

Almacenamiento de datos en el lado cliente e integración avanzada de componentes



Ariana Nataly Mamani Ticona
Henry David Orbe Cisneros
Anabel Pedrajas Navarro
Román Kornyejev
Javier Lasso Pons

Almacenamiento de datos en entorno cliente	4
Cookies	5
Ejemplo práctico	5
Local Storage	9
Funciones	10
Ejemplos	10
Session Storage	11
Funciones	11
Ejemplo básico de Session Storage en Javascript	12
Ejemplo JSON de Session Storage en Javascript	12
Ejemplo desarrollado de Session Storage en Javascript	13
IndexedDB	14
Primeros pasos con IndexedDB	15
Abrir una base de datos:	15
Almacenar datos:	16
Transacciones:	17
Búsquedas:	18
Web SQL Database	19
Integración avanzada de componentes	20
Componentes de interfaz de usuario (UI)	21
Control con inputs	21
Componentes de navegación	22
Componentes de información	22
Contenedores	23
Componentes de automatización	25
Jest	25
Mocha	26
Cypress	26
Karma	27
Angular	27
Webdriver IO	28
Componentes de seguridad	29
A nivel general	29
Comprobación de integridad. Atributo Integrity	30
Test frecuentes para detectar vulnerabilidades en NPM	31
Permitir las actualizaciones para parches y versiones menores	31
Validaciones para evitar inyecciones	32
Uso del modo Estricto	32
Herramientas de análisis del código	32
Control del envío de código y del código enviado	33
Frameworks de desarrollo	35
Frameworks back end	36
Frameworks front end	37

Componentes de terceros	38
Apis	39
SDKs	39
ıBibliografía	40

Almacenamiento de datos en entorno cliente

El almacenamiento de datos en el lado del cliente se refiere a la capacidad de almacenar datos en un dispositivo del cliente (como un navegador web o una aplicación móvil) en lugar de en un servidor. Esto permite que los datos estén disponibles incluso cuando el dispositivo no está conectado a Internet, y puede ayudar a mejorar la velocidad y la funcionalidad de la aplicación.

Hay varias tecnologías que se utilizan para el almacenamiento de datos en el lado del cliente en un entorno web, incluyendo:

- **Cookies:** Son pequeñas cantidades de datos que se almacenan en el dispositivo del cliente y que se envían al servidor junto con cada solicitud. Se pueden utilizar para almacenar información como preferencias de usuario, información de sesión, y datos de autenticación.
- **Local Storage:** Es una tecnología del navegador que permite almacenar datos en el lado del cliente en un formato de clave-valor. Los datos almacenados en Local Storage son permanentes y permanecerán en el dispositivo del cliente hasta que se eliminen explícitamente.
- **Session Storage:** Similar a Local Storage, pero los datos almacenados en Session Storage sólo permanecerán en el dispositivo del cliente hasta que se cierre el navegador.
- **IndexedDB:** Es una base de datos en el lado del cliente que permite almacenar grandes cantidades de datos en el formato de clave-valor, similar a una base de datos relacional, pero con una interfaz de programación de aplicaciones (API) JavaScript.
- **Web SQL Database:** Es otra tecnología de base de datos en el lado del cliente basada en SQL (Lenguaje de Consulta Estructurada).

Estas tecnologías son útiles para almacenar información como preferencias de usuario, información de sesión y datos de autenticación, al mismo tiempo que también pueden ayudar a mejorar la velocidad y la funcionalidad de la aplicación, ya que los datos se pueden acceder localmente en lugar de tener que ser recuperados cada vez desde un servidor remoto. Sin embargo, es importante tener en cuenta que el almacenamiento de datos en el lado del cliente también presenta algunos riesgos de seguridad, ya que los datos pueden ser fácilmente accedidos o modificados por atacantes maliciosos o por software malicioso en el dispositivo del cliente.

Cookies



Son ficheros de información almacenados en el ordenador del usuario, que nos sirven para recopilar y almacenar información del usuario, para así poder realizar un seguimiento de su actividad en nuestro sitio web y personalizar su experiencia de navegación. Algunos de los ejemplos de uso más conocidos que hemos visto tanto en entorno cliente como en entorno servidor permiten al usuario:

- Mantener la sesión iniciada
- Mantener preferencias de cookies (genius)
- Mantener preferencia de modo claro/oscuro
- Mantener preferencias/configuraciones del sitio web en general.

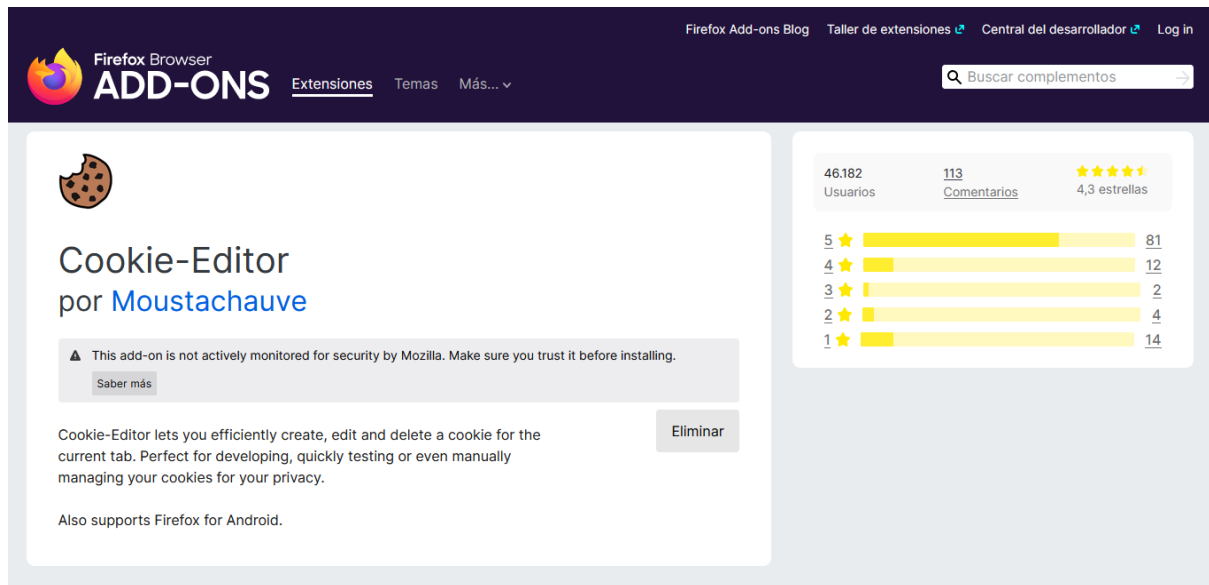
Las cookies son enviadas al servidor (a nuestra aplicación web) cada vez que el usuario entra en ella, lo que permite que el sitio web reconozca al usuario y sus preferencias. Es importante destacar que las cookies sólo pueden ser leídas por el sitio web que las creó, y no pueden ser utilizadas para acceder a otros archivos en el dispositivo del usuario. Cabe también a destacar que las cookies pueden tener una fecha de expiración.

Antes no eran muy relevantes, pero con el tiempo se volvieron cada vez más y más populares. A medida que se volvían más populares, también surgieron preocupaciones sobre la privacidad. Como resultado, los navegadores web ahora permiten a los usuarios controlar su uso de las cookies, cómo bloquearlas, eliminarlas o aceptar solo las necesarias para que el sitio funcione.

Ejemplo práctico

Nos vamos a adentrar un poco más en el apartado técnico y vamos a ver cómo crear, manipular y leer cookies. Pero antes de nada, nos descargaremos una extensión para

Firefox que nos permita ver y manipular las cookies de forma sencilla (es la que tenemos de entorno servidor):



Bien, una vez hecho esto, vamos al lío. Crearemos 2 funciones, una para crear cookies y otra para recuperar el valor de las cookies por su nombre.

Función para crear cookies:

```
function setCookie(cname, cvalue, exdays) {
    var d = new Date();
    d.setTime(d.getTime() + (exdays * 24 * 60 * 60 * 1000));
    var expires = "expires=" + d.toUTCString();
    document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
```

Esta función acepta tres argumentos: el nombre de la cookie (cname), el valor de la cookie (cvalue) y el número de días que debe durar la cookie (exdays). La función crea una fecha de caducidad para la cookie y la almacena en el documento utilizando el método **document.cookie**.

Función para buscar y leer una cookie:

```
81 function getCookie(cname) {
82     var name = cname + "=";
83     var ca = document.cookie.split(';');
84     for(var i = 0; i < ca.length; i++) {
85         var c = ca[i];
86         while (c.charAt(0) == ' ') {
87             c = c.substring(1);
88         }
89         if (c.indexOf(name) == 0) {
90             return c.substring(name.length, c.length);
91         }
92     }
93     return "";
94 }
```

Esta función acepta un argumento: el nombre de la cookie (cname). La función divide el contenido de **document.cookie** en un array usando **split** y busca el valor de la cookie especificada. Si se encuentra la cookie, la función devuelve su valor, en caso contrario devuelve una cadena vacía.

Bien, habiendo hecho estas funciones y escribiendo una pequeña estructura de formulario en html+css+js, podremos establecer las cookies que queramos y buscar las que queramos de forma cómoda:

Nombre cookie

Valor

Días para expirar

Crear cookie

Cookie creada!

