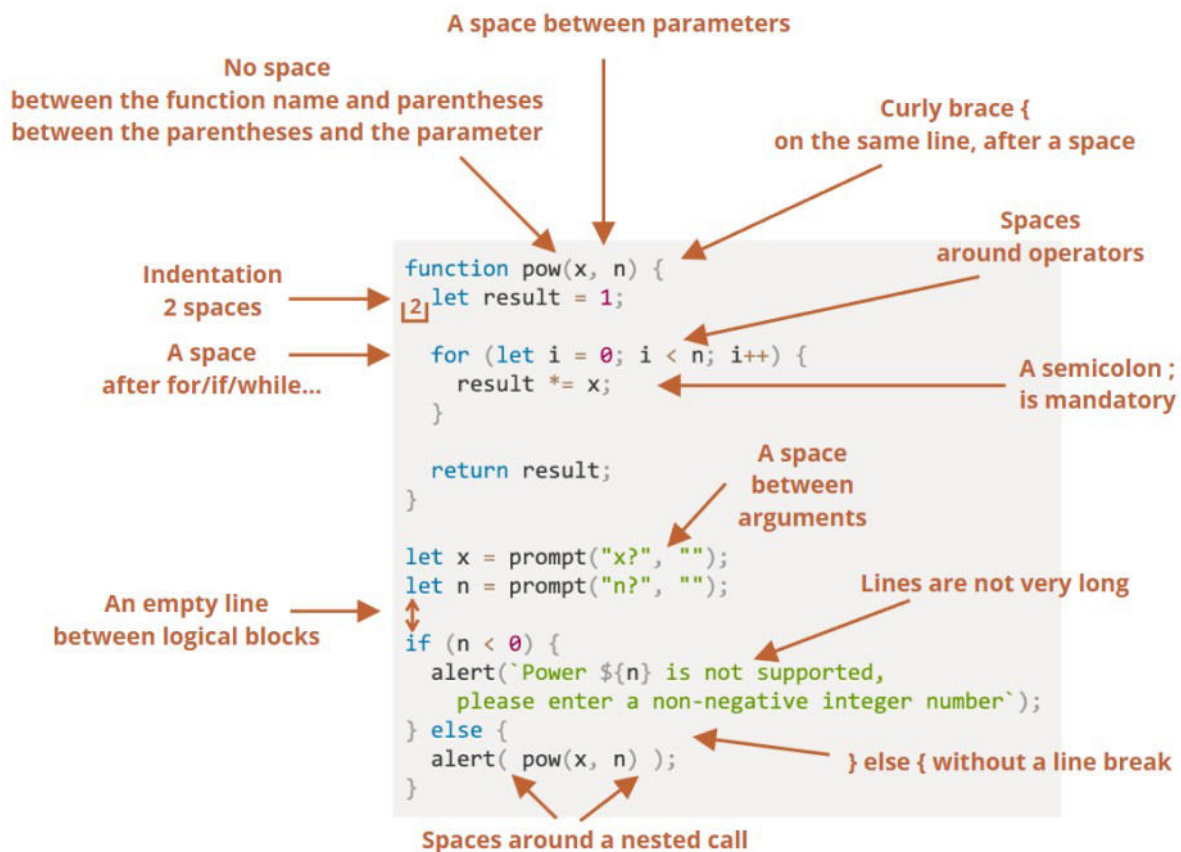
```
let markup = document.documentElement.innerHTML;
```
- Para acceder a elementos:

```
let myElement = document.getElementById("intro");  
let x = document.getElementsByTagName("p");
```

- ```
let x = document.getElementsByClassName("intro");
let x = document.querySelectorAll("p.intro");
```
- Para añadir elementos nuevos:  

```
let element = document.getElementById("new");
element.appendChild(tag);
```

## Guía de estilo



## Callback

Funciones pasadas como argumento a otra función. Se invocan cuando se completa alguna acción o rutina. Pueden ser síncronas, aunque habitualmente se usan asíncronamente:

```
function greeting(name) {
 alert('Hello ' + name);
}

function processUserInput(callback) {
 let name = prompt('Please enter your name. ');
 callback(name);
}

processUserInput(greeting);
```

## Input – Teclado

Los eventos principales del teclado son los siguientes:

- **keydown**: cuando se pulsa una tecla.
- **keypress**: mientras se pulsa una tecla (múltiples ocasiones). **Deprecado.**
- **keyup**: cuando se libera la tecla.

```
document.addEventListener('keydown', logKey);
function logKey(e) {
 console.log(e.code);
}
```

## Input – Ratón

### Eventos del ratón:

- **click**: cuando se pulsa y luego se libera el ratón.
- **dblclick**: doble click.
- **mouseup**: cuando se libera el botón del ratón.
- **mousedown**: cuando se pulsa el botón del ratón.
- **mousemove**: mientras se mueve el ratón.

```
document.addEventListener('click', click);
function click(e) {
 console.log(e);
}
```

## Canvas

### Elemento HTML que nos permite dibujar gráficos

```
<canvas id="myCanvas" width="500" height="500">fallback content</canvas>
```

Por defecto es un rectángulo blanco.

- **fallback content**: contenido recomendable añadir para que se muestre en navegadores antiguos que no soporten la etiqueta (ej: IE < 9)

Necesita la etiqueta de cierre.

