PROGRAMACIÓN WEB

INTRODUCCIÓN

BÁSICOS

NETWORKING

INTRODUCCIÓN

SERVER/CLIENT SIDE

- **Server side**: el Código se ejecuta en el servidor. (mayores capacidades computacionales)
- **Client side**: el Código se ejecuta en el cliente (en nuestros ordenadores/móviles que utilizamos). Primero se descarga y después se ejecuta.

BÁSICOS

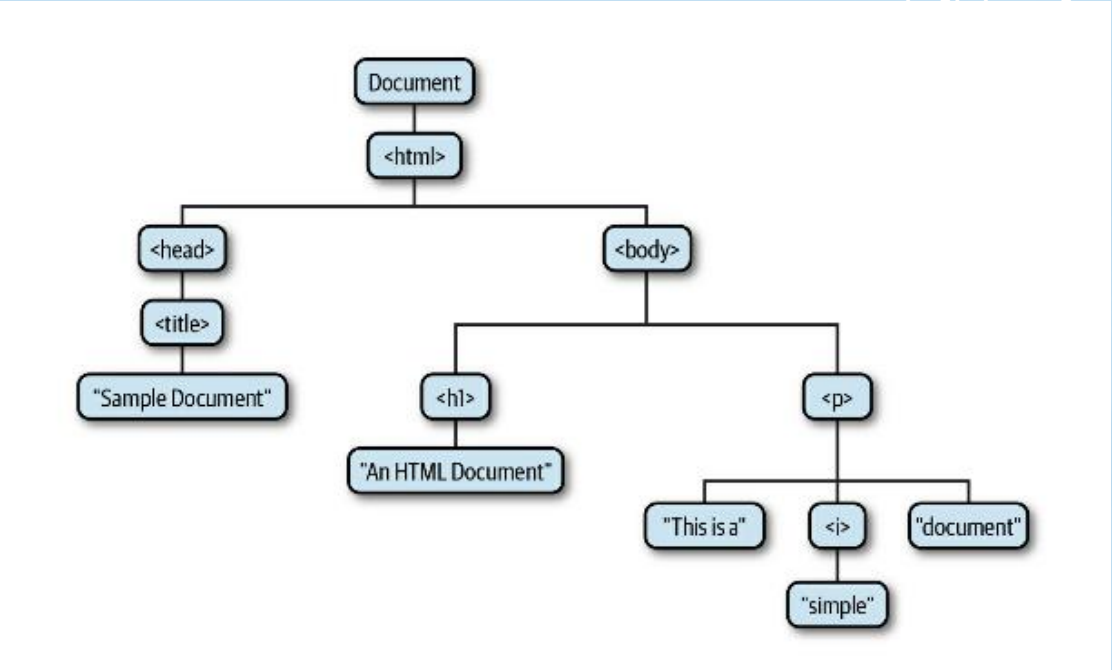
<script>

- Introducir Código **javascript dentro de la página** web
- Añadir la key **src** para descargar el .js asociado (más limpio y ordenado)
- Key **url**, te descarga Código de otros servidores

BÁSICOS

DOM

- **Document Object Model**
- Objeto que representa el archivo html
- Árbol principal



BÁSICOS

Objeto Global

- **Global** a toda la **pestaña/Ventana**
- El Código **javascript corre en este objeto** (exceptuando hilos)
- **Precarga las librerías y módulos y mantiene un histórico**
- En Código, se denomina **window**

BÁSICOS

Namespace

- Podemos **modularizar y privatizar las clases**
- Si no lo precisamos: las variables pueden ser accedidas desde cualquier parte del Código
- **¡Cuidado con conflictos de nombres!**

BÁSICOS

Ejecución de javascript

- Se carga el archivo y todos los scripts comentados se inician.
- El Código **javascript** se utilizará de manera **asíncrona**
- **DOMContentLoaded y load** son los primeros eventos
- JavaScript es **single-threaded**
- Dos eventos **no** se ejecutan a la vez.
- **¡Ojo con bloquear la ejecución!**
- **web worker** (solo para cálculos complejos sin acceso al contenido)