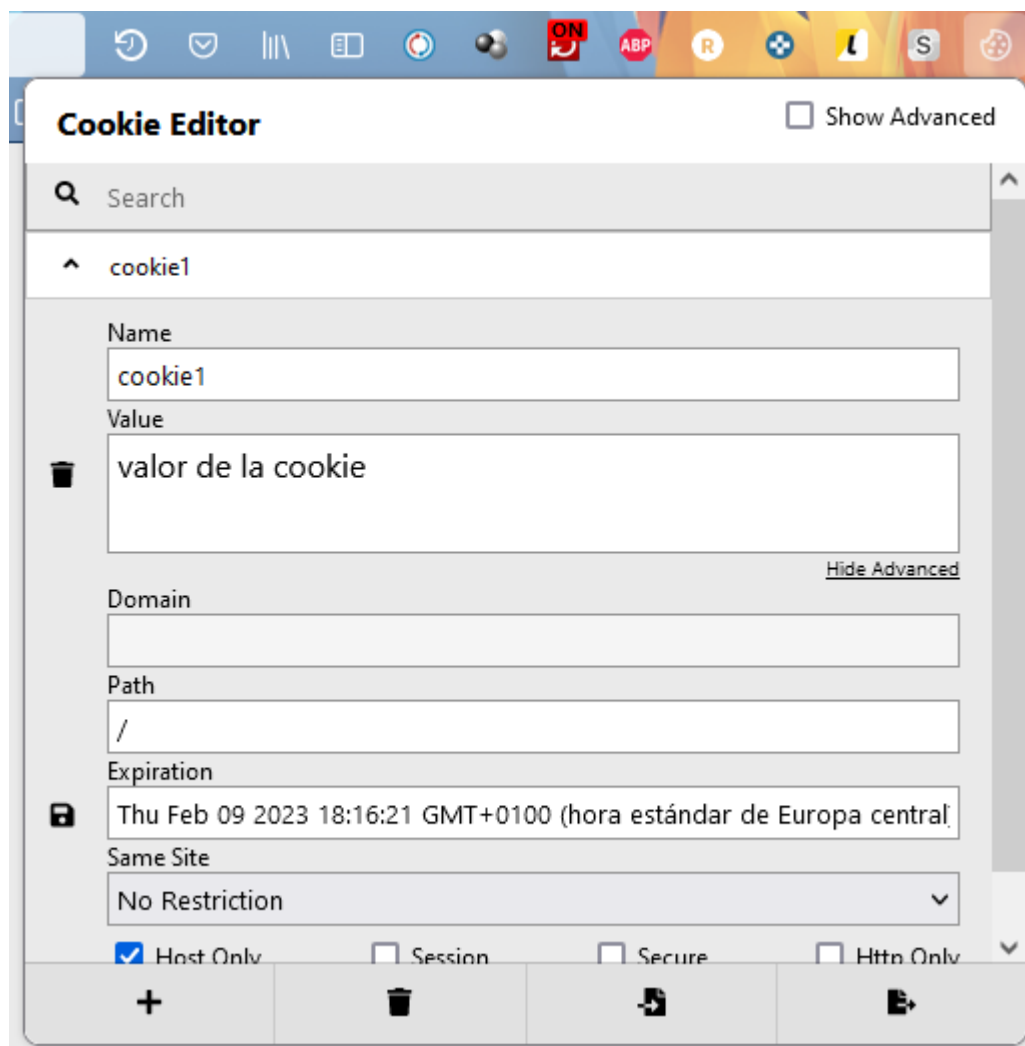
Nombre cookie

Buscar cookie

La cookie **cookie1** tiene el valor: **valor de la cookie**

**archivo: cookies.html*

Podemos también comprobar todas las cookies que tengamos de forma cómoda y con un par de clicks con el editor de cookies que he mencionado antes, así como ver su nombre, valor y fecha de expiración:



Local Storage



El Local Storage es una característica de HTML5 y una forma más avanzada de almacenamiento en el lado del cliente que permite almacenar grandes cantidades de datos de manera persistente (sin una fecha de caducidad). A diferencia de las cookies, los datos guardados en el Local Storage no se envían automáticamente al servidor con cada solicitud, lo que lo hace más eficiente en términos de rendimiento.

Es similar a las cookies, pero tiene algunas ventajas adicionales, como un mayor límite de almacenamiento (normalmente entre 5 y 10MB) y una forma más fácil de acceder y manipular los datos almacenados.

El Local Storage es útil en muchas situaciones donde se requiere que la información se mantenga disponible después de que el usuario haya cerrado el navegador o la pestaña. Algunos ejemplos comunes de uso de Local Storage son:

- Almacenamiento de preferencias del usuario, como la configuración de un tema o la elección de un idioma.
- Almacenamiento de información temporal, como el contenido de un carrito de compras.

Puede utilizarse desde javascript, permitiendo almacenar y recuperar datos en formato JSON. Los datos almacenados en Local Storage, al igual que las cookies, son accesibles sólo desde el dominio que los ha almacenado, esto significa que una página web no puede acceder a los datos almacenados por otra página web.

En resumen, las cookies son una forma de almacenar pequeñas cantidades de información en el navegador del usuario y se envían al servidor con cada solicitud. El Local Storage es una forma más avanzada de almacenamiento en el lado del cliente que permite almacenar grandes cantidades de datos sin una fecha de caducidad y que no se envían al servidor automáticamente. Ambas tecnologías son útiles para mantener el estado de la aplicación y para almacenar información del usuario.

Funciones

```
// Guardar un valor en el Local Storage
localStorage.setItem("llave", "valor");

// Obtener un valor del Local Storage
let valor = localStorage.getItem("llave");

// Eliminar un valor del Local Storage
localStorage.removeItem("llave");

// Eliminar todos los valores del Local Storage
localStorage.clear();
```

Ejemplos

```
// Almacenar una variable de sesión en LocalStorage
localStorage.setItem("session_id", "12345");

// Recuperar una variable de sesión de LocalStorage
let sessionId = localStorage.getItem("session_id");
console.log(sessionId); // Imprime "12345"
```

Session Storage

La SessionStorage es un tipo de almacenamiento en el navegador similar al LocalStorage, pero con algunas diferencias importantes. Al igual que el Local Storage, la Session Storage permite almacenar datos en el navegador del usuario, pero los datos almacenados en Session Storage solo están disponibles durante la sesión actual del usuario. Una sesión se inicia cuando el usuario abre el navegador y finaliza cuando el usuario cierra el navegador o se cierra la pestaña.

Se utiliza para almacenar información temporal que solo se necesita durante la sesión actual del usuario, como el estado de una aplicación web o el contenido de un formulario. Está limitado al uso dentro de la misma pestaña, no se puede compartir información entre ventanas o pestañas.

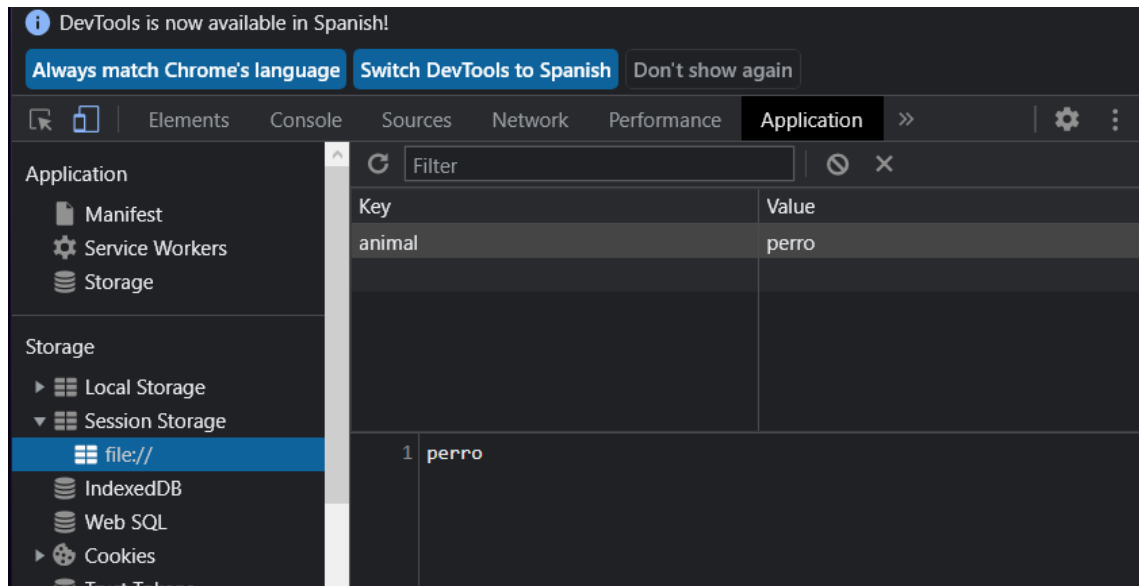
Desde Javascript, permite almacenar y recuperar datos en formato JSON. Al contrario que el Local Storage, los datos almacenados en Session Storage no persisten a través de múltiples sesiones, si el usuario cierra el navegador o se cierra la pestaña, los datos almacenados en Session Storage son eliminados.