### Para trabajar con canvas con JS necesitamos el contexto 2d:

```
var canvas = document.getElementById("myCanvas");
var ctx = canvas.getContext("2d");
```

## Canvas – Formas

### Dibujar una línea:

```
ctx.moveTo(0, 0); //Arriba a la izquierda
ctx.lineTo(200, 100); //Coordenadas en px
ctx.stroke();
```

### Dibujar una figura:

```
ctx.moveTo(100, 200);
ctx.lineTo(200, 200);
ctx.lineTo(150, 100);
ctx.lineTo(100, 200);
ctx.stroke();
//ctx.fill(); //rellena
```

### Dibujar un rectángulo:

```
//strokeRect(x, y, width, height)
ctx.strokeRect(50, 50, 100, 50);
```

### Dibujar un rectángulo lleno:

```
//fillRect(x, y, width, height)
ctx.fillRect(50, 50, 100, 50);
```

### **Borrar un rectángulo:**

```
//clearRect(x, y, width, height)
ctx.fillRect(50, 50, 100, 50);
ctx.clearRect(60, 60, 80, 30);
```

### **Dibujar un círculo:**

```
//arc(x, y, radius, startAngle, endAngle, counterclockwise)
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.stroke();
```

### **Dibujar un círculo relleno:**

```
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.fill();
```

### **Añadir Texto:**

```
ctx.font = "30px Arial";
ctx.fillText("Hello World", 10, 50);
```

### **Añadir StrokeText:**

```
ctx.font = '48px serif';
ctx.strokeText('Hello world', 10, 50);
```

### **Definir color:**

```
//fillStyle = color
//strokeStyle = color
ctx.fillStyle = 'orange';
ctx.strokeStyle = 'blue';
ctx.beginPath();
ctx.arc(95, 50, 40, 0, 2 * Math.PI);
ctx.stroke();
ctx.fill();
```

### **Imágenes:**

```
//drawImage(image, x, y)
//drawImage(image, x, y, width, height)
let img = new Image();
 img.onload = function() {
 ctx.drawImage(img, 0, 0);
 };
img.src = 'image.png';
```

### **Canvas**

**Hay muchas más funcionalidades** (Transformaciones, Gradientes, Modificación de Imágenes, Manipulación de Píxeles, Efectos, Gráficos 3D, ...).

**Existen diferentes librerías para facilitar el trabajo.**

### **Canvas – Animaciones**

**Para dibujar un frame necesitamos:**

1. **Limpiar el Canvas**

## 2. Dibujar los elementos que queramos

Para controlar las animaciones:

- `setInterval(function[, delay]);`
  - Para invocar una función cada *delay* milisegundos
- `setTimeout(function[, delay]);`
  - Invoca una función una sola vez dentro de *delay* milisegundos
- `requestAnimationFrame(callback);`
  - Ejecuta una animación antes del siguiente repaint del navegador.
  - Se ejecuta una sola vez.
  - La tasa de refresco es unas 60 veces por segundo.
  - La tasa de refresco puede variar según el display.
  - La tasa de refresco puede ser más lenta si el navegador está sobrecargado.

## 5. JQUERY

### Introducción

jQuery es una librería JavaScript fast, small and feature rich. Permite recorrer y manipular HTML y CSS, gestión de eventos, efectos y animaciones, Ajax, etc.

Es compatible con muchos navegadores, usada por más de 70 Millones de Webs.

Se puede **descargar**:

- Archivos sin comprimir (.js), comprimidos (.min.js) y map files (.min.map).

Se puede emplear como enlace a un **CDN**:

- Añadirlo al final de body.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></
script>
```

### Uso

Para acceder a jQuery se usa el símbolo \$, con la sintaxis:

**\$(selector).acción()**

- **Selector**: elemento HTML sobre el que queremos actuar.
- **Acción**: acción a realizar sobre el elemento.

jQuery también tiene acciones que no necesitan selector:

`$.action()`

### Selectores

- **window**: `$(window)`
- **document**: `$(document)`
- **this**: `$(this)`
- **Selectores CSS**
- **:contains("texto")** → elementos que contengan "texto". Se debe tener cuidado con las comillas simples, dobles y template literals.
  - Se pueden usar otros selectores antes de los 2 puntos.

### Eventos

- **.click(handler)** → se ejecuta la función «handler» cuando se hace click sobre el elemento.
  - Ej: `$("p").click(function() { ... });`
- Se debe tener cuidado con las funciones con argumentos → no se pueden pasar

directamente, es necesario utilizar una función **wrapper**:

- Sintaxis: `$(“p”).click( function() { info(“hola”); });`
- **\$(handler)**: se ejecuta cuando el Document Object Model ha terminado de cargar:
  - Punto de comienzo para ejecutar JS.
  - Antiguamente: `$(document).ready(handler) → Deprecado`
  - Pueden no haber cargado los assets (imágenes, vídeos, ...)
- **.on(“load”, handler)**: se ejecuta handler cuando se ha terminado de cargar el elemento.
  - En vez de load, se pueden usar otros nombres de evento: click, keydown, ...
- **\$(window).on(“load”, handler)**: handler se ejecuta cuando se han terminado de cargar los assets de la página.
  - Cuidado → pueden haber pasado algunos segundos en los que el usuario ha interactuado con la página.
- **.on(handler)**: permite asociar múltiples acciones:

```
$(“p”).on({
 mouseenter: function(){
 $(this).css("background-color", "green");
 },
 mouseleave: function(){
 $(this).css("background-color", "lightblue");
 },
 click: function(){
 $(this).css("background-color", "yellow");
 }
});
```
- Hay múltiples eventos de ratón, teclado, formularios, ...
- **Todos los handlers disponen de información del evento que lo activó** en un objeto evento.
  - `$(“p”).click(function(e) { console.log(e); });`
- **Propiedades del objeto evento**:
  - **e.pageX, e.PageY**: posición del ratón.
  - **e.type**: tipo de evento.
  - **e.target**: elemento HTML que inició el evento.
  - **e.timestamp**: tiempo dese que se cargó la página.
- **Funciones del objeto evento**:
  - **e.preventDefault()**: previene la acción por defecto.
  - **e.stopPropagation()**: evita que el evento se propague a los hijos.

## CSS

- **.css(propertyname)**: devuelve el valor de una propiedad CSS.
  - En caso de que haya múltiples elementos sólo devuelve el del primero.
- **.css(propertyname, value)**: cambiar el valor de una propiedad.
- **.css({ “propertyname”: “value”, “propertyname”: “value” })**: actualizar múltiples propiedades CSS a la vez.
- **.addClass(clase) / .removeClass(clase)**: añade / elimina la clase de los



elementos HTML

- **toggleClass(clase)**: añade la clase a los elementos si no la tienen o la elimina si ya la tienen.
- Más comandos CSS:
  - .hasClass()
  - .height()
  - .width()
  - .position()

## Efectos

- **.hide()**: oculta el elemento.
- **.show()**: muestra el elemento.
- **.toggle()**: oculta el elemento si está visible y lo muestra si está oculto.
- **.fadeIn()**: muestra el elemento con efecto fade
- **.fadeOut()**: oculta el elemento con efecto fade
- **.fadeToggle()**: muestra/oculta el elemento con efecto fade.
- **.slideDown()**: desliza el elemento hacia abajo para mostrarlo.
- **.slideUp()**: desliza el elemento hacia arriba para mostrarlo.
- **.slideToggle()**: Desliza el elemento hacia arriba/abajo

Parámetros de los efectos:

**\$(“p”).effect(speed, callback)**

- **speed**: “slow”, “fast”, un número que representa los **milisegundos**.
- **Callback**: función que se ejecuta cuando termina el efecto.

Se pueden encadenar múltiples efectos (chaining). Cuando termina uno se ejecuta el siguiente, excepto los que son instantáneos.

## Manipulación del DOM

- **.html()**: devuelve el contenido del primer elemento HTML.
  - **\$(“div”).html()**
- **.html(contenido)**: modifica el contenido de todos los elementos HTML
  - **\$(“div”).html(<span>Sin contenido</span>);**
- **.text()**: devuelve el texto combinado de todos los elementos y sus descendientes
  - Ignora las etiquetas html
- **.text(texto)**: modifica el texto de todos los elementos HTML
  - Trata las etiquetas HTML como texto.
- **.val()**: devuelve el valor del primer elemento. Se usa con elementos de formulario (input, select, textarea, ...).
- **.val(valor)**: modifica el valor de todos los elementos HTML.
- **.attr(nombreAtributo)**: devuelve el valor de un atributo HTML.
- **.attr(nombreAtributo, valor)**: modifica el valor del atributo.
- **.prepend(contenido)/ .append(contenido)**: añade contenido al principio / final dentro de cada elemento HTML.
  - Permiten añadir múltiple contenido.
- **.before(contenido) / .after(contenido)**: añade contenido antes / detrás de cada elemento HTML.
  - Permiten añadir múltiple contenido

- **.remove()**: elimina los elementos HTML y todo su contenido
- **.remove(selector)**: elimina los elementos HTML que cumplan el selector.
  - Ejemplo: `$("p").remove(":contains('Lorem')");`
- **.empty()**: vacía los elementos HTML.
- **.children()**: devuelve los hijos de los elementos HTML.
- **.parent()**: devuelve el elemento padre de los elementos HTML.
- **.siblings()**: devuelve los hermanos de los elementos HTML.

## TEMA 6 – NODE.JS

### 1. INTRODUCCIÓN

Node.js es un runtime environment para JavaScript open-source y cross-platform. Incluye más de 1000000 de paquetes opensource.

Node.js emplea los mismos principios que JS:

- Un único proceso.
- Síncrono
- Paradigmas no bloqueantes → bloquear es la excepción.
- Gestión de eventos.

**Diferencias con JavaScript y programar en el navegador:**

- No se trabaja con el DOM, cookies, ...
- No están las variables del navegador como *document* o *window*.
- No tiene restricciones de acceso a ficheros.
- No tiene problemas de versiones del navegador (incompatibilidad con versiones antiguas de JS).

### 2. INSTALACIÓN

**Herramientas necesarias**

- Git Bash (sólo para Windows).
  - Linux y MacOS ya tienen consola.
- Editar de texto.
- Postman.

**Instalación**

Descargar e instalar la versión LTS de nodejs.org.

- Versión LTS (Long Term Support).

¿Qué significan los números de la versión X.Y.Z.?

- Estándar de numeración de versiones para software: **Semantic Versioning (SemVer)**.
- **MAJOR.MINOR.PATCH:**
  - **MAJOR:** cambios incompatibles
  - **MINOR:** Se añade funcionalidad retrocompatible.
  - **PATCH:** corrección de errores retrocompatibles.

¿Qué es LTS?

LTS significa **Long Term Support**, son versiones que tienen soporte durante mucho tiempo. Esto aplica a muchos programas de software.

**Nvm**

**nvm** (Node Version Manager) permite trabajar con diferentes versiones de Node.js.

- **nvm list:** lista las versiones de nvm instaladas.
- **nvm use x.y.z:** activa la versión x.y.z de Node.js.
- **nvm install x.y.z:** instala la versión x.y.z de Node.js
- **nvm --help:** muestra la ayuda.

**Npm**

**npm** (Node Package Manager) es un gestor de paquetes de Node.js, incluido en la instalación. También se usa para JavaScript en frontend.

Gestiona las dependencias de un proyecto (descarga, actualización, versiones).

Permite definir y ejecutar tareas (testing, etc).

Comandos:

- **npm init**: Crear el archivo de configuración package.json.
- **npm install** / **npm i**: instalación de las dependencias. Requiere el archivo de configuración package.json.
- **npm install nombre\_paquete**: instalación de un paquete.
- **npm install <paquete>@<versión>**: instalación de la versión x.y.z de un paquete.
- **npm list** / **npm ls**: listar los paquetes instalados junto con la versión.
- **npm update**: actualizar todas las dependencias.
- **npm update nombre\_paquete**: actualizar un paquete específico.
- **npm uninstall nombre\_paquete**: eliminar un paquete.
- **npm view nombre\_paquete**: ver información de un paquete.
- **npm view nombre\_paquete versions**: ver las versiones de un paquete.

## REPL

**REPL** (Read Evaluate Print Loop) es un entorno de Node.js en forma de consola. Se puede usar el tabulador para autocompletar. Para ejecutarlo, empleamos el comando **node**.

Comandos especiales:

- **.help**: muestra la ayuda.
- **.exit**: sale de REPL (equivalente a pulsar Ctrl-C dos veces).

## Npx

**npx** permite ejecutar código. No hace falta tener instalado el paquete.

## Ejemplo 0

Para importar módulos se emplea la sentencia **require**:

```
const http = require("http");
```

**http** es un módulo incluido en el core de Node.js para atender peticiones http. Otros módulos core incluyen:

- **url**: para trabajar y parsear URLs.
- **path**: para trabajar y parsear paths.
- **fs**: para trabajar con I/O en archivos.
- **util**: funcionalidades diversas.

**createServer** es un método del módulo **http** para crear un servidor HTTP. Toma como parámetro una función (**requestListener**) que se ejecutará con cada petición y toma como parámetros:

- **req**: objeto **http.IncomingMessage** con información de la petición.
- **res**: objeto **http.ServerResponse** con información de la respuesta.

**listen**(**[[port[, host[, backlog]]], callback]**); es un método que arranca el servidor escuchando en un puerto. Toma como parámetros:

- **port**: puerto donde queremos escuchar. Si se omite, se elegirá uno aleatorio.
- **host**: por defecto la **unespecified IPv6 address** (equivale a 0.0.0.0 en IPv4).
- **backlog**: es el número máximo de conexiones pendientes. Por defecto es 511.
- **callback**: se ejecuta cuando termina de arrancar el servidor.

## Variables de Entorno

Las variables de entorno son accesibles con el módulo **process** (disponible sin necesidad de require) mediante la sintaxis **process.env.nombre\_variable**.

Se pueden definir al ejecutar el programa:

USER\_ID=234567 USER\_KEY=foobar node app.js

Se suelen usar para determinar el puerto en el que arranca el servidor:

```
const port = process.env.PORT || 3000;
```

**NODE\_ENV** se usa para definir variables si se está en un entorno de producción o de desarrollo. Es empleado por muchos módulos. Puede adoptar dos valores: **production** y **development**.

Las variables de entorno también se pueden **cargar** en un **archivo .env** con el módulo **dotenv**:

```
require("dotenv").config();
```

## Argumentos

Se pueden añadir al ejecutar el programa:

```
node index.js joe smith
```

```
node index.js name=joe surname=smith
```

Son accesibles con **process.argv**:

- Devuelve un array
- Posición 0 → full path del comando de node
- Posición 1 → full path del archivo.
- Posiciones 2 en adelante → los argumentos.

Si queremos usar clave=valor hay que parsearlo. Podemos emplear el módulo **minimist** para tratar los argumentos:

- Los argumentos se pasan como --nombre=valor.
- También se pueden usar como en Unix con -opción valor, donde opción es una letra.

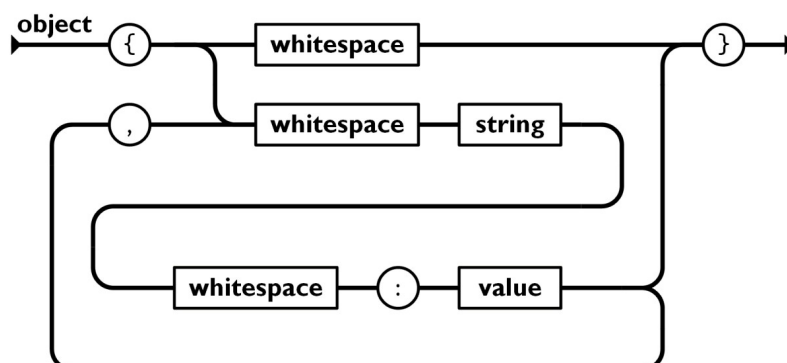
### 3. JSON

**JSON** (JavaScript Object Notation) es un formato ligero para el intercambio de datos. Soporta objetos, arrays y valores. No permite comentarios.

## Sintaxis

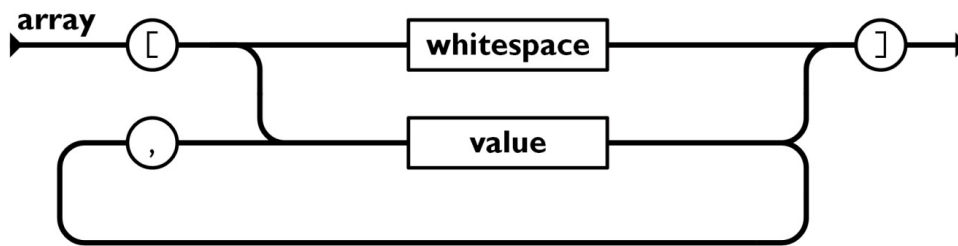
**Objeto:**

- set de pares “clave”: valor.
- Rodeado por llaves.
- La clave es un String y tiene que tener comillas dobles.
- La clave debería usar nomenclatura lower camel case.
- Los datos se separan con comas (cuidado con las trailing commas).
- Puede estar vacío: { }.

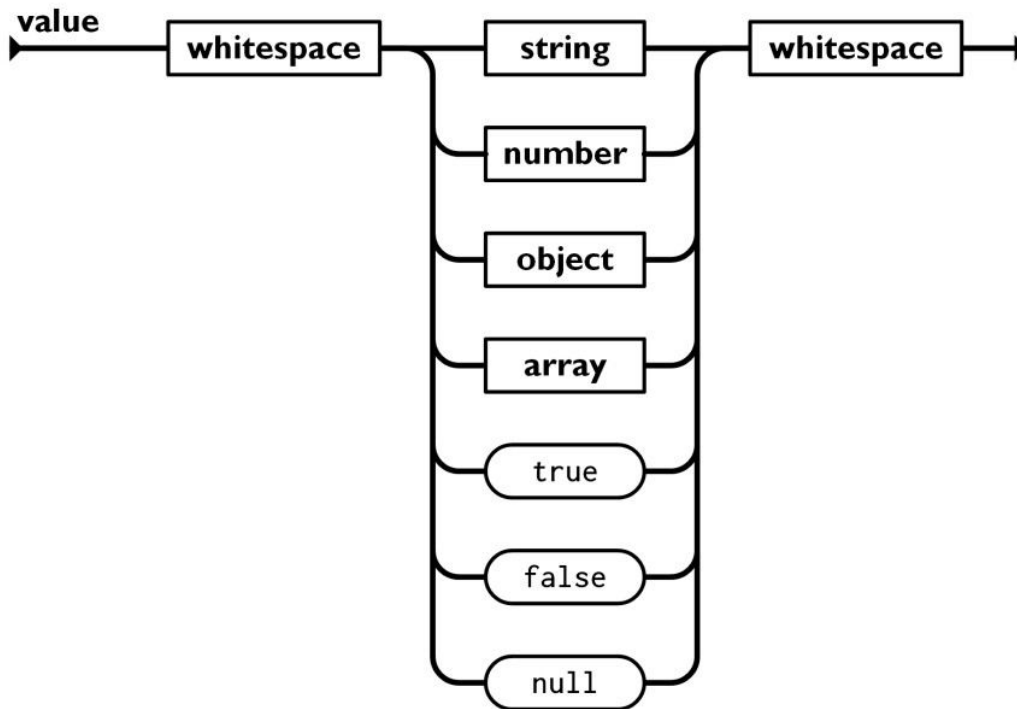


## Array

- Colección ordenada de valores.
- Rodeado por corchetes [ y ].



Valor:



Se pueden anidar elementos, y debe haber **un único elemento raíz** que tiene que ser un **valor**:

[ todo lo demás ] o { todo lo demás }

### Array vs Objeto

No se garantiza el orden del contenido de los objetos, mientras que los arrays sí que están ordenados.

### JSON con JS

Se utiliza el paquete **JSON**, que no es específico de Node.js.

Ejemplos:

- Convertir un objeto JSON a texto: `let text = JSON.stringify(obj);`
- Convertir un texto en formato JSON a objeto: `let obj = JSON.parse(text);`
- Recorrer un objeto JSON: `for (let key in jsonObj) { ... }`

### Archivo package.json

Archivo en formato JSON usado por npm. Contiene los metadatos del proyecto (dependencias, descripción, versión, licencia, configuración, ...).

Se puede crear manualmente o usando **npm init**.

- **"name": "nombre-proyecto"** → nombre del paquete, hasta 214 caracteres. Solo minúsculas, '-' o '\_'.
- **"version": "x.y.z"** → versión del paquete, en formato semver.

- **“description”**: **“descripción”** → descripción del paquete.
- **“main”**: **“src/main.js”** → punto de entrada del paquete, donde se buscará para exportarlo como módulo. Si no existe, el valor por defecto es **index.js**. Se ejecuta con **node**.
- **“scripts”**: **{ “nombre”: “comando”, ... }** → define scripts ejecutables a través de la sentencia **node run nombre**. Algunos especiales (**start** y **test**) se pueden ejecutar sin **run**.
- **“keywords”**: **[ “keyword\_1”, ... ]** → array de palabras clave, útil para buscar el paquete en npm.
- **“author”**: **“Nombre <email> (web)”** o **“autor”**: **{ “name”: “nombre”, “email”: “email”, “url”: “...” }** → solo puede haber uno. Email y url son opcionales.
- **“contributors”** **[ ... ]** → puede haber varios. El formato es el mismo que para el autor.
- **“dependencies”**: **{ “paquete”: “^x.y.z” }** → dependencias del paquete. Se añaden automáticamente al hacer **npm install**. Siguen un formato especial para indicar cómo se actualiza.
- **“devDependencies”**: **{ ... }** → dependencias sólo para desarrollo. No son necesarias en producción. Se añaden automáticamente al hacer **npm install --save-dev**.

Existen muchas opciones más (bugs, homepage, license, repository, private, ...).

Hay un problema con **package.json**: el proyecto no es reproducible al 100%. Ejemplo:

```
“dependencies”: {
 “vue”: “^2.5.2”
}
```

Alguien podría usar la versión 2.5.2, otro la 2.5.3 y otro la 2.6.0, lo que puede dar lugar a inconsistencias en el proyecto.

Una alternativa es subir la carpeta **node\_modules** al repositorio, pero no es recomendable porque puede llegar a ocupar mucho (varios cientos de megas).

### **package-lock.json**

Especifica la versión exacta de cada dependencia. El paquete es 100% reproducible. Este fichero se genera y actualiza de forma automática:

- Al hacer **npm install**
- Al hacer **npm update**

## **3. EVENT LOOP**

El **event loop** es un único thread que se repite (loop). Cada iteración se denomina **tick**, que gestiona la ejecución de los eventos mediante una cola FIFO de **callbacks** pendientes, en cada tick se ejecutan todos los callbacks pendientes hasta que se vacía la cola.

Este proceso se gestiona internamente por la librería **libuv**. En realidad, hay varias fases y cada una tiene su propia cola FIFO.

### **Ejemplo**

# Event Loop

