



Programación de Aplicaciones Telemáticas

TEMA 5: JAVASCRIPT

A yellow square containing the letters 'JS' in a bold, dark grey font.

JS

AGENDA

- Introducción
- Core Language
- Javascript & Web Browser
- Formularios Web
- Referencias

SESIÓN 1: JAVASCRIPT CORE

INTRODUCCIÓN

- Creado por Brendan Eich (co-founder of the Mozilla Project)
- JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time)
- Es más conocido como un lenguaje de scripting
- JavaScript es un lenguaje de programación basada en prototipos
- Utilizado no solo en Navegadores Web (Por ejemplo NodeJS)

INTRODUCCIÓN

- No confundir JavaScript con el lenguaje de programación Java
- Ambos lenguajes de programación tienen sintaxis, semántica y usos muy diferentes
- Multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa (por ejemplo programación funcional)

CORE LANGUAGE

- Objetos, Tipos y Variables
- Expresiones y Operadores
- Sentencias y Declaraciones
- Funciones
- Expresiones Regulares (Match Data)

OBJETOS, TIPOS Y VARIABLES

OBJETOS



JavaScript Demo: Classes Constructor

```
1 class Polygon {  
2   constructor() {  
3     this.name = 'Polygon';  
4   }  
5 }  
6  
7 const poly1 = new Polygon();  
8  
9 console.log(poly1.name);  
10 // expected output: "Polygon"  
11
```

Run >

> "Polygon"

Reset

OBJETOS, TIPOS Y VARIABLES

TIPOS

Tipos de Datos:

- undefined : typeof instance === "undefined"
- Boolean : typeof instance === "boolean"
- Number : typeof instance === "number"
- String : typeof instance === "string"
- BigInt : typeof instance === "bigint"
- Symbol : typeof instance === "symbol"

OBJETOS, TIPOS Y VARIABLES

TIPOS

Tipos de Datos:

- Objetos : `typeof instance === "object"`
- Funciones : No es una estructura, devuelve un Tipo de Datos

Estructuras de Datos: `new Object`, `new Array`, `new Map`,
`new Set`, `new WeakMap`, `new WeakSet`, `new Date`

OBJETOS, TIPOS Y VARIABLES

TIPOS (OBJETOS)

```
var myCar = new Object();  
myCar.make = 'Ford';  
myCar.model = 'Mustang';  
myCar.year = 1969;
```

```
var myCar = {  
  make: 'Ford',  
  model: 'Mustang',  
  year: 1969  
};
```

OBJETOS, TIPOS Y VARIABLES

TIPOS (FUNCIONES)



JavaScript Demo: Functions Default

```
1 function multiply(a, b = 1) {  
2   return a * b;  
3 }  
4  
5 console.log(multiply(5, 2));  
6 // expected output: 10  
7  
8 console.log(multiply(5));  
9 // expected output: 5  
10
```

Run ›

Reset

> 10

> 5

OBJETOS, TIPOS Y VARIABLES

TIPOS (PROPIEDADES DE VALOR)

- Infinity
- NaN
- undefined
- null
- globalThis

OBJETOS, TIPOS Y VARIABLES

TIPOS (PROPIEDADES DE FUNCIONES)

- `eval()`
- `uneval()`
- `isFinite()`
- `isNaN()`
- `parseFloat()`
- `parseInt()`

OBJETOS, TIPOS Y VARIABLES

TIPOS (PROPIEDADES DE FUNCIONES)

- `decodeURI()`
- `decodeURIComponent()`
- `encodeURI()`
- `encodeURIComponent()`
- `escape()`
- `unescape()`

OBJETOS, TIPOS Y VARIABLES

TIPOS (OBJETOS FUNDAMENTALES)

- Object
- Function
- Boolean
- Symbol
- Error
- EvalError

OBJETOS, TIPOS Y VARIABLES

TIPOS (OBJETOS FUNDAMENTALES)

- `InternalError`
- `RangeError`
- `ReferenceError`
- `SyntaxError`
- `TypeError`
- `URIError`

OBJETOS, TIPOS Y VARIABLES

TIPOS (NÚMEROS Y FECHAS)

- Number
- BigInt
- Math
- Date

OBJETOS, TIPOS Y VARIABLES

TIPOS (TEXTO)

- String
- RegExp

OBJETOS, TIPOS Y VARIABLES

TIPOS (COLECCIONES INDEXADAS)

- Array
- Int8Array
- Uint8Array
- Uint8ClampedArray
- Int16Array
- Uint16Array

OBJETOS, TIPOS Y VARIABLES

TIPOS (COLECCIONES INDEXADAS)

- Int32Array
- UInt32Array
- Float32Array
- Float64Array
- BigInt64Array
- BigUint64Array