Funciones

Iniciar sessionStorage	<code>sessionStorage.setItem("key", "value");</code>
Leer un ítem	<code>let data = sessionStorage.getItem("key");</code>
Eliminar un ítem	<code>sessionStorage.removeItem("key");</code>
Limpiar todos los ítems de Local Storage	<code>sessionStorage.clear();</code>

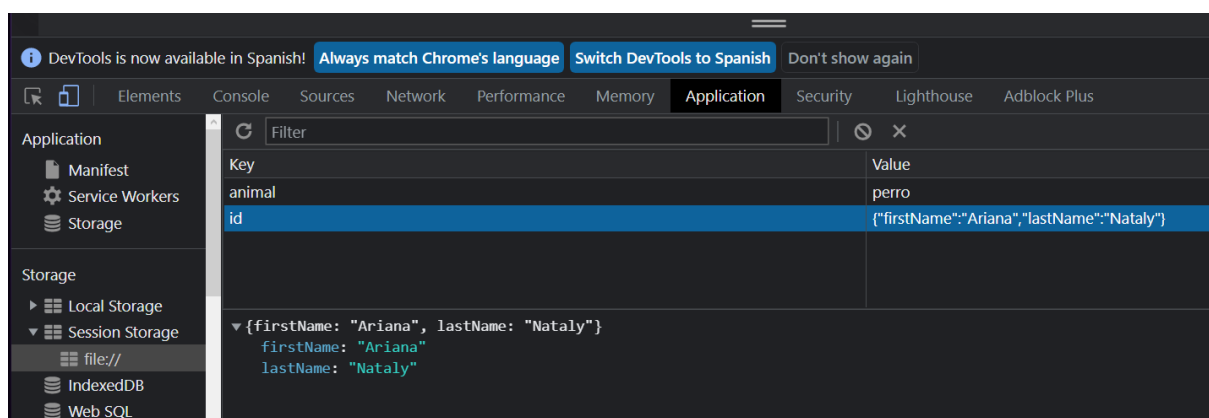
Ejemplo básico de Session Storage en Javascript

```
<script>
  sessionStorage.setItem("animal","perro");
  var animalito = sessionStorage.getItem("animal");
</script>
```



Ejemplo JSON de Session Storage en Javascript

```
var user = {
  firstName : "Ariana",
  lastName : "Nataly"
}
sessionStorage.setItem("id", JSON.stringify(user));
```



Ejemplo desarrollado de Session Storage en Javascript

```
<div class="container">  
  <a id="cambio" class="btn"></a>  
</div>
```

```
const oscuro = 'Haz click en el texto para cambiar a oscuro';  
const luz = 'Haz click en el texto para cambiar a luminoso';  
const modo_oscuro = 'dark';  
const modo_claro = 'light';  
const modo_default = modo_oscuro;  
const btn = document.querySelector('#cambio');
```

```
inicio();  
//para dar nuevo valor a la sesion  
btn.addEventListener('click', function () {  
  let mode = sessionStorage.getItem('mode');  
  if (mode) {  
    //el nuevo modo: si es oscuro lo cambiamos a claro, sino a oscuro  
    let newMode = mode === modo_oscuro ? modo_claro : modo_oscuro;  
    cambiarModo(newMode);  
    sessionStorage.setItem('mode', newMode);  
  }  
});  
//el primer valor al entrar a la pagina  
function inicio() {  
  let modo = sessionStorage.getItem('mode');  
  if (!modo) {  
    modo = modo_default;  
    sessionStorage.setItem('mode', modo_default);  
  }  
  cambiarModo(modos);  
}  
  
function cambiarModo(mode) {  
  if (mode === modo_oscuro) {  
    //añadimos texto a la a  
    btn.textContent = luz;  
    //añadimos una clase "dark" si el modo es el oscuro sino lo eliminamos  
    document.body.classList.add(modos_oscuro);  
  } else if (mode === modo_claro) {  
    btn.textContent = oscuro;  
    document.body.classList.remove(modos_oscuro);  
  }  
}
```

IndexedDB

IndexedDB es una característica de HTML5 que permite almacenar y acceder a grandes cantidades de datos en el navegador del usuario de forma asíncrona, es decir, la tarea se ejecuta en segundo plano y una vez finalizado se presenta el resultado, de esta forma no se bloquea el navegador a la hora de acceder a la información. Un aspecto importante a destacar es que las aplicaciones que usan esta API pueden trabajar tanto en línea como fuera de línea y puede ser combinado con otras tecnologías.

IndexedDB es una base de datos de almacenamiento de objetos, lo que significa que los datos se almacenan en forma de objetos JavaScript, en lugar de simples pares clave-valor como en el caso de las cookies o el Local Storage. Los datos se almacenan bajo el directorio raíz del usuario junto con la configuración personal del navegador, sus extensiones, etc. Cada usuario dispondrá de un almacenamiento independiente.

Este sistema se utiliza para almacenar datos que necesitan ser accedidos y manipulados de forma eficiente, como grandes cantidades de datos de una aplicación, información de sesión, o datos que se utilizan para mantener el estado de una aplicación web. El tipo de dato a almacenar es complejo, siendo posible registrar archivos de audio y video.

IndexedDB se puede utilizar desde JavaScript y ofrece una interfaz similar a las bases de datos relacionales, permitiendo almacenar, actualizar, consultar y eliminar datos. Esta API permite además el uso de índices (claves que identificarán a cada objeto registrado de forma única) con el fin de mejorar el rendimiento y realizar consultas más rápidas y eficientes.

El soporte para IndexedDB se agregó a Firefox versión 4, Google Chrome versión 11 e Internet Explorer versión 10 mientras que Safari agregó soporte en la versión 8. Por ello, es importante tener en cuenta que no todos los navegadores soportan IndexedDB, por lo que en algunos casos puede ser necesario usar una biblioteca de terceros para asegurar la compatibilidad en todos los navegadores.



Primeros pasos con IndexedDB

Abrir una base de datos:

El primer paso para trabajar con IndexedDB es conectarnos o abrir una base de datos, para ello escribimos lo siguiente:

```
1
2  let openRequest = indexedDB.open('mibasededatos', 1);
3
```

Parámetros a introducir:

- En el primer parámetro escribimos el nombre de nuestra base de datos.
- El segundo es la versión la cual viene predeterminada en 1.

Esto nos devolverá un objeto que podrá contar con uno de los siguiente eventos:

- **Success:** la base de datos está lista. Hay un “objeto database” en openRequest.result que usaremos en los siguientes pasos.
- **Error:** la apertura de la base de datos falló.
- **Upgradeneeded:** la base de datos está lista pero la versión especificada es obsoleta. Esto ocurre cuando la versión que hemos establecido en la función anterior es menor que la estructura de la base de datos, en ese caso tendremos que comparar las versiones y realizar una actualización. También puede ocurrir este error cuando la base de datos no existe.

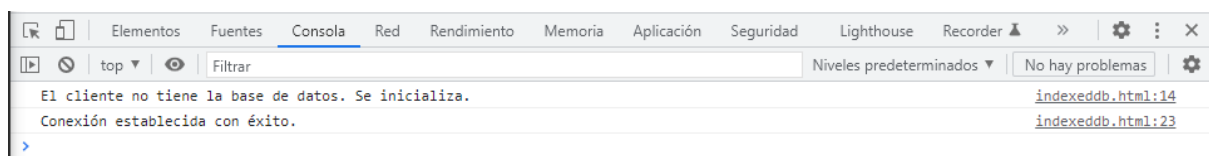
```
openRequest.onupgradeneeded = function() {
  console.log('El cliente no tiene la base de datos. Se inicializa.');
```

```
};

openRequest.onerror = function() {
  console.error("Error al crear la base de datos.", openRequest.error);
};

openRequest.onsuccess = function() {
  let objBases = openRequest.result;
  console.log('Conexión establecida con éxito.');
```

```
};
```



Como podemos ver, al ejecutar el script por primera vez y establecer la conexión salta el evento “onupgradeneeded” y se inicializa la base de datos, para después saltar al evento “onsuccess” indicándonos que todo salió bien.

Para eliminar una base de datos se escribe lo siguiente:

```
let deleteRequest = indexedDB.deleteDatabase('mibasededatos')
```

Almacenar datos:

IndexedDB usa almacenes de datos (object stores) en lugar de tablas, y una base de datos puede contener cualquier número de almacenes. Cuando un valor es almacenado, se le asocia con una clave. Existen diversas maneras en que una clave puede ser indicada dependiendo de si el almacén usa una ruta de clave (keyPath) o generador (autoIncrement).

Primero creamos el almacén de objetos con la siguiente sintaxis:

```
db.createObjectStore(  
  'playlist',  
  {  
    keyPath: 'id',  
    autoIncrement: true  
  }  
);
```

Parámetros a introducir:

- En el primer parámetro escribimos el **nombre de nuestro almacén de objetos**.
- El segundo un array llamado keyOptions que contendrá **propiedades para un futuro objeto a almacenar**. Las opciones son las siguientes:
 - **KeyPath**: la clave que usará IndexedDB.
 - **autoIncrement**: si es true, la clave para los futuros objetos que se almacenarán en la base se generarán de forma automática de manera autoincremental.

Si no establecemos el array keyOptions tendremos que proporcionar una clave de forma manual.

Por otro lado para borrar un almacén de datos se escribirá lo siguiente:

```
db.deleteObjectStore('playlist')
```


Transacciones:

Una transacción es un grupo de operaciones cuyos resultados se encuentran vinculados, es decir, que o todas las acciones deben ser exitosas o deben fallar. Esta seguridad es lo que garantiza una transacción. Para iniciar una transacción se establece la siguiente función:

```
let transaccion = db.transaction('playlist', 'readwrite');
```

Parámetros a introducir:

- En el primer parámetro escribimos el **nombre de nuestro almacén de objetos** al que la transacción va a acceder. Puede ser un array de nombres de almacenes si vamos a acceder a múltiples almacenes.
- El segundo llamado type, definirá el **tipo de transacción**. Las opciones son las siguientes:
 - **readonly**: solo de lectura (predeterminado). Si se presentan varias transacciones de este tipo sobre un mismo almacén, pueden suceder a la vez.
 - **readwrite**: lectura y escritura con la excepción de que no se puede crear, quitar o alterar los almacenes de objetos. Este tipo de transacción bloquea el almacén para escribir en él por lo que si se requieren realizar varias acciones en un mismo almacén deben ejecutarse una tras otra.
 - **versionchange**: puede hacer de todo pero no podemos crearla nosotros a mano. IndexedDB la crea automáticamente cuando abrimos la base de datos para poder disponer del evento upgradeneeded. Este es el único lugar en donde podemos modificar el almacén de objetos.