```
const bar = () => console.log('bar');
const baz = () => console.log('baz');
const foo = () => {
 console.log('foo');
 bar();
 baz();
}
foo();
```



## No se debe BLOQUEAR el EVENT LOOP

Nunca se debe bloquear el event loop:

- Cada tick debe ser corto.
- El trabajo asociado a cada cliente tiene que ser breve.
- Se debe intentar dividir las tareas más intensivas.

## 4. PETICIONES HTTP

Los paquetes **https** y **http** permiten hacer peticiones HTTP (GET, POST, PUT, DELETE...).

### Petición GET

**const req = https.request( options [, callback]);**

- **options**: string y objeto con las opciones
- **callback**: se ejecuta cuando se recibe la respuesta.
- Para el método **GET** se puede usar **https.get(options,callback)** en lugar de especificar el método en las options.

### Gestión de Eventos - Errores

**req.on("error", callback)**

El método **on** define un callback para un evento. Si no se especifica el evento **error**, podría saltar una excepción y detener el programa.

### Petición POST

Es equivalente al GET, pero se deben añadir **cabeceras** y especificar el **mensaje**. PUT/DELETE son equivalentes a POST.

El evento **data** [res.on('data', d => {});] permite procesar la respuesta que se recibe en binario.

## 6. FICHEROS

Para trabajar con ficheros se usa el paquete **fs** (**filesystem**), que permite el acceso e interacción con el sistema de ficheros. Forma parte del core de Node.js.

Los métodos son **async** por defecto, aunque hay una versión síncrona añadiendo **Sync** al final del método:

fs.access() ↔ fs.accessSync()



Hay múltiples métodos:

- `fs.access()`: comprueba si un fichero existe y Node.js puede acceder a él con sus permisos.
- `fs.appendFile()`: añade datos a un fichero. Si el fichero no existe, se crea.
- `fs.close()`: cierra un descriptor de fichero.
- `fs.mkdir()`: crea un directorio.
- `fs.open()`: abre un fichero en el modo seleccionado.
- `fs.readdir()`: lista los contenidos de un directorio.
- `fs.readFile()`: lee el contenido de un fichero.
- `fs.realpath()`: convierte rutas relativas (., .., etc) en rutas absolutas.
- `fs.rename()`: renombra un fichero o directorio.
- `fs.rmdir()`: elimina un directorio.
- `fs.stat()`: devuelve el estado del fichero identificado por el nombre de fichero pasado.

**`fs.open()`**

**`fs.open(path [, flags[, mode]], callback);`**

flags:

- `r`: modo lectura (por defecto).
- `r+`: modo lectura y escritura. No creará el archivo si no existe.
- `w+`: modo lectura y escritura (al principio del fichero). Se crea el archivo si no existe.
- `a`: escritura al final del fichero. Se crea el archivo si no existe.

## 7. EXPRESS

**Express** es un framework web que permite definir métodos HTTP. Permite definir las rutas y trabajar con middleware.

Es un framework “**unopinionated**”: es decir, no decide la forma en la que se debe trabajar con él, deja libertad al desarrollador:

- No define el template engine.
- No define la base de datos.

Instalación:

- `npm install express`

### Métodos HTTP

Para definir manejadores para los diferentes métodos HTTP, se debe crear un objeto express y emplear la sintaxis:

**`app = express();`**

**`app.METHOD(path, callback [, callback ...]);`**

- **METHOD**: get, post, delete, ...
- **path**: string, patrón o expresión regular que identifica el recurso solicitado por el cliente.
- **Callback**: una o más funciones a ejecutar cuando se reciba una petición del método asociado.

Para iniciar el servicio, se usa el método:

**`app.listen(port, callback);`**

### Middleware

El **middleware** es **código** que ejecutamos en medio de otra ejecución. Se definen como

funciones con tres parámetros: req, res y next. Se necesita invocar el método **next()** para continuar la cadena.

Para usar una función definida como middleware, lo asociamos a la aplicación mediante el método **use**.

Ejemplo:

```
let myLogger = function (req, res, next) { // Definimos el middleware
 ...
 next();
}
app.use(myLogger); // Indicamos que se debe usar
```

## **Templates**

Los **template engines** son plantillas estáticas para generar las vistas. Facilitan el diseño de una página HTML.

En tiempo de ejecución:

- Se reemplazan las variables por sus valores.
- Se transforma a un archivo HTML que se le envía al cliente.

Express soporta varios, además de la creación de nuestros propios templates.

## **Pug**

Antiguamente conocido como Jade, Pug es un **template basado en Haml** (HTML abstraction markup language). Ejemplo:

```
html
 head
 title= title
 body
 h1= message
```

## **EJS – Embedded JavaScript templates**

Es más recomendable, ya que usa HTML. Usa etiquetas extras para el flujo de control y variables:

**<% ... %>**: acciones de control de **flujo** (for, if, etc).

**<%= ... %>**: representa el contenido de la **variable**

Ejemplo:

```
// index.ejs
<!DOCTYPE html>
<html>
 <head>
 <title><%= title %></title>
 </head>
 <body>
 <% var fruits = ["Apple", "Pear"] %>
 <h1><%= message %></h1>
 <% for(let i = 0; i < fruits.length; i++) {%>
 - Fruta: <%= fruits[i] %>
 <% } %>
 </body>
```

</html>

## **express-generator**

Express generator es una herramienta que permite definir la estructura del proyecto Node para emplear express.

Ejecutar express-generator:

```
npx express-generator -v ejs express_test
```

alternativamente, se puede instalarlo en global, lo que nos permitirá ejecutar el comando directamente desde la consola:

```
npm install -g express-generator
```

```
express -v ejs express_test # Nombre del comando: express
```

Luego:

```
cd express_text
```

```
npm install
```

```
DEBUG=express-test:* npm start
```

Este comando creará la siguiente estructura de ficheros:

```
.
|-- app.js # Configuración de express
|-- bin
| `-- www # Punto de arranque del programa (js)
|-- package.json
|-- public # Archivos estáticos: Imágenes, JS y CSS
| |-- images
| |-- javascripts
| `-- stylesheets
| `-- style.css
|-- routes # Ficheros para gestionar cada ruta
| |-- index.js # Gestiona las peticiones a "/"
| `-- users.js # Gestiona peticiones a "/users"
`-- views # Templates (formato ejs)
|-- error.ejs # Errores
`-- index.ejs # Página principal
```

## **public**

Se usa para añadir archivos estáticos. Se podría añadir archivos html. Se enlaza en **app.js** con:

```
app.use(express.static(path.join(__dirname, "public")));
```

## **Carpeta routes**

Define las rutas, es decir, cómo responde la aplicación a un endpoint (URI + Método HTTP). Para enlazarlo en **app.js**:

```
var indexRouter = require('./routes/index');
```

```
var usersRouter = require('./routes/users');
```

```
//...
```

```
app.use('/', indexRouter);
```

```
app.use('/users', usersRouter);
```

## **Carpeta views**

Define los templates. Se enlazan en **app.js** de la siguiente manera:

```
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs'); // ejs, pug, ...
```

### **Información de Sesión**

Para guardar información de la sesión, es necesario utilizar el **módulo express-session**. Será accesible a través del atributo **session** de la petición:

**req.session**.<nombre de la variable que queramos>

Para usarlo (en app.js):

```
const session = app.require('express-session');
app.use(session({
 resave: false,
 secret: "clave secreta para chats",
 saveUninitialized: false
}));
```

Otras formas de guardar información:

- **res.locals**: objeto con **variables locales para la petición**. Es accesible directamente en las vistas, pero sólo está disponible durante la petición, se borra cada vez. Para guardar información se puede combinar con **req.session**.
- **app.locals**: objeto con **variables locales de la aplicación**. Está disponible durante toda la aplicación (no sólo durante una petición).

## **8. SOCKET.IO**

Socket.io es un **módulo de node.js** que permite comunicación bidireccional cliente – servidor a través de WebSockets.

Incorpora reconexión en caso de fallo. Es escalable.

Se utiliza para enviar y recibir eventos.

**Instalación:**

```
npm install socket.io
```

### **Ejemplo**

```
//servidor
```

```
const express = require("express");
const { createServer } = require("http");
const { Server } = require("socket.io");
const app = express();
const httpServer = createServer(app);
const io = new Server(httpServer, { /* options */ });
//...
io.on('connection', (socket) => {
 console.log('a user connected');
 socket.on('disconnect', () => {
 console.log('user disconnected');
 });
});
httpServer.listen(3000);
```

### **Resumen**

**socket.emit**("nombre evento", mensaje): enviar un mensaje.

**io.emit**("evento", mensaje): enviar un mensaje para todos los usuarios (**broadcast**).

**socket.broadcast.emit**(\* ... \*) : enviar un mensaje a todos menos a uno mismo.

### Nombres de eventos reservados

- connect
- connect\_error
- disconnect
- disconnecting
- newListener
- removeListener

## **9. LOGGING**

### **¿Qué es hacer logging?**

Hacer **logging** es guardar información relevante de nuestro programa para su posterior análisis.

#### **Objetivos:**

- Comprobar que el programa funciona correctamente
- Solucionar bugs
- Análisis de parámetros

### **¿Qué información queremos loggear?**

Cambios en el estado del programa, interacción con el usuario, interacción con otros programas, interacción con ficheros, comunicaciones, cada vez que se entra en un método, cada vez que se sale de un método, errores, excepciones.

### **¿Qué información NO queremos loggear?**

Información sensible (información personal, información médica, información financiera, contraseñas, direcciones IP, ...).

Se debe tener cuidado con las URL porque podrían contener información sensible pasada como parámetro (/user/<email>, ...).

Se debe tener en consideración el tamaño del log.

### **¿Por qué no usar console.log() ?**

- No se puede desactivar, tendríamos que borrar todas las líneas → el **logging** se puede activar/desactivar.
- No es granular, se imprime todo → en **logging** se pueden usar distintos niveles y solo imprimir los que nos convenga.
- Se mezcla con otra información en la consola → **Logging** se distingue en la consola.
- No es persistente → Con **logging** podemos usar archivos u otros soportes.
- No hay más información que la que añadimos nosotros → Al **logging** se pueden añadir metadatos.

### **Niveles**

- **Fatal**: situación catastrófica de la que no nos podemos recuperar.
- **Error**: error en el sistema que detiene una operación, pero no todo el sistema.
- **Warn**: condiciones que no son deseables pero no son necesariamente errores.
- **Info**: mensaje informativo.
- **Debug**: información de diagnóstico.
- **Trace**: todos los detalles posibles del comportamiento de la aplicación.

## Console

Es una herramienta simple, similar a la versión de navegador. Se trata de un objeto global con varias versiones:

- `console.error()`
- `console.warn()`
- `console.log()`
- `console.debug()`

## Logging

Conviene usar una librería para facilitar el logging:

- Ligera
- Que permita dar formato
- Que pueda distribuir los logs (terminal, fichero, base de datos, HTTP, ...).

Librerías de logging en Node.js:

### Logging – Morgan

**Morgan** es una librería usada por expressjs para logging automático (middleware):