OBJETOS, TIPOS Y VARIABLES

TIPOS (COLECCIONES CON CLAVE)

- Map
- Set
- WeakMap
- WeakSet

OBJETOS, TIPOS Y VARIABLES

TIPOS (DATOS ESTRUCTURADOS)

- ArrayBuffer
- SharedArrayBuffer
- Atomics
- DataView
- JSON

OBJETOS, TIPOS Y VARIABLES

TIPOS (ABSTRACCIÓN DE CONTROL)

- Promise
- Generator
- GeneratorFunction
- AsyncFunction

OBJETOS, TIPOS Y VARIABLES

VARIABLES Y CONSTANTES

Variables son contenedores que almacenan valores
(Objetos)

- var
- let
- const

OBJETOS, TIPOS Y VARIABLES

VARIABLES Y CONSTANTES

```
let foo = 42; // foo ahora es un número  
foo     = 'bar'; // foo ahora es un string  
foo     = true; // foo ahora es un booleano
```

EXPRESIONES Y OPERADORES

- Operadores de asignación
- Operadores de comparación
- Operadores aritmeticos
- Operadores bit a bit
- Operadores lógicos
- Operadores de cadenas de texto
- Operador condicional
- Operador coma
- Operadores unarios
- Operadores relacionales

EXPRESIONES Y OPERADORES

OPERADORES DE ASIGNACIÓN

Un operador de asignación asigna un valor a su operando izquierdo basándose en el valor de su operando derecho. El operador de asignación simple es igual (=), que asigna el valor de su operando derecho a su operando izquierdo. Es decir, $x = y$ asigna el valor de y a x

EXPRESIONES Y OPERADORES

OPERADORES DE COMPARACIÓN

Un operador de comparación compara sus operandos y devuelve un valor lógico en función de si la comparación es verdadera (true) o falsa (false). Los operandos pueden ser valores numéricos, de cadena, lógicos u objetos.

EXPRESIONES Y OPERADORES

OPERADORES ARITMETICOS

Un operador aritmético toma valores numéricos (ya sean literales o variables) como sus operandos y devuelve un solo valor numérico. Los operadores aritméticos estándar son suma (+), resta (-), multiplicación (*) y división (/). Estos operadores funcionan como lo hacen en la mayoría de los otros lenguajes de programación cuando se usan con números de punto flotante (en particular, ten en cuenta que la división entre cero produce Infinity).

EXPRESIONES Y OPERADORES

OPERADORES BIT A BIT

Un operador bit a bit trata a sus operandos como un conjunto de 32 bits (ceros y unos), en lugar de números decimales, hexadecimales u octales.

EXPRESIONES Y OPERADORES

OPERADORES LÓGICOS

Los operadores lógicos se utilizan normalmente con valores booleanos (lógicos); cuando lo son, devuelven un valor booleano. Sin embargo, los operadores `&&` y `||` en realidad devuelven el valor de uno de los operandos especificados, por lo que si estos operadores se utilizan con valores no booleanos, pueden devolver un valor no booleano.

EXPRESIONES Y OPERADORES

OPERADORES DE CADENAS DE TEXTO

Además de los operadores de comparación, que se pueden usar en valores de cadena, el operador de concatenación (+) concatena dos valores de cadena, devolviendo otra cadena que es la unión de los dos operandos de cadena.

EXPRESIONES Y OPERADORES

OPERADOR CONDICIONAL

El operador condicional es el único operador de JavaScript que toma tres operandos. El operador puede tener uno de dos valores según una condición.

La sintaxis es:

`condition ? val1 : val2`

EXPRESIONES Y OPERADORES

OPERADOR COMA

El operador coma (,) simplemente evalúa ambos operandos y devuelve el valor del último operando. Este operador se utiliza principalmente dentro de un bucle for, para permitir que se actualicen múltiples variables cada vez a través del bucle.

EXPRESIONES Y OPERADORES

OPERADORES UNARIOS

Una operación unaria es una operación con un solo operando.

EXPRESIONES Y OPERADORES

OPERADORES RELACIONALES

Un operador relacional compara sus operandos y devuelve un valor Boolean basado en si la comparación es verdadera.