Tras crear la transacción podemos agregar un elemento a nuestro almacén:

```
//Obtenemos el almacen de objetos con el que queremos operar.
let playlist = transaccion.objectStore('playlist');

//Creamos una variable con los datos deseados.
let cancion = {
  titulo: 'Flowers',
  artista: 'Miley Cyrus',
  duracion: 3,
  creacion: new Date().getFullYear(),
  agregado: new Date().getTime()
}

//Creamos una petición que tendrá lugar al añadir el objeto creado
anteriormente en nuestro almacén.
let request = playlist.add(cancion);

//Si la petición fue exitosa se ejecuta el evento onsuccess
mostrandonos por consola la clave de nuestro nuevo objeto/elemento.
request.onsuccess = function() {
  console.log("Canción agregado a la playlist", request.result);
};

//Si la petición falló, se ejecuta el evento onerror mostrandonos
por consola ese error.
request.onerror = function() {
  console.log("Error", request.error);
};
```

Los almacenes de objetos permite dos métodos distintos para almacenar el valor:

- **put(value, [key]):** El primer valor es lo que se agrega al almacén. La clave key debe ser suministrada sólo si al almacén no se le asignó la opción keyPath o autoIncrement. Si ya hay un valor con la misma clave, este será reemplazado.
- **add(value, [key]):** A diferencia de put, si hay un valor con la misma clave, la petición falla y se genera un error 'ConstraintError'.

Una vez que la transacción se ha realizado y la cola de microtarefas está vacía, se hace un commit (consumación) automático. De forma general, podemos asumir que una transacción se consuma cuando todas sus peticiones fueron completadas y el código actual finaliza. Para poder detectar que la finalización ha sido exitosa podemos emplear este evento:

```
request.oncomplete = function() {  
  console.log("Transacción completa");  
};
```

Búsquedas:

Hay dos maneras de buscar en un almacén de objetos:

- Por clave o por rango de claves. En nuestro almacén "playlist", puede ser por un valor o por un rango de valores de playlist.clave.

```
function getAll() {  
  let resultado = bd.transaction('playlist').objectStore  
    ('playlist').getAll();  
  resultado.onsuccess = (e) => {  
    console.table(resultado.result);  
  }  
}
```

- Por algún otro campo del objeto. Esto requiere una estructura de datos adicional llamada índice "index".

```
function getArtista (bd, artista) {  
  //Se crea una transacción llamando a nuestro almacén de datos, esta vez de modo lectura.  
  const playlist = bd.transaction('playlist', 'readonly').objectStore('playlist');  
  
  //Creamos un index que no tiene por que ser único, nos servirá para hacer las búsquedas en nuestra base de datos.  
  const index = playlist.index('artista');  
  
  //Pasamos el index creado a la función de búsqueda 'getAll' y lo guardamos en la variable query.  
  let query = index.getAll(artista)  
  
  //Si la transacción ha sido exitosa, imprimiremos los resultados en formato de tabla.  
  query.onsuccess = (e) => {  
    console.table(query.result);  
  }  
}
```

Web SQL Database

Web SQL Database es una característica de HTML5 que permite almacenar y acceder a grandes cantidades de datos en el navegador del usuario mediante una interfaz similar a una base de datos relacional. Es una especificación desarrollada por el W3C que permite almacenar datos en una base de datos SQLite y acceder a ellos mediante sentencias SQL.

Web SQL Database se utiliza para almacenar y acceder a datos que necesitan ser accedidos y manipulados de forma eficiente, como grandes cantidades de datos de una aplicación, información de sesión, o datos que se utilizan para mantener el estado de una aplicación web.

Web SQL Database se puede utilizar desde javascript y ofrece una interfaz similar a las bases de datos relacionales, permitiendo almacenar, actualizar, consultar y eliminar datos mediante sentencias SQL. Sin embargo, es importante tener en cuenta que la especificación de Web SQL Database no está siendo mantenida y no es soportada por algunos navegadores modernos, incluyendo Google Chrome desde versión 18 y Safari desde versión 6.1, por lo que su uso no se recomienda y se recomienda usar IndexedDB como una alternativa.

Integración avanzada de componentes

La integración avanzada de componentes se refiere a la incorporación de componentes de software pre-construidos en una aplicación web para agregar funcionalidad adicional. Los componentes pueden ser desarrollados por terceros o internamente y pueden ser reutilizados en varias aplicaciones.

De esta manera, podemos mejorar significativamente la eficiencia del desarrollo de aplicaciones web, ya que permite a los desarrolladores reutilizar componentes existentes en lugar de tener que construirlos desde cero. Esto puede ahorrar tiempo y reducir los costos de desarrollo.

Algunos ejemplos de componentes comunes que se pueden utilizar en aplicaciones web incluyen:

- **Componentes de interfaz de usuario (UI):** Estos componentes pueden incluir botones, menús desplegables, tablas, formularios y otros elementos de interfaz gráfica.
- **Componentes de automatización:** Estos componentes pueden incluir librerías de pruebas automatizadas, herramientas de automatización de trabajo, y componentes de integración de sistemas.
- **Componentes de seguridad:** Estos componentes pueden incluir mecanismos de autenticación, encriptación, y protección contra ataques de inyección de SQL.
- **Framework de desarrollo:** Los frameworks de desarrollo incluyen conjunto de librerías, patrones, y principios que se utilizan para facilitar el desarrollo de aplicaciones web, son un gran ejemplo de componentes que facilitan el desarrollo de aplicaciones web.
- **Componentes de terceros:** Muchas empresas y desarrolladores independientes ofrecen componentes que se pueden integrar en aplicaciones web. Estos componentes pueden incluir elementos de interfaz de usuario personalizados, herramientas de análisis, y componentes de seguridad.

Al integrar componentes en una aplicación web, es importante asegurarse de que sean compatibles con la aplicación y que se ajusten a los estándares de calidad y seguridad adecuados. Es recomendable revisar la documentación, las revisiones y las calificaciones de los componentes antes de usarlos, y mantenerlos actualizados siempre.

Además, algunas veces puede haber problemas de compatibilidad entre componentes, especialmente cuando se utilizan componentes de diferentes proveedores, por eso es importante testear la integración antes de implementarlo en producción.

Componentes de interfaz de usuario (UI)

Los componentes de interfaz de usuario son elementos visuales que se utilizan en una página web para interactuar con el usuario. **El objetivo principal de los componentes de UI es proporcionar una interfaz fácil de usar e intuitiva** para que los usuarios puedan interactuar con el contenido de una página web de manera eficiente, simple y atractiva.

Los elementos de interfaz de usuario se puede clasificar en las siguientes categorías:

Control con inputs

El control con inputs sirve principalmente para aportar información por parte del usuario, lo que lo convierte en una de las partes más importantes de la interfaz de usuario.

Se suelen utilizar para implementar formularios (<form>): un formulario es un conjunto de inputs que tienen una intención específica; por ejemplo: un login o registro de usuario, para establecer un medio de contacto via email o bien para escoger diversas opciones de personalización de diferentes aspectos. Es importante que los formularios sean concisos y no sobrecojan al usuario con demasiados inputs o que contengan campos totalmente irrelevantes.

Tipos de elementos: checkboxes, buttons, input text, input date, dropdown lists, toggles, select, etc..

Ejemplo de un formulario una vez implementado:

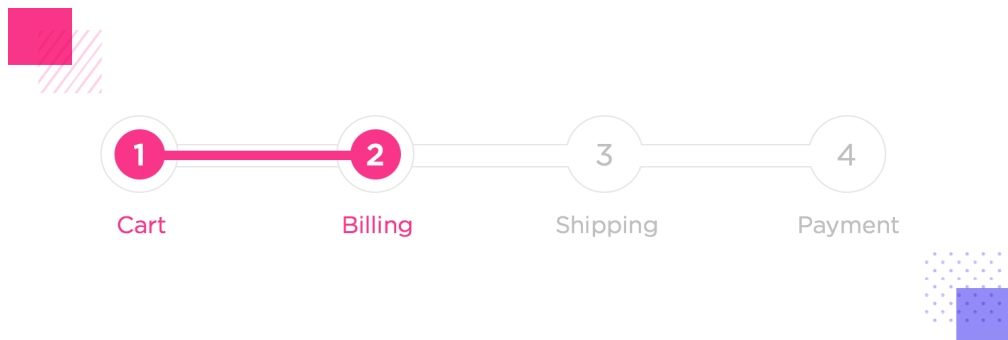
The image displays a variety of user interface (UI) controls on a light blue background. On the left, there is a text input field labeled 'Your full name' containing 'Jane Smith'. Below it are two columns of controls: 'Checkboxes' with three options (Option 1, Option 2, Option 3) where Option 1 and Option 3 are checked, and 'Radio buttons' with three options (Option 1, Option 2, Option 3) where Option 2 is selected. To the right of these is a dropdown menu labeled 'Select one option' with 'London' selected and a list of other options: New York, Sidney, Ottawa, and Wellington. Below the dropdown is a horizontal range slider. At the bottom left is a tabbed interface with three tabs: 'Tab 1' (active), 'Tab 2', and 'Tab 3'. Below the tabs is a large button labeled 'Button' with a right-pointing arrow. At the bottom right is a date picker control showing the number '3' and a 'Select a date' button with a calendar icon.