```
const logger = require('morgan');
if (app.get('env') === 'production') {
 app.use(logger('common', {
 skip: function(req, res) {return res.statusCode < 400},
 stream: __dirname + '/../morgan.log'
 }));
} else {
 app.use(logger('dev'));
}
```

### Logging – Winston

**Winston** es una librería simple con soporte para múltiples transportes que desacopla las partes del proceso de logging.

Proporciona flexibilidad en el formato (json, xml, ...). Permite definir nuestros propios niveles.

Nota: **transporte**  $\equiv$  lugar donde se guarda la información

#### Ejemplo de configuración:

```
const winston = require('winston');
const logger = winston.createLogger({
 level: 'info',
 format: winston.format.json(),
 defaultMeta: { service: 'user-service' },
 transports: [
 new winston.transports.File({ filename: 'error.log', level: 'error' }),
 new winston.transports.File({ filename: 'combined.log' })],
});
if (process.env.NODE_ENV !== 'production') {
 logger.add(new winston.transports.Console({
 format: winston.format.simple(),
 }));
}
```

```
}
```

### Ejemplo de uso

```
logger.log({
 level: 'info',
 message: 'Hello distributed log files!'
});
logger.log('info', 'Mensaje de info');
logger.info('Otro mensaje de info');
logger.warn("Mensaje de aviso");
logger.error("Mensaje de error");
```

## 10. BASES DE DATOS

Node.js puede trabajar con cualquier base de datos:

- **SQL** (MySQL, Oracle, SQLite, PostgreSQL ...)
- **No SQL** (MongoDB, Cassandra, Redis)

Funciona mejor con bases de datos NoSQL.

### Bases de Datos

Node.js proporciona dos librerías MySQL muy parecidas:

- **mysql**
- **mysql2**

### Uso:

Require:

```
//npm install mysql2
const mysql = require('mysql2');
```

Configuración:

```
const con = mysql.createConnection({/*Opciones*/})
```

Conexión inicial:

```
con.connect(function(err) {/*...*/});
```

Query:

```
con.query("Query", function(err) {/*...*/});
```

Prepared Statements:

```
con.execute(
 'SELECT * FROM `table` WHERE `name` = ? AND `age` > ?',
 ['Rick C-137', 53],
 function(err, results, fields) {/*...*/}
);
```

Cerrar la conexión:

```
con.end();
```

## 11. PASSPORT

Passport es un middleware de autenticación fácilmente integrable con Express.

Soporta muchas estrategias:

- **username/password**
- **Oauth** (Facebook, Twitter, Google, ...)
- **OpenID**

- ...

## 12. APM

**APM** (Application Performance Monitoring) son interfaces para la gestión de:

- Rendimiento, Disponibilidad, Logs, Tráfico, Uso de Recursos, Tasa de Errores, Latencia, etc.

Hay varias opciones con Node.js (app metrics, retrace, PM2, Clinic.js, Express-status-monitor...).

## 13. TESTING

El testing permite buscar bugs y evitar futuros errores, de ejecución automática.

### Testing – Mocha

**Mocha** es una librería para el testing.

#### Instalación

```
npm install -g mocha
```

Añadir a package.json:

```
{
 "scripts": {
 "test": "mocha"
 }
}
```

Crear una carpeta en el directorio raíz.

**Escribir tests requiere** de una **assertion library**, por ejemplo, **Chai**

```
npm install chai
```

Hay múltiples formas de aserción:

- **Test Driven Development (TDD):**  
assert.equal(3,3)
- **Behavior Driven Development (BDD):**  
expect(3).to.equal(3)

### Testing – Continuous Integration

Cada merge del código en la rama principal implica:

- **Code Build:**
  - Se descargan las dependencias
  - Se instalan las herramientas
  - Se compila el código
  - Se hace **linting** (se verifican los errores de estilo)
  - Se genera la versión final.
- **Test:** se ejecutan
  - Unit tests
  - Integration tests
  - End-to-End tests
  - UI tests



# DISEÑO Y ACCESIBILIDAD

## 1. INTERFACES DE USUARIO

## 2. DISEÑO CENTRADO EN HUMANOS (HUMAN CENTERED DESIGN)

## 3. ACCESIBILIDAD WEB

### Guías:

- Proporcionar texto alternativo
- Proporcionar encabezados
- Subtitular y/o proporcionar transcripciones
- Permitir saltar elementos repetitivos
- No transmitir información únicamente con el color
- Asegurar que el texto es claro y fácil de leer
- Garantizar la accesibilidad del contenido (PDF, Word, etc).

El usuario más importante es ciego. La mitad de las visitas vienen de Google, y Google sólo ve lo que un ciego puede ver. Si tu sitio no es accesible, tendrás menos visitas.

### De cara al diseño:

- Haz accesible el JavaScript
- Diseña siguiendo las normas HTML y CSS
- Crea contenido adaptable que se pueda presentar de distintas formas
- Ten en cuenta la navegación por teclado

### WCAG - Web Content Accessibility Guidelines

Desarrollado por el W3C, es un estándar para la accesibilidad web. Basado en los siguientes 4 principios: perceptible, operable, entendible y robusto.

Define 13 guías para que el contenido sea más accesible: *seizure safety*, *compatible*, *navigation*, *input assistance*, *readable*, *predictable*, *video alternatives*, *text alternatives*, *adaptable*, *keyboard accessible*, *clarity*, *time*.

### ARIA – Accessible Rich Internet Applications

ARIA es una especificación del W3C que indica una serie de **roles** y **atributos** para hacer la web más accesible.

Si se puede, es mejor usar etiquetas HTML nativas con el mismo significado.