SENTENCIAS Y DECLARACIONES

- Control de flujo
- Declaraciones
- Funciones
- Iteraciones

SENTENCIAS Y DECLARACIONES

CONTROL DE FLUJO

SENTENCIAS Y DECLARACIONES

CONTROL DE FLUJO

- **Empty:** Una sentencia vacía se utiliza para proveer una "no sentencia", aunque la sintaxis de JavaScript esperaba una
- **if...else:** Ejecuta una sentencia si una condición especificada es true. Si la condición es false, otra sentencia puede ser ejecutada
- **switch:** Evalua una expresión, igualando el valor de la expresión a una clausula case y ejecuta las sentencias asociadas con dicho case

SENTENCIAS Y DECLARACIONES

CONTROL DE FLUJO

- **throw:** Lanza una excepción definida por el usuario
- **try...catch:** Marca un bloque de sentencias para ser probadas (try) y especifica una respuesta, en caso de que se lance una excepción

SENTENCIAS Y DECLARACIONES

DECLARACIONES

- **var:** Declara una variable, opcionalmente inicializándola a un valor
- **let:** Declara una variable local de ámbito de bloque, opcionalmente inicializándola a un valor
- **const:** Declara una constante de solo lectura

SENTENCIAS Y DECLARACIONES

FUNCIONES

- **function:** Declara una función con los parámetros especificados
- **function*:** Los generadores de funciones permiten escribir iteradores con mas facilidad
- **async function:** Declara una función asíncrona con los parámetros especificados

SENTENCIAS Y DECLARACIONES

FUNCIONES

- **return:** Especifica el valor a ser retornado por una función
- **class:** Declara una clase

SENTENCIAS Y DECLARACIONES

ITERACIONES

- **do...while:** Crea un bucle que ejecuta una instrucción especificada hasta que la condición de prueba se evalúa como falsa. La instrucción especificada se ejecute al menos una vez
- **for:** Crea un bucle que consiste en 3 parametros opciones, seguido del bloque a ejecutar en cada iteración
- **for each...in:** Itera una variable especificada sobre todos los valores de las propiedades del objeto

SENTENCIAS Y DECLARACIONES

ITERACIONES

FUNCIONES

- Declaración
- Expresiones
- Alcance
- Parametros

FUNCIONES

DECLARACIÓN

Una función esta compuesta de los siguientes elementos:

- Nombre de la función
- Listado de parametros separados entre "comas"
- Instrucciones de la función separado por "{....}"

FUNCIONES EXPRESIONES

Las funciones se pueden declarar como expresiones:

```
const square = function(number) { return number * number }  
var x = square(4) // x gets the value 16
```

FUNCIONES

ALCANCE

Variables definidas dentro de una función no son visibles fuera, pero una función puede acceder a todas las variables y funciones definidas en el mismo "scope"

FUNCIONES

PARÁMETROS

- Los valores de los parámetros por defecto son "undefined"
- El número de parámetros puede ser dinámico
- Los parámetros no tienen que ser variables, pueden ser también objetos

PATTERN MATCHING WITH REGULAR EXPRESSIONS

Las expresiones regulares son patrones que se utilizan para hacer coincidir combinaciones de caracteres en cadenas. En JavaScript, las expresiones regulares también son objetos.

Estos patrones se utilizan con los métodos `exec()` y `test()` de `RegExp`, y con `match()`, `matchAll()`, `replace()`, `replaceAll()`, `search()` y `split()` métodos de `String`.

SESIÓN 2: JAVASCRIPT & WEB BROWSER

JAVASCRIPT & WEB BROWSER

- The Window Object
- Handling Events
- Scripted HTTP
- Client-Side Storage
- HTML5 APIs

THE WINDOW OBJECT (BOM & DOM)

- Modelo de objetos que nos permite interactuar con el navegador y sus elementos
- Soportado por todos los navegadores
- Tenemos acceso a todos sus elementos y funciones

BOM (BROWSER OBJECT MODEL)

- Redimensionar ventanas del navegador
- Obtener información del navegador (ventana actual): propiedades, dimensión, histórico, URL actual...
- Gestión de cookies
- Acceder al DOM
- **Funciones con temporizador:** `setTimeout(function() {}, timeoutMillis)` **y** `setInterval(function() {}, intervalMillis)`

EJEMPLOS

```
//Browser Object Model  
//recargar página  
window.location.reload()  
//navegar  
window.location.href = "https://google.es";  
//hitórico  
window.history  
//ir atrás  
window.history.back()  
window.history.go(-1)  
//actualizar  
window.history.go()  
//ir adelante  
window.history.forward()  
//Objeto navigator -> información del navegadores  
window.navigator  
//Manipular la screen  
//infor de la pantalla actual  
window.screen  
window.moveTo(0, 0);  
window.resizeTo(window.screen.availWidth, window.screen.availHeight);
```

DOM (DOCUMENT OBJECT MODEL)

- API para documentos HTML y XML
- Estándar del W3C
- Representación estructurada de los elementos del documento Web: elements HTML y nodes
- Permite la comunicación de JavaScript con el document Web

ACCESO A ELEMENTOS ESPECÍFICOS

- `getElementById()`
- `getElementsByName()`
- `getElementsByTagName()`
- `getElementsByClassName()`