Componentes de navegación

Esta categoría sirve para esclarecer la situación en la que se encuentra el usuario, esté donde esté en nuestra página/aplicación. Los componentes de navegación deben estar muy claros para servir de guía para **todo tipo de usuarios, ya estén familiarizados con los elementos de la página o no**.

Tipos de elementos: breadcrumb, slider, barra de búsqueda, paginación, carrusel, etiquetas, iconos, etc...

Ejemplo de breadcrumb: en específico este sirve para situar al usuario en los diferentes pasos a seguir a la hora de comprar un producto en una página online, dejando claro los pasos realizados, el paso en el que está y los pasos que faltan.

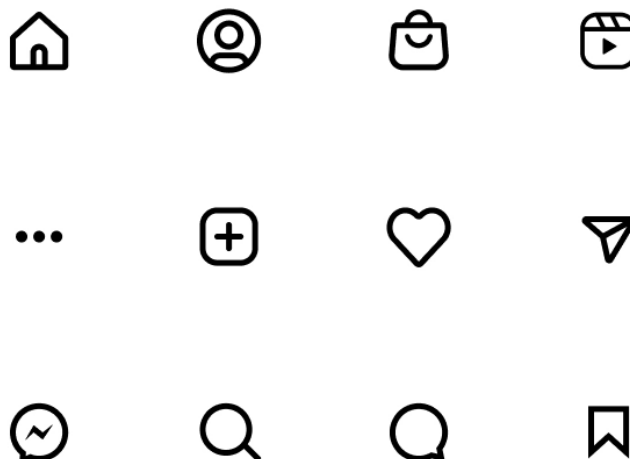


Componentes de información

Los componentes de información son similares a los de navegación, en tanto que son elementos complementarios muchas veces. La diferencia es que **los componentes de información deben transmitir una información muy específica** y no siempre está necesariamente relacionada con la situación del usuario en nuestra página/aplicación.

Tipos de elementos: tooltips, iconos, progress bar, notificaciones, message boxes, etc...

Ejemplo de iconos: estos son algunos de los iconos que utiliza Instagram.

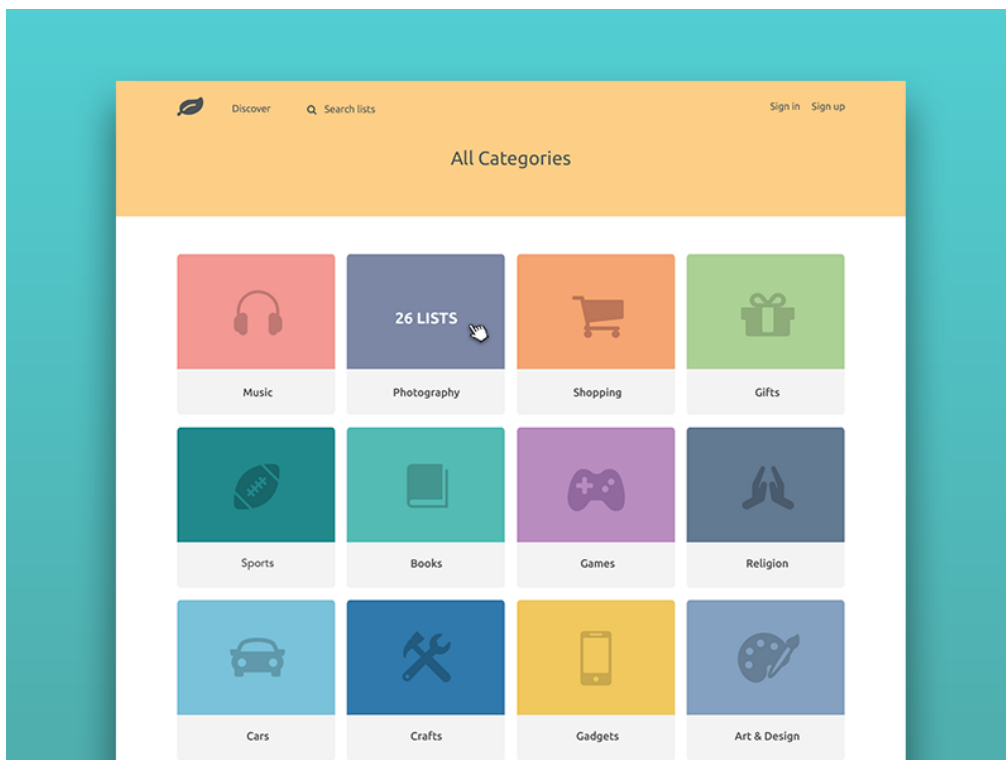


Contenedores

Los elementos contenedores usualmente se utilizan para implementar menús. **Los menús son un componente común de la interfaz de usuario en muchos tipos de aplicaciones**, estos tienen diferentes formatos y formas y sirven para agrupar la información por secciones

Tipos de elementos: acordeón, menú emergente, menú de opciones, bento menús: Bento, Hamburger, meatball, kebab...

Ejemplo de un menú principal de una aplicación:



Los componentes de UI se pueden crear e implementar utilizando, principalmente, HTML, CSS y JavaScript. Esencialmente, cada una de estas tecnologías se suele utilizar para:

- **HTML** proporciona la estructura básica de los componentes.
- **CSS** se utiliza para darles estilo y diseño.
- **JavaScript** sirve para proporcionar funcionalidad dinámica.

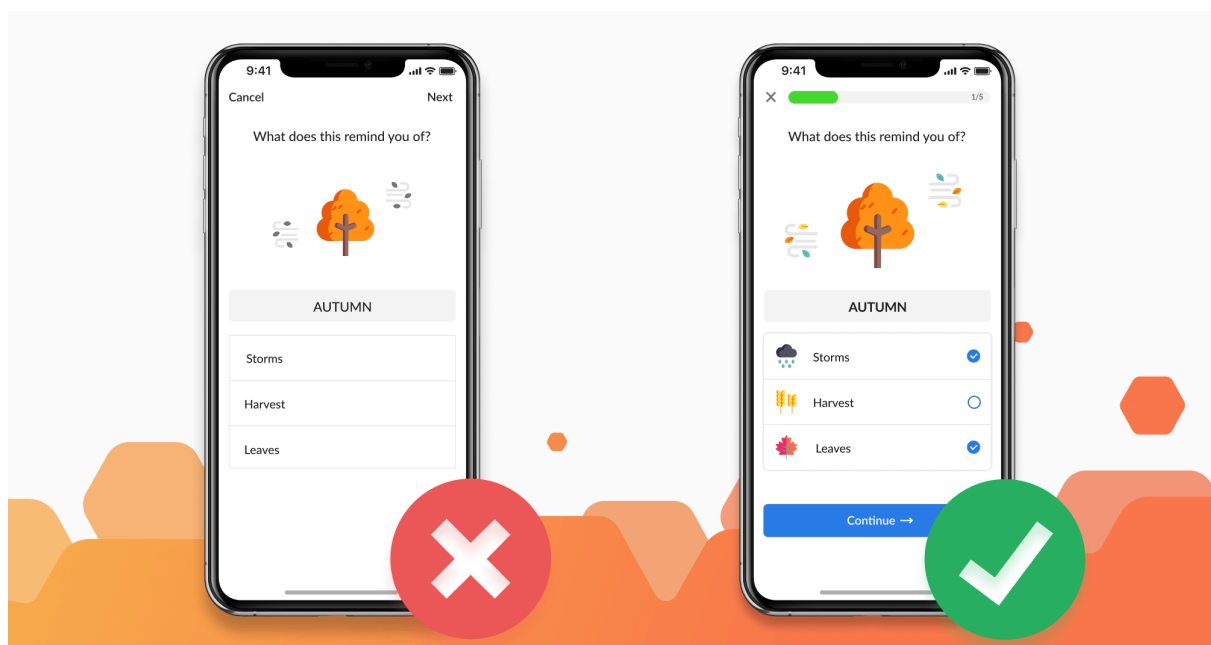


Existen muchas **bibliotecas y frameworks de UI disponibles para hacer más sencillo el desarrollo de componentes** de UI, como Bootstrap, Foundation o Material-UI. Estas bibliotecas y frameworks proporcionan un conjunto de componentes pre-construidos y estilos predefinidos que se pueden utilizar para crear una interfaz de usuario de forma consistente. Los frameworks se tratarán más adelante.



Los componentes de UI son muy importantes a la hora de establecer contacto con los usuarios, ya que de no tener una interfaz **intuitiva y sencilla** para facilitar la actividad del usuario, este puede simplemente buscar otro proveedor o abandonar la página/aplicación.

Además de ser eficiente, la interfaz de usuario ha de ser atractiva y representativa para hacer agradable la estancia de usuario y simultáneamente dejar una buena impresión al usuario lo que potencialmente generaría su vuelta al uso de la página/aplicación o incluso nuevos usuarios, por lo que la labor de diseño toma mucha importancia, casi tanto como la funcionalidad.



Componentes de automatización

Entendemos por automatización, en el contexto del desarrollo, el uso de herramientas para la realización de procesos minimizando la intervención humana durante los mismos, consiguiendo de esta manera reducir la posibilidad de cometer errores durante el proceso de desarrollo o creación de una aplicación o sitio web. Además, supone un aumento en la productividad y de la eficiencia en el proceso de desarrollo.

La automatización puede aplicarse a diversas partes del proceso, pero para que su uso sea eficaz debe llevarse a cabo una buena planificación y toma de decisiones respecto a las partes que tendrá nuestro proyecto y las necesidades que debe satisfacer.

Generalmente se aplican sobre las siguientes partes del proceso:

- Preparación del entorno o infraestructura necesaria para la aplicación.
- Configuración y personalización/adaptación de la infraestructura a las necesidades de nuestro proyecto.
- Organización de los servicios a usar.
- Implementación de las aplicaciones o servicios, propios o de terceros, que formen parte de nuestro proyecto.
- Aspectos de seguridad.
- Pruebas, testeo o debugging.