ACCESO A ELEMENTOS ESPECÍFICOS

```
//obtener un elemento con id específico  
document.getElementById("input-password")  
//obtener Formularios  
document.getElementsByTagName("form")  
//obtener por Clases  
document.getElementsByClassName("card")  
//obteener por Nombre  
document.getElementsByName("myinput")
```

CREACIÓN DE ELEMENTOS Y NODOS

- `createElement(tag, options)`
- `createComment(comment)`
- `createText(text)`
- `cloneDeep()`

CREACIÓN DE ELEMENTOS Y NODOS

```
const div = document.createElement("div");      // Creamos un <div></div>
const span = document.createElement("span");   // Creamos un <span></span>
const img = document.createElement("img");     // Creamos un <img>
const comment = document.createComment("Comentario"); // <!--Comentario-->
const text = document.createTextNode("Hola");  // Nodo de texto: 'hola'

const div = document.createElement("div");
div.textContent = "Elemento 1";
div.id = "page";           // <div id="page"></div>
div.className = "data";   // <div id="page" class="data"></div>
div.style = "color: red"; // <div id="page" class="data" style="color: red"></div>

const div2 = div.cloneNode(); // Ahora SÍ estamos clonando
div2.textContent = "Elemento 2";

div.textContent; // 'Elemento 1'
```

INSERTAR ELEMENTOS

- `innerHTML, outerHTML`
- `innerText, outerText`
- `appendChild(element)`
- `insertAdjacentElement(position, element), insertAdjacentText()`
- `remove(), removeChild(node), replaceChild(new, old)`

INSERTAR ELEMENTOS

```
//Insercción de contenido
const data = document.querySelector(".data");
data.innerHTML = "<h1>Tema 1</h1>";
data.textContent; // "Tema 1"
data.innerHTML; // "<h1>Tema 1</h1>"
data.outerHTML; // "<div class='data'><h1>Tema 1</h1></div>"
const div = document.createElement("div"); // <div></div>
div.textContent = "Ejemplo"; // <div>Ejemplo</div>

const app = document.querySelector("#app"); // <div id='app'>App</div>
app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id='app'>App</div>
app.insertAdjacentElement("afterbegin", div);
// Opción 2: <div id='app'> <div>Ejemplo</div> App</div>
app.insertAdjacentElement("beforeend", div);
// Opción 3: <div id='app'>App <div>Ejemplo</div> </div>
app.insertAdjacentElement("afterend", div);
// Opción 4: <div id='app'>App</div> <div>Ejemplo</div>

const div = document.querySelector(".item:nth-child(2)"); // <div class='item'>2</div>
document.body.removeChild(div); // Desconecta el segundo .item

const div = document.querySelector(".item:nth-child(2)");
const newnode = document.createElement("div");
newnode.textContent = "DOS";
document.body.replaceChild(newnode, div);
```


MANIPULAR CLASSES CSS

- `className`, `getAttribute("class")`,
`setAttribute("class", class)`
- `classList`

INSERTAR ELEMENTOS

```
<div id="page" class="info data dark" data-number="5"></div>
//manipulación de clases de elementos
const div = document.querySelector(".info");
// Obtener clases CSS
div.className; // "info data dark"
div.getAttribute("class"); // "info data dark"
// Modificar clases CSS
div.className = "elemento brillo tema-oscuro";
div.setAttribute("class", "elemento brillo tema-oscuro");

<div id="page" class="info data dark" data-number="5"></div>
const div = document.querySelector("#page");
div.classList; // ["info", "data", "dark"]
div.classList.add("uno", "dos"); // No devuelve nada.
div.classList; // ["info", "data", "dark", "uno", "dos"]
div.classList.remove("uno", "dos"); // No devuelve nada.
div.classList; // ["info", "data", "dark"]
```

HANDLING EVENTS

- Programación JavaScript reactiva basada en eventos
- Se capturan los diferentes tipos de eventos en la página a través de `Event`
`Listeners/Handlers`
- Estos listeners reciben el evento y realizan una acción
- Un Handler/Listener por cada tipo de evento

HANDLING EVENTS

TIPOS DE EVENTOS

<i>Evento</i>	<i>Evento</i>	<i>Manejador de evento</i>
Situar Ratón sobre	mouseover	onMouseOver
Situar Ratón fuera	mouseout	onMouseOut
"Clickar" (Pulsar)	click	onClick
Mover el Ratón	mousemove	onMouseMove
Cambiar el valor	change	onChange
Seleccionar	select	onSelect
Al cargar	load	onLoad
Al salir	unload	onUnload
"Submit" (Enviar formulario)	submit	onSubmit
"Resetear" (Reinicializar formulario)	reset	onReset
Enfocar	focus	onFocus
Abandonar elemento	blur	onBlur
Redimensionar	resize	onResize
Mover	move	onMove
Al ocurrir un error	error	onError
Abortar	abort	onAbort

Referencia a [todos los eventos](#)

HANDLING EVENTS

REGISTRO DE HANDLERS

Varias formas de registrar un Handler para un evento:

- A través de propiedad HTML: p.e. `<button onclick="doSomething()">`
- A través del DOM estableciendo el atributo HTML:
`document.getElementById("btn").onclick=doSomething`
- A través del DOM haciendo uso de
`addEventListener(eventType, function() {})` **y**
`removeEventListener(eventType, function() {})`

HANDLING EVENTS

OBJETO EVENT

Cuando se desencadena un evento, en el Handler recibimos el objeto Evento asociado. A través de la propiedad `target` podemos acceder al elemento del DOM donde se produjo el evento

```
54
55     <script>
56         const password = document.getElementById("password");
57         const form = document.getElementById("form");
58         password.addEventListener("keypress", function (event) {
59             console.log("La contraseña es " + event.target.value)
60         });
```

HANDLING EVENTS

EVITAR EL COMPORTAMIENTO POR DEFECTO

Cuando hacemos un submit de un formulario, podemos romper el flujo y evitar el envío del mismo. Para ello utilizamos la función `preventDefault()` y en algunos casos, `return false`.

EJEMPLO BÚSQUEDA DE PELÍCULAS CON EVENTOS

```
const searchForm = document.getElementById("search-form");
var results = [];
searchForm.addEventListener("submit", function(event){
  event.preventDefault();
  fetch(`http://www.omdbapi.com/?apikey=cc1014ca&s=${document.getElementById("search-input").value}`, {
    headers: {
      Accept: 'application/json'
    },
    method: 'GET'
  })
  .then(res => {
    console.log("Response here")
    return res.json()
  })
  .then(r => {
    results= r.Search;
    console.log("Updating cards");
    updateCards();
  })
  .catch(e => {
    console.error("Error " + e)
  })
  return false;
})
```


SCRIPTED HTTP

Consiste en la realización de peticiones HTTP a través de JavaScript

- A través del atributo `location` del `Window` `Object`
- A través de la función `submit()` para enviar formularios
- A través de AJAX

EJEMPLOS SCRIPTED HTTP

CLIENT-SIDE STORAGE

Consiste en almacenar datos simples o complejos en la memoria del navegador para:

- Mejorar la experiencia de usuario, p.e. guardar preferencias, colores, gustos...
- Almacenar datos de la sesión anterior, p.e. cesta de compra
- Almacenar recursos/datos para que la carga de la página sea mucho más eficiente
- Almacenar recursos/datos para poder utilizar el modo sin conexión

CLIENT-SIDE STORAGE

- Cookies
- Web Storage: local & session
- IndexedDB
- Cache

COOKIES

- Es la forma clásica de almacenar datos sobre sesión de usuario, preferencias, etc.
- Se almacenan en formato clave=valor
- Se crean en servidor con la cabecera `Set-Cookie: <nombre-cookie>=<valor-cookie>`
- Se pueden crear con JavaScript a través del atributo `cookie` del DOM
- Asociadas a un dominio y un path, y se envían en todas las request contra ese dominio/path en la cabecera `Cookie`

COOKIES: ATRIBUTOS

- Expires y Max-Age: sirven para establecer la caducidad de la cookie. Si no se establece, será una cookie de sesión
- HttpOnly y Secure: sirven para evitar el acceso a la misma a través de JavaScript, y para solo enviarla en caso de ser una conexión HTTPS respectivamente

Name	Value	Domain	Path	Expi...	Size	HttpOnly	Secure	Same...	Priority
_gat_gtag_UA_127650363_2	1	.onesaitplatform.com	/	202...	25				Medium
_gid	GA1.2.1199609027.1612436211	.onesaitplatform.com	/	202...	31				Medium
_ga	GA1.2.348022328.1612436211	.onesaitplatform.com	/	202...	29				Medium
JSESSIONID	node01v92kbo7h3ant1vkfgdd02dxqd11437.node0	lab.onesaitplatform.com	/controlpanel	Ses...	52	✓	✓		Medium

COOKIES: EJEMPLOS

▼ Response Headers [view source](#)

Connection: keep-alive

Date: Sat, 06 Feb 2021 11:25:42 GMT

Expires: Thu, 01 Jan 1970 00:00:00 GMT

Location: https://lab.onesaitplatform.com/controlpanel/login

Set-Cookie: JSESSIONID=node09rzpv1zuwka812mf88sjb8h1t11440.node0;Path=/controlpanel;Secure;HttpOnly

Strict-Transport-Security: max-age=31536000

▼ Request Headers [view source](#)