En lo referente a JavaScript, nos encontramos que, debido a la naturaleza del lenguaje, estas herramientas están centradas en la realización de pruebas, test o debugging, aunque los *frameworks* y otras herramientas pueden proporcionarnos herramientas de otras áreas como la seguridad.

Respecto a la automatización de test nos encontramos con diversas herramientas, por lo que tendremos la posibilidad de elegir la herramienta que mejor se adapte a nuestras necesidades.

Jest



Desarrollada por Facebook. Se trata de una de las herramientas de automatización más populares. Debe su popularidad a su enfoque simplista para su uso e integración.

Se trata de una gran opción por su interfaz de usuario que prima la facilidad de uso además de primar la velocidad de realización de los test. Destaca por su soporte de *cross-browser*¹ y robustez.

Mocha



Esta herramienta destaca por su implementación e idoneidad para aplicaciones pensadas para Node. Se trata de una herramienta fácil de usar y que cuenta con una extensa documentación y comunidad.

Otro de los aspectos primordiales es su flexibilidad, que viene dada por ser Open-Source. Esta flexibilidad viene dada por sus vastas capacidades en la realización de test.

Cypress



Se trata de una herramienta que permite la creación de test, debugging y la automatización de los mismos en los diversos procesos de integración. Puede usarse junto a Mocha, por lo que su uso está indicado para aplicaciones pensadas para Node.

Destaca frente a otras herramientas por la sencillez para implementar test automatizados, su capacidad de proporcionar resultados anticipados, una vez se ha configurado la automatización de tests, su rapidez, la toma de instantáneas de cada una de las partes del proceso y por su capacidad de modificación de elementos front-end y back-end.

¹ Son test de análisis del comportamiento de una página web en diferentes navegadores web, con el fin de proporcionar una buena experiencia de uso para los usuarios sin importar el navegador web que utilicen.

Karma



Se trata de una herramienta open-source, que destaca por su capacidad de configuración y adaptación al entorno en el que vamos a realizar los test, por su capacidad de ser usado junto a otros *frameworks* y por la posibilidad de elegir diversos dispositivos y entornos para realizar los test. Su punto diferencial frente a otras herramientas es su integración de CI/CD².

Cuenta con una documentación extensa y buena integración con diferentes IDEs.

Angular



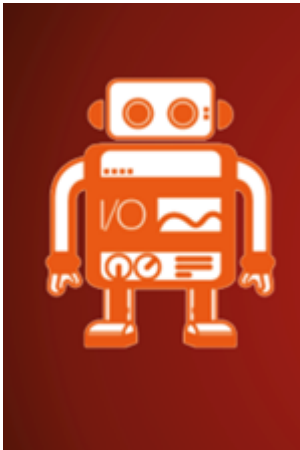
Se trata de un *framework* mantenido por Google. Destaca por facilitar el proceso de creación de aplicaciones web, además cuenta con una numerosa comunidad y se trata de una herramienta muy extendida.

En lo relativo a sus capacidades para realizar test, se caracteriza por su rapidez en la realización de test y la posibilidad de personalizar los elementos y aspectos a tener en cuenta en la realización de tests.

² Método de integración y distribución de aplicaciones a clientes de forma frecuente. Debemos distinguir entre:

- **Distribución continua (CD):** sometimiento a pruebas y errores de forma automática a los cambios que realiza el equipo de desarrollo antes de su incorporación a la producción.
- **Implementación continua (CI):** es el lanzamiento automático de los cambios realizados por el equipo de desarrollo desde el repositorio a producción, aprovechando los test automatizados, minimizando el tiempo de disponibilidad para el cliente.

Webdriver IO



Destaca por su implementación nativa de test para navegadores de escritorio y para dispositivos móviles. Por lo que se trata de la herramienta indicada en el caso de que tengamos que realizar test para múltiples dispositivos.

Otro aspecto destacado es su compatibilidad con otros *frameworks* y por su facilidad para poner en marcha los test.

Componentes de seguridad

Entendemos por componentes de seguridad la serie de medidas y herramientas utilizadas con el fin de garantizar la seguridad de las diferentes partes de una aplicación web. Como elementos de una aplicación web podemos encontrar bases de datos, sistemas de terceros, además de otros componentes internos o servicios web.

Son elementos comunes para todos los distintos lenguajes de programación ya que todos ellos deben hacer frente a los mismos desafíos a la hora de ser implementados en el desarrollo de una aplicación o servicio.

Muchos de estos componentes de seguridad vienen integrados en los diferentes *frameworks* o son proporcionados por distintas herramientas, pero las necesidades a nivel de seguridad que requiera nuestro proyecto deben ser un factor determinante para elegir entre las diferentes opciones disponibles, ya sea por la facilidad de implementación o bien por el nivel de seguridad que nos brindan.

A nivel general

En primer lugar, vamos a hablar de medidas relativas a los componentes de seguridad de manera general y luego se especificará su implementación o adaptación con JavaScript.

El principal objetivo de integrar componentes avanzados de seguridad es garantizar la confidencialidad, integridad y disponibilidad de los datos almacenados, por ello se usan algunas medidas de seguridad como:

- Cifrado de datos: se usa para proteger la confidencialidad de los datos. Se convierten datos en texto plano a un formato codificado.
- Control de acceso: para conceder el acceso a los datos solamente a los usuarios y sistemas autorizados.
- Autenticación: Usada para verificar la identidad del usuario, de forma previa, que quiere acceder a los datos o al servicio. Existen distintas maneras de hacerlo:
 - o Mediante QR.
 - o Doble factor de autenticación.
 - o Códigos de un solo uso.
 - o Biometría
 - o Certificado digital

En lo relativo a la protección frente a posibles ataques maliciosos, se encuentra la *IDPS (Intrusion Detection and Prevention System)*. Son mecanismos implementados para prevenir ataques o amenazas y accesos no autorizados a nuestra aplicación web.

La implantación de componentes de seguridad no sirve de nada si no se acompaña de una serie de medidas y de buenas prácticas durante el desarrollo de la aplicación. Esto puede establecerse en una serie de políticas de seguridad que incluyan prácticas cuyo seguimiento debe ser prioritario, ya sea en la implantación o mantenimiento de componentes. Siendo imprescindible mantener un control de las actualizaciones y vigilancia de los diferentes componentes, no solo los de seguridad, con el fin de evitar posibles vulnerabilidades.

Además, la implantación de estos componentes debe estar prevista en las diferentes fases de vida del proyecto, para que su integración sea lo más óptima posible y no se trate de un elemento añadido sobre una base ya existente.

Todos estos elementos en conjunto nos permitirán tener una aplicación segura y lo más protegida posible frente a diferentes vulnerabilidades y problemas de seguridad.

En lo relativo a los componentes de seguridad y JavaScript debemos tener en cuenta el creciente uso de JavaScript en entorno servidor, por lo que los componentes de seguridad cobran aún más importancia.

Para saber cómo implementar los componentes de seguridad es necesario conocer las amenazas que pueden aparecer con el uso de JavaScript.

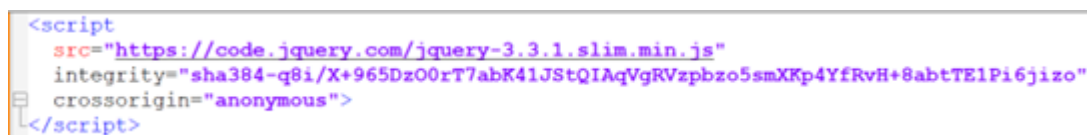
Comprobación de integridad. Atributo Integrity

El uso de la etiqueta `<script>` para la importación de librerías de terceros puede suponer ciertos riesgos, ya que haciendo uso de esta etiqueta llevamos a cabo el renderizado de recursos externos en nuestra aplicación o sitio web.

Las posibles brechas pueden venir del código fuente del recurso que hemos importado o bien de cambios que se hayan hecho sobre ese código por terceros. Ante el primer supuesto, debemos llevar a cabo una verificación de que el recurso importado no tiene vulnerabilidades. Mientras que en el segundo supuesto contamos con una medida de seguridad llamada *SRI* (*Subresource integrity*).³

Se hace añadiendo un atributo a la etiqueta `<script>`, este atributo permite la comprobación de la integridad del recurso importado.

Podemos ver un ejemplo de su uso en la siguiente imagen:



```
<script
  src="https://code.jquery.com/jquery-3.3.1.slim.min.js"
  integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
  crossorigin="anonymous">
</script>
```

³ Medida de seguridad que permite a los navegadores verificar si los recursos que obtienen se han enviado sin ninguna manipulación inesperada. La comprobación se lleva a cabo mediante el uso de encriptación Hash. Solamente se usará el recurso en el caso de que la comprobación de la integridad sea exitosa.

Test frecuentes para detectar vulnerabilidades en NPM

NPM o *Node Package Manager* es el gestor de paquetes usado por Node.js. En el caso de que tengamos dependencias instaladas es necesario llevar a cabo auditorías de vulnerabilidades con frecuencia para evitar que se acumulen y solucionarlas lo antes posible.

Se trata de un proceso de vital importancia ya que puede generar vulnerabilidades en la seguridad de nuestra aplicación. Se lleva a cabo usando el comando **npm audit**, en el directorio del comando.

Ejemplo de un informe de seguridad.