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Encoding: gzip, deflate, br

Accept-Language: es-ES,es;q=0.9

Cache-Control: no-cache

Connection: keep-alive

Cookie: JSESSIONID=node0ch9eoadawlrkifiyxe820c411439.node0; _ga=GA1.2.179506249.1612610579; _gid=GA1.2.711548225.1612610579

Host: lab.onesaitplatform.com

WEB STORAGE

- Cada Web Site tiene su propio Web Storage aislado, no pudiendo acceder al de otros dominios
- Almacenamiento de datos simples: texto, numérico (objetos también pero con `JSON.stringify()`)
- `LocalStorage`: almacenamiento persiste tras cerrar el navegador
- `SessionStorage`: almacenamiento no persiste tras cerrar el navegador (sesión finaliza)

WEB STORAGE: EJEMPLO

```
//Set an authentication object
var auth ={"token"
:"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJwcmVudGkiOiJnYXNfbWVya2V0IiwiaWF0Ij0iJnYXNfbWVya2V0Iiwic2NvcGU0lsib3BlbmkiOiJ0sIm5hbWUiOiJ3JtIn0.2_dB1y73cCvqSwNpqlGS6yvdh-L9oJs0tXaGI6CzXc"};
//store
localStorage.setItem('access_token', JSON.stringify(auth));
sessionStorage.setItem('access_token', JSON.stringify(auth));
//recuperamos el valor y pasamos de nuevo a objeto
var auth = localStorage.getItem('access_token')
auth = JSON.parse(auth);
```

INDEXEDDB

- Sistema de base de datos completo embebido en el navegador
- Se utiliza para almacenar datos complejos y pesados como imágenes, vídeos...
- Su uso no es trivial

Ver ejemplo de clase y [referencia](#)

INDEXEDDB

- Interfaz disponible para almacenar los pares de objetos Request Response
- Tantas cachés como queramos, por nombre
- Tenemos que implementar la lógica a través de un [ServiceWorker del navegador](#)

Ver ejemplo de clase y [referencia](#)

HTML5 APIS

En la especificación HTML5 se introducen **numerosas APIs** de las cuales destacamos las más interesantes:

- **Drag and Drop**
- Geolocalización
- Web Storage: Indexed DB & Local Storage
- Web Workers: scripts en segundo plano
- **Canvas**
- Bluetooth
- Camera

FORMULARIOS WEB

- Estructura de un formulario web
- Ajax

ESTRUCTURA DE UN FORMULARIO WEB

- `<form>`: es el bloque que define y agrega todos los elementos del formulario
- `<fieldset>` bloque que agrupa elementos del formulario con el mismo propósito
- `<legend>` da un significado a una agrupación `<fieldset>`

ESTRUCTURA DE UN FORMULARIO WEB

- `<label>` define una etiqueta para un control del formulario. Relación a través del atributo `for`
- `<input>` controles del Formulario (inputs): text, password, email, date, checkbox, hidden...

EJEMPLO DE FORMULARIO WEB BÁSICO

```
1 <html>
2 <body>
3 <div class="form">
4 <form>
5 <fieldset>
6 <div class="form-group">
7 <label class="form-field" for="user">Usuario</label>
8 </div>
9 <div class="form-group">
10 <input class="form-field" id="user" type="input"/>
11 </div>
12 <div class="form-group">
13 <label class="form-field" for="password">Password</label>
14 </div>
15 <div class="form-group">
16 <input class="form-field" id="password" type="password"/>
17 </div>
18 <div class="form-group">
19 <input type="submit" class="login-btn" value="Login"/>
20 </div>
21 </fieldset>
22 </form>
23 </div>
24 </body>
25 </html>
```


VALIDACIÓN DE FORMULARIOS

- Validación client-side: formato correcto
- Protección ante uso malicioso (en cierta medida)
- UX

TIPOS DE VALIDACIÓN DE FORMULARIOS

- Validación HTML incorporada: mejor rendimiento, menos personalización.
- Validación JavaScript: peor rendimiento, mayor personalización

VALIDACIÓN HTML

Atributos de validación

- `required`
- `minlength, maxlength`
- `min, max`
- `type`
- `pattern`

EJEMPLO DE VALIDACIÓN HTML

```
<form id="form">
  <fieldset>
    <div class="form-group">
      <label class="form-field" for="user">Usuario</label>
    </div>
    <div class="form-group">
      <input required minlength="5" class="form-field" id="user" type="input"/>
    </div>
    <div class="form-group">
      <label class="form-field" for="password">Password</label>
    </div>
    <div class="form-group">
      <input required title="Must contain at least: one uppercase letter, one lowercase letter and a number or symbol. Password must be between 10 and 20 characters."
        pattern="(?!^[0,128}$)((?=.*\d)|(?=.*\W+))(?![\.\n])(?=.*[A-Z])(?=.*[a-z]).*$" class="form-field" id="password" type="password"/>
    </div>
    <div class="form-group">
      <input type="submit" class="login-btn" value="Login"/>
    </div>
  </fieldset>
</form>
```

Usuario

Password



Utiliza un formato que coincida con el solicitado

Must contain at least: one uppercase letter, one lowercase letter and a number or symbol. Password must be between 10 and 20 characters.

VALIDACIÓN JAVASCRIPT

API de validación de restricciones

- Disponible para elementos: input, select, textarea, output, fieldset
- Se utilizan los mismos atributos de validación: required, type, pattern...
- Estos elementos obtienen los atributos `validity` (`ValidityState`), `validationMessage`, `willValidate`
- Permite utilizar los métodos `checkValidity()` y `setCustomValidity(string)`

EJEMPLO VALIDACIÓN JAVASCRIPT

```
<script>
  const user = document.getElementById("user");
  const password = document.getElementById("password");
  user.addEventListener("input", function (event) {
    if (user.validity.tooShort) {
      user.setCustomValidity("Introduzca un nombre de usuario con al menos 5 caracteres");
    } else {
      user.setCustomValidity("");
    }
  });
  password.addEventListener("input", function (event) {
    if (password.validity.patternMismatch) {
      password.setCustomValidity("Introduzca una contraseña de entre 10 y 20 caracteres alfanuméricos.");
    } else {
      password.setCustomValidity("");
    }
  });
</script>
```

ENVÍO DE FORMULARIOS

- Destino del formulario (URL): atributo `action`
- Tipo de envío (método HTTP): atributo `method` con valor `GET` / `POST`
- Atributo `enctype` para enviar el contenido con diferente codificación, por ejemplo `multipart/form-data` para imágenes

EJEMPLO DE FORMULARIO DE LOGIN

```
<form id="form" action="/login" method="post">
  <fieldset>
    <div class="form-group">
      <label class="form-field" for="user">Usuario</label>
    </div>
    <div class="form-group">
      <input required minlength="5" class="form-field" id="user" type="input"/>
    </div>
    <div class="form-group">
      <label class="form-field" for="password">Password</label>
    </div>
    <div class="form-group">
      <input required title="Must contain at least: one uppercase letter, one lowercase letter and a number or symbol. Password must be between 10 and 20 characters."
        pattern="(?!.{10,128}$)((?=.*\d)|(?=.*\W+))(?![\.\n])(?=.*[A-Z])(?=.*[a-z]).*" class="form-field" id="password" type="password"/>
    </div>
    <div class="form-group">
      <input type="submit" class="login-btn" value="Login"/>
    </div>
  </fieldset>
</form>
```


EJEMPLO DE FORMULARIO CON IMAGEN

```
<form id="form" action="/alta-dni" method="post" enctype="multipart/form-data">
  <fieldset>
    <label>Número DNI</label>
    <input type="text" id="dni-text" placeholder="Número DNI"/>
    <label>DNI escaneado</label>
    <input type="file" id="dni"/>
  </fieldset>
</form>
```

ENVÍO DE FORMULARIO CON JAVASCRIPT

Es posible enviar un formulario con JavaScript. Tan solo debemos hacer uso de la función JavaScript nativa `submit()` sobre el formulario

```
function submitForm (){  
    document.login.submit();  
}
```

ASYNCHRONOUS JAVASCRIPT (AJAX)

AJAX nos permite realizar llamadas HTTP con JavaScript de manera asíncrona (no bloqueante) en segundo plano, aunque también se puede usar de manera síncrona.

Se suele usar para recargar elementos del DOM que contienen datos de servidores externos, mandar y actualizar datos...

- Se basa en el uso de objetos `XMLHttpRequest`
- Se puede usar cualquier tipo de método HTTP: GET, POST, PUT, DELETE
- La respuesta se procesa en una función del callback, la cual es ejecutada por el navegador en cuanto se recibe respuesta del servidor

PLAIN AJAX (VANILLA)

- Uso del objeto `XMLHttpRequest` con propiedades:
`onreadystatechange,readyState,status,statusText, responseText`
- `readyState` contiene el estado en el que se encuentra la petición, va del 0 al 4, siendo este último el estado en el que se ha completado y podemos leer la respuesta
- Podemos asignar al atributo `onreadystatechange` una función de callback para saber cuando cambiar el estado de la petición
- En el `status` podemos leer el código de la respuesta HTTP
- `statusText, responseText` contienen el texto asociado al código de respuesta y a la propia respuesta respectivamente, por ejemplo `statusText="Not Found" responseText={"data":"json"}`

AJAX READystate

Value	State	Description
0	UNSENT	Client has been created. <code>open ()</code> not called yet.
1	OPENED	<code>open ()</code> has been called.
2	HEADERS_RECEIVED	<code>send ()</code> has been called, and headers and status are available.
3	LOADING	Downloading; <code>responseText</code> holds partial data.
4	DONE	The operation is complete.