=== npm audit security report ===	
# Run <code>npm install react-scripts@4.0.1</code> to resolve 109 vulnerabilities	
SEVER WARNING: Recommended action is a potentially breaking change	
Low	Regular Expression Denial of Service
Package	braces
Dependency of	react-scripts
Path	react-scripts > babel-jest > babel-plugin-istanbul > test-exclude > micromatch > braces
More info	https://npmjs.com/advisories/786
Low	Regular Expression Denial of Service
Package	braces
Dependency of	react-scripts
Path	react-scripts > jest > jest-cli > jest-config > babel-jest > babel-plugin-istanbul > test-exclude > micromatch > braces
More info	https://npmjs.com/advisories/786

Permitir las actualizaciones para parches y versiones menores

Relativa a las versiones de los diferentes paquetes que se hayan importado dentro de un proyecto.

La ausencia de actualizaciones incrementa las posibilidades de que se aprovechen vulnerabilidades existentes sin reparar, por lo que se recomienda permitir la actualización en el caso de versiones menores o bien de parches de seguridad. Aunque si se tratan de componentes críticos de la aplicación es necesario comprobar previamente que estas actualizaciones no ponen en riesgo el funcionamiento de nuestra aplicación debido a la existencia de bugs a consecuencia de las actualizaciones o parches.

Validaciones para evitar inyecciones

Además de la validación en entorno cliente es necesario implementar una serie de validaciones en el lado del servidor, ya que las validaciones del lado cliente pueden burlarse de manera relativamente sencilla. Se trata de una parte importante de la seguridad de una aplicación que implemente JavaScript, o cualquier otro lenguaje de programación.

Algunas de las formas de evitarlo son:

- Uso de Frameworks seguros. Llevan a cabo codificaciones automáticas del contenido con el fin de evitar este tipo de ataques.
- Sustitución de caracteres y validaciones de los diferentes tipos de inputs. Como por ejemplo, sustituir los signos < o > por < o >.
- Codificación sensitiva al contexto (para prevenir XSS DOM).
- Habilitar Políticas de Seguridad de Contenido.

Todo ello con el fin de evitar el XSS (*Cross-Site Scripting*) y su uso para el robo de sesiones de usuarios y de sus datos almacenados en el navegador. También pueden estar dirigidos al robo de información contenida en el servidor como en el caso del acceso al contenido de las bases de datos. En el caso de JavaScript, este tipo de ataques puede llevarse a cabo de diversas maneras, siendo la modificación del DOM una de las más comunes.

Uso del modo Estricto

Es una manera de evitar la creación de código inseguro durante la creación de una aplicación o de sus componentes. Para usarlo se debe añadir *"use strict"* como primera línea de los ficheros JavaScript usados.

Activar el modo estricto supone:

- Avisos de errores que hayan podido pasar desapercibidos previamente.
- Arreglo de errores que pueden dificultar la lectura del código por parte de los motores de JavaScript, permitiendo una ejecución más optimizada.
- Limitación en el uso de posibles palabras reservadas.
- Avisos de errores ante acciones que pueden generar vulnerabilidades.

Herramientas de análisis del código

Se tratan de herramientas aplicadas al código, no son exclusivas de JavaScript, especialmente útiles para lenguajes interpretados como JavaScript. Ayuda a establecer una serie de parámetros de calidad en el código y el uso de posibles fallos dentro del mismo que puedan suponer brechas de seguridad dentro de la aplicación.

Esto se consigue mediante:

- Diagnóstico de errores de sintaxis y problemas técnicos.
- Consistencia en el código.
- Medidas objetivas de calidad del código.
- Estableciendo una guía de estilo y estándares para el código.

En el caso de JavaScript se pueden usar JSLint, Sonar, JSHint o ESLint.

El uso de este tipo de herramientas busca que se genere un código de la mayor calidad posible con el fin de reducir potenciales problemas de seguridad y otros problemas durante la fase desarrollo y la vida de una aplicación.

Control del envío de código y del código enviado

Esto puede llevarse a cabo mediante el uso de minificación o bien reduciendo al máximo el código enviado al cliente durante el uso. Como el segundo supuesto puede ser más difícil de realizar, se debe priorizar el uso de minificación ya que además ayuda al rendimiento de la aplicación.

Se trata de eliminar espacios en blanco, indentación, eliminación de comentarios y otras modificaciones con el fin de dificultar su lectura y comprensión, además de mejorar la velocidad de un sitio web o de una aplicación al reducir el tamaño del fichero que se carga en cliente. Esto se puede ver en los siguientes ejemplos:

Código antes de minificar:

```
// program to find the factorial of a number

// take input from the user
const number = parseInt(prompt('Enter a positive integer: '));

// checking if number is negative
if (number < 0) {
    console.log('Error! Factorial for negative number does not exist.');
```

```
}

// if number is 0
else if (number === 0) {
    console.log(`The factorial of ${number} is 1.`);
}

// if number is positive
else {
    let fact = 1;
    for (i = 1; i <= number; i++) {
        fact *= i;
    }
    console.log(`The factorial of ${number} is ${fact}.`);
}
```

Código después de haber sido minificado:

```
const number=parseInt(prompt('Enter a positive integer: '));if(number<0){console.log('Error! Factorial for negative  
number does not exist.')}  
else if(number==0){console.log('The factorial of ${number}is 1.')}  
else{let fact=1;for(i=1;i<=number;i++){fact*=i;}  
console.log('The factorial of ${number}is ${fact}.')}
```

Todo esto se hace con el fin de evitar que atacantes aprovechen la lógica de nuestra aplicación para sus ataques. Por ello, lo ideal sería combinar ambas estrategias de control de código.

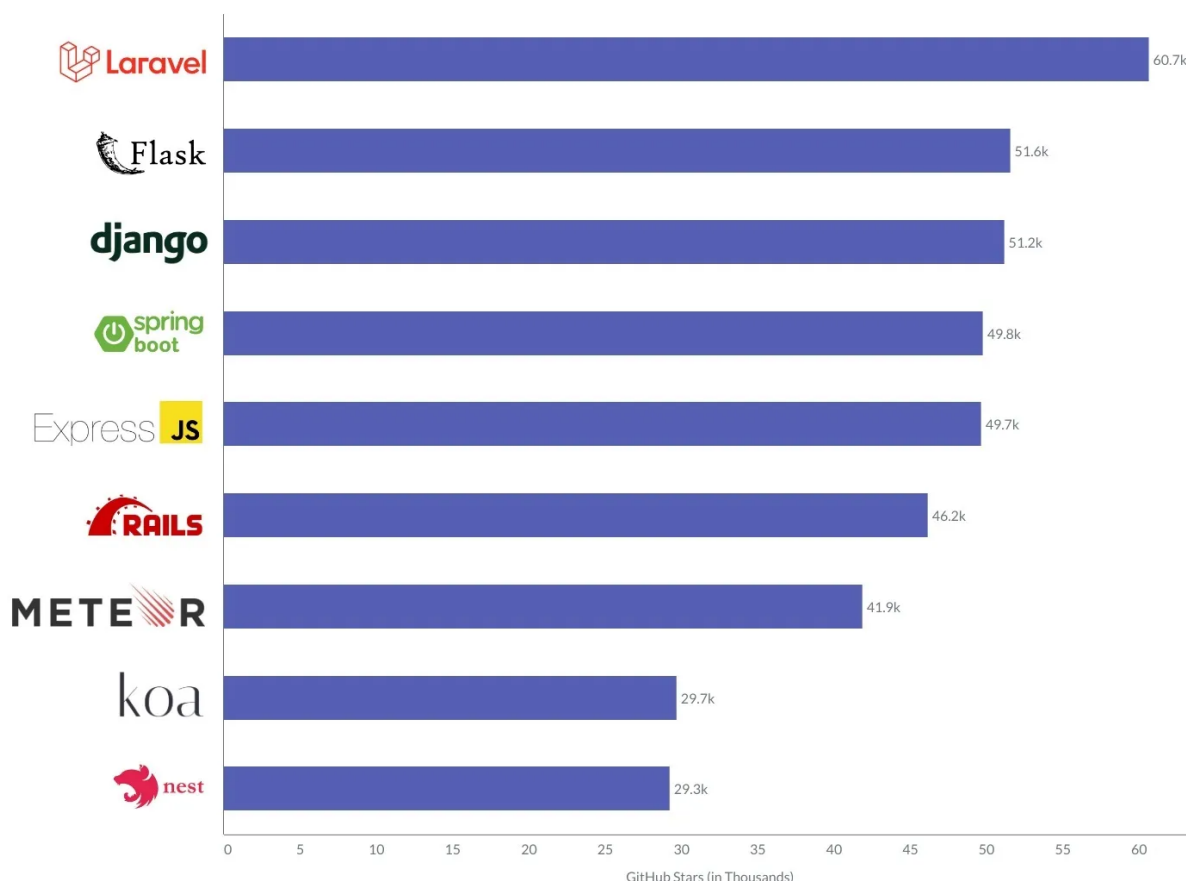
Frameworks de desarrollo

Los frameworks de desarrollo son conjuntos de herramientas y librerías que proporcionan una estructura básica para el desarrollo de aplicaciones web. En la integración avanzada de componentes web, estos frameworks proporcionan una forma de integrar y conectar diferentes componentes de una aplicación web de manera eficiente.

Es importante distinguir entre frameworks *front end* y frameworks *back end*. Los primeros facilitan la creación de interfaces y elementos visuales, mientras que los últimos se utilizan para facilitar la programación de los elementos del servidor (conexiones con bases de datos, resoluciones de lógica, etc).

Es común, sobre todo en front end, que varios frameworks se acumulen unos encima de otros para mejorar prestaciones y optimizar procesos de forma escalonada. Por ejemplo, Angular.js, que está desarrollado en TypeScript, que a su vez es otro framework que transforma Javascript en un lenguaje de tipado fuerte con clases, interfaces y módulos.

Aquí podemos ver los frameworks más populares del último año por cantidad de estrellas en Github:



Frameworks back end

La mayoría de frameworks back end comparten una estructura similar. Generan una jerarquía ordenada de archivos que luego los desarrolladores hacen crecer con todo lo necesario.

La mayoría de ellos se basa en el modelo de diseño MVC (Model View Controller), aunque se están empezando a popularizar otros modelos de diseño como MVVM (Model View ViewModel).

Algunos de los más utilizados son:

- Django: Este framework de Python es utilizado para la creación de aplicaciones web complejas y es conocido por su escalabilidad, seguridad y productividad. La estructura básica de un proyecto de Django se vería así:

```
mysite # django-admin startproject <nombre>
├── db.sqlite3 # Por defecto base de datos SQLite
├── manage.py # Se crea solo, manejar el proyecto
├── mysite # mysite configuraciones generales del proyecto
│   ├── asgi.py # tiene que ver con el despliegue.
│   ├── __init__.py # está vacío. Indica que es un módulo e python
│   ├── settings.py # configuración general
│   ├── urls.py # Información para el router. GENERAL. distribuidor de peticiones
│   └── wsgi.py # tiene que ver con el despliegue.
├── polls # Primera aplicación
│   ├── admin.py # lo veremos. Tiene el scaffolding (Andamiaje)
│   ├── apps.py # Define info de polls
│   ├── __init__.py # está vacío. Indica que es un módulo e python
│   ├── migrations # Cambio en la base de datos
│   │   └── __init__.py
│   ├── models.py # Defino clases de datos y lógica de negocio
│   ├── tests.py # Unit testing
│   ├── urls.py # Defino quién controla esa petición (vista -> "controlador")
│   └── views.py # interactúa con modelos y lo pinta en un template (template -> "vista")
```

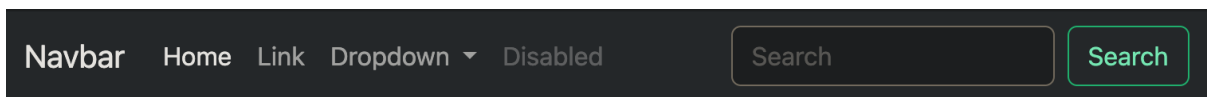
- Laravel: Este framework de PHP es popular por su simplicidad y facilidad de uso, y se utiliza para el desarrollo de aplicaciones web complejas y aplicaciones móviles.
- Ruby on Rails: Este framework de Ruby es conocido por su enfoque en la productividad y la rapidez de desarrollo, y se utiliza para la creación de aplicaciones web dinámicas.
- Express.js: Este framework de Node.js es ligero y flexible, y se utiliza para el desarrollo de aplicaciones web y API en el lado del servidor.
- Spring: Este framework de Java es una plataforma completa para el desarrollo de aplicaciones empresariales, con un enfoque en la seguridad y la escalabilidad.

Frameworks front end

Basándonos en un modelo de diseño MVC, los frameworks front end se especializan en trabajar sobre la View.

entre los más populares del momento, encontramos:

- **Bootstrap:** Este framework proporciona plantillas de html y clases prefabricadas de css para aplicar estilos con comodidad. Aquí un ejemplo de navbar predefinida de bootstrap, altamente personalizable:



- **Jquery:** Este framework (o biblioteca de métodos) es una mejora sustancial de la engorrosa sintaxis por defecto de javascript. Aquí tenemos un ejemplo:

```
// Vanilla JavaScript
document.querySelector('.highlight').style.backgroundColor = 'yellow';

// jQuery
$('.highlight').css('background-color', 'yellow');
```

Como se puede ver, la sintaxis de jQuery es más concisa y legible que la de Vanilla JavaScript. Además, jQuery proporciona una gran cantidad de métodos adicionales para manipular y animar elementos HTML y CSS de forma más fácil y eficiente. A partir de aquí, tenemos frameworks que cambian la manera que tenemos de estructurar las vistas:

- **React.js:** Este framework de javascript es perfecto para la escritura de interfaces de forma declarativa, con módulos altamente reutilizables.
- **Angular:** Plataforma completa para el desarrollo de aplicaciones web basada en javascript, que permite crear aplicaciones interactivas y modernas.
- **Vue.js:** Este framework de javascript es sencillo de aprender y utilizar, proporciona un enfoque en componentes y una alta personalización.
- **Ember.js:** Proporciona una plataforma sólida para la creación de aplicaciones web ambiciosas, con un enfoque en la escalabilidad y la productividad.

Componentes de terceros

Los componentes de terceros en la integración avanzada de componentes en desarrollo web son software o servicios proporcionados por compañías diferentes a la que está desarrollando la aplicación. Estos componentes pueden incluir servicios web, bases de datos, sistemas de terceros, y otros componentes de software que son utilizados en una aplicación web.

La integración de estos componentes de terceros en una aplicación web puede proporcionar una serie de beneficios, como aumentar la funcionalidad de la aplicación, reducir el tiempo y el esfuerzo requeridos para desarrollar ciertas funciones, y ahorrar costos. Sin embargo, también puede presentar desafíos en cuanto a la seguridad y la compatibilidad.

Algunos ejemplos de componentes de terceros:

- Servicios de autenticación como Google o Facebook.
- Servicios de pagos como PayPal o Stripe.
- Servicios de análisis de datos como Google Analytics.
- Servicios de mensajería como Twilio.
- Servicios de almacenamiento en la nube como Amazon S3 o Google Drive.

La integración de estos componentes de terceros en una aplicación web puede proporcionar una serie de beneficios, como aumentar la funcionalidad de la aplicación, reducir el tiempo y el esfuerzo requeridos para desarrollar ciertas funciones, y ahorrar costos, pero también puede presentar desafíos en cuanto a la seguridad y la compatibilidad.

Para integrar componentes de terceros en una aplicación web, se utilizan diferentes enfoques, como el uso de APIs (interfaz de programación de aplicaciones), SDK (kit de desarrollo de software) o plugins. Estos enfoques permiten conectar la aplicación con los servicios o componentes de terceros y utilizarlos en la aplicación.

Apis

Una API (Application Programming Interface) es un conjunto de reglas y protocolos que permite la comunicación entre diferentes aplicaciones o componentes de software. Es decir, una API es una interfaz que permite a dos sistemas interactuar entre sí.

Las APIs permiten que una aplicación acceda a datos o funciones de otra aplicación. Por ejemplo, la API de Google Maps permite a una aplicación web mostrar mapas y obtener información sobre direcciones, mientras que la API de Twitter permite a una aplicación acceder a los datos de una cuenta de Twitter.

Las APIs son ampliamente utilizadas en la industria para integrar diferentes sistemas y mejorar la eficiencia. Además, también se utilizan para crear aplicaciones de terceros y para proporcionar a los desarrolladores acceso a los datos y funcionalidades de una aplicación.

Algunos ejemplos:

Google Maps API - permite agregar mapas interactivos y funcionalidad de geolocalización a aplicaciones web y móviles.

Twitter API - permite acceder a datos de Twitter, como tweets, usuarios y tendencias, para integrarlos en otras aplicaciones.

Stripe API - permite añadir funcionalidad de pagos en línea a aplicaciones, como la gestión de tarjetas de crédito y la realización de transferencias.

Spotify Web API - permite acceder a los datos de la plataforma de música en línea, como listas de reproducción, álbumes y canciones.

OpenWeatherMap API - permite acceder a información meteorológica, como la temperatura y las condiciones climáticas actuales para una ubicación específica.

SDKs

Un SDK es un kit de desarrollo de software que proporciona herramientas y componentes para que los desarrolladores creen aplicaciones para una plataforma específica. Los SDKs suelen incluir bibliotecas, documentación, muestras de código, y herramientas de desarrollo para facilitar el desarrollo de software para una plataforma en particular, como un sistema operativo, una plataforma móvil, un videojuego, etc.

Algunos ejemplos:

Android SDK - permite desarrollar aplicaciones para dispositivos móviles con el sistema operativo Android.

Microsoft .NET SDK - permite desarrollar aplicaciones para la plataforma de Microsoft .NET, que incluye tecnologías como ASP.NET y C#.

Facebook SDK - permite integrar funcionalidades de Facebook, como el inicio de sesión y la publicación en el muro, en aplicaciones móviles y web.

Amazon AWS SDK - permite acceder a los servicios en la nube de Amazon, como Amazon S3 y Amazon DynamoDB, desde aplicaciones.

Twilio SDK - permite agregar funcionalidades de mensajería y llamadas a aplicaciones, como enviar mensajes de texto y hacer llamadas telefónicas.

¡Bibliografía

- SRI. https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity
- Integrity https://www.w3schools.com/Tags/att_script_integrity.asp
- Documentación oficial de React: <https://reactjs.org/>
- Curso en línea de Node.js, Express y MongoDB: <https://www.udemy.com/course/nodejs-express-mongodb-dev-to-deployment/>
- Tutorial de Ruby on Rails: https://guides.rubyonrails.org/getting_started.html
- Documentación oficial de Django: <https://docs.djangoproject.com/en/3.2/>
- Curso en línea de Vue.js y Node: <https://www.udemy.com/course/full-stack-web-development-with-vuejs-and-node/>
- Documentación oficial de Laravel: <https://laravel.com/docs/8.x/installation>
- Tutorial de Flask: <https://flask.palletsprojects.com/en/2.1.x/tutorial/>
- SessionStorage: <https://www.merkle.com/es/es/blog/almacenamientos-navegador-local-storage-session-storage-indexeddb-cookie>