EJEMPLO AJAX (VANILLA)

```
//objeto XMLHttpRequest
var request = new XMLHttpRequest();
//funcion de callback asignada al cambio del estado de la petición
request.onreadystatechange = function() {
  //estado == 4 quiere decir que se ha completado
  if(request.readyState === 4) {
    //comprobamos el código de la respuesta HTTP
    if(request.status === 200) {
      document.getElementById("response-div").innerHTML = "Response from the server: " + request.responseText;
    } else {
      document.getElementById("error-div").innerHTML = 'An error occurred during your request: ' + request.status + ' ' + request.statusText;
    }
  }
}
//Tipo de método HTTP y URL de la petición AJAX
request.open('GET', '/api/status');
//Lanzamos la petición AJAX
request.send();
|
```

API FETCH

API mucho mas intuitiva y fácil de usar

Se hace uso de la función nativa `fetch()`

Basado en Promises: objetos `Request`,
`Response`

FUNCIÓN FETCH

- Acepta dos parámetros: la URL del recurso y un objeto `Request` donde se indican: método HTTP, cabeceras HTTP, body...
- Devuelve una `Promise` a la que podemos suscribirnos con `.then()` para procesar la respuesta en forma de objeto `Response`
- La respuesta se puede resolver para extraer el contenido utilizando el `mixin` del objeto `Body` de la respuesta

EJEMPLO FETCH GET

```
var movieSearch = {}:  
fetch('http://www.omdbapi.com/?apikey=cc1014ca&s=Star+Wars',  
  {  
    method: 'GET',  
    headers: {  
      'Content-Type': 'application/json',  
      'Accept': 'application/json',  
    }  
  })  
  
//Suscribimos a la promesa Response  
.then(res => {  
  if(res.ok){  
    //Si la respuesta se resolvió bien, procedemos a resolver la promesa Body  
    return res.json()  
  }else{  
    throw res;  
  }  
})  
  
//Suscribimos a la promesa Body  
.then(r => {  
  //Una vez resuelta la última promesa, asignamos el valor de la respuesta a una variable JSON  
  movieSearch=r  
})  
  
//Errores de RED y respuestas K0  
.catch(e => console.log(e))
```

EJEMPLO FETCH POST

```
var movie = {"Title":"Star Wars: Episode IV – A New Hope","Year":"1977","imdbID":"tt0076759","Type":"movie","Poster":"https://m
//POST
function fetchMovieSearch(){
  fetch('http://www.omdbapi.com/?apikey=cc1014ca&s=Star+Wars',
  {
    method: 'POST',
    body: JSON.stringify(movie)
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json',
    }
  })
  //Suscribimos a la promesa Response
  .then(res => {
    if(res.ok){
      //Si la respuesta se resolvió bien, procedemos a resolver la promesa Body
      return res.json()
    }else{
      throw res;
    }
  })
  //Suscribimos a la promesa Body
  .then(r => {
    //Una vez resuelta la última promesa, asignamos el valor de la respuesta a una variable JSON
    movieSearch=r
  })
  //Errores de RED y respuestas KO
  .catch(e => console.log(e))
}
```

EJEMPLO FETCH POST FORMULARIO MULTIPART

```
var formData = new FormData();
var dni = document.querySelector("input[type='file']");

formData.append('dni-text', '000044444N');
formData.append('dni', dni.files[0]);
//POST con imagen form-data
fetch('https://localhost:8085/microservice/alta-dni', {
  method: 'POST',
  headers: {
    'Content-Type': 'multipart/form-data',
    'Accept': 'text/plain',
  },
  body: formData
})
.then(response => response.json())
.then(response => console.log("DNI entry OK"))
.catch(error => console.error('Error:', error))
```

REFERENCIAS

SESIÓN 1:

- <https://www.ecma-international.org/publications/standards/Ecma-262.htm>
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>
- <https://www.oreilly.com/library/view/javascript-the-definitive/9781491952016/>
- <https://eloquentjavascript.net/>
- <https://js.do/>

REFERENCIAS

SESIÓN 2:

- <https://jsfiddle.net/>
- <https://developer.mozilla.org/es/docs/DOM>
- <https://lenguajejs.com/javascript/dom/que-es/>
- https://developer.mozilla.org/es/docs/Learn/JavaScript/Client-side_web_APIs/Client-side_storage
- https://developer.mozilla.org/en-US/docs/Web/Events#standard_events
- <https://developer.mozilla.org/es/docs/Learn/HTML>
- <https://www.digitallearning.es/blog/javascript-api-html5/>
- https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX/Getting_Started
- https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fet