BÁSICOS

Input y Output

- Input:
  - Contenido del document
  - User input: ratón, clicks, teclado…
  - Recepción de información por url
  - Cookies
  - Entradas del propio programa navegador
- Output
  - Manipulación del html para exponer datos (audio, video, logs, texto, pop ups
  - …)

NETWORKING

Introducción

- **Fetch**: ya lo conocemos, es un objeto promise que permite hacer una petición http.
- **SSE (Server-Sent Events):** permiten mantener la conexión abierta para enviar y recibir datos de manera más continuada.
- **Web-sockets**: es un protocol que permite trabajar con HTTP de manera asíncrona

NETWORKING

Fetch

- Acepta por entrada una url
- Crea un objeto respuesta asíncrono con el contenido y el estatus
- Utiliza el cuerpo de la petición para interactuar con él.

```
fetch("/api/users/current")       // Make an HTTP (or HTTPS) GET request
  .then(response => response.json()) // Parse its body as a JSON object
  .then(currentUser => {          // Then process that parsed object
    displayUserInfo(currentUser);
  });
```

Pero también lo podemos hacer con el sistema de async await:

```
async function isServiceReady() {
  let response = await fetch("/api/service/status");
  let body = await response.text();
  return body === "ready";
}
```

NETWORKING

Fetch

- **Query parameters**: parámetros para configurar la petición y variar la respuesta.

```
async function search(term) {
    let url = new URL("/api/search");
    url.searchParams.set("q", term);
    let response = await fetch(url);
    if (!response.ok) throw new Error(response.statusText);
    let resultsArray = await response.json();
```

NETWORKING

Fetch

- **Headers**: cabeceras que se envían al inicio de la petición, tipo de http, tipo de contenido, autenticación...

```
let authHeaders = new Headers();

// Don't use Basic auth unless it is over an HTTPS connection.
authHeaders.set("Authorization", `Basic ${btoa(`${username}:${password}`)}`);
fetch("/api/users/", { headers: authHeaders })
    .then(response => response.json())          // Error handling omitted...
    .then(usersList => displayAllUsers(usersList));
```

NETWORKING

Fetch

- **body**: definir qué tipo de respuesta estamos esperando
  - Text
  - Json
  - arrayBuffer
  - Blob
  - formData

NETWORKING

Fetch

- **body**: definir qué tipo de respuesta estamos esperando
  - text()
  - json()
  - arrayBuffer()
  - blob()
  - formData()

NETWORKING

Fetch

- **Tipos de petición**
  - **GET**: se utiliza solo cuando no hace falta enviar ningún dato, ni se espera que el servidor compute nada.
  - **POST, PUT**: suelen tener contenido a la hora de mandar la petición, el post suele ser para iniciar y el put para actualizar datos en la comunicación (ejemplo document)
  - **DELETE**: para pedir que se borren datos

NETWORKING

Fetch

- **Abortar una petición:** decidimos que hay que desechar la petición

```
// This function is like fetch(), but it adds support for a timeout
// property in the options object and aborts the fetch if it is not complete
// within the number of milliseconds specified by that property.
function fetchWithTimeout(url, options={}) {
  if (options.timeout) {  // If the timeout property exists and is nonzero
    let controller = new AbortController();  // Create a controller
    options.signal = controller.signal;      // Set the signal property
    // Start a timer that will send the abort signal after the specified
    // number of milliseconds have passed. Note that we never cancel
// this timer. Calling abort() after the fetch is complete has
    // no effect.
    setTimeout(() => { controller.abort(); }, options.timeout);
  }
  // Now just perform a normal fetch
  return fetch(url, options);
}
```

NETWORKING

Server-Sent Events

- **Mantener una conexión abierta**
- Chat en una web
- No pasa nada si la conexión se cierra, se puede reabrir.
- **API** de JavaScript

```
let ticker = new EventSource("stockprices.php");
ticker.addEventListener("bid", (event) => {
    displayNewBid(event.data);
}
```

- Buscar en internet más información sobre **EventSource**

NETWORKING

WebSocket

- **Mantener una conexión abierta durante tiempo**
- **API** de JavaScript
- **Cambia la extension (wss://)**
- **El servidor debe estar preparado**

```
let socket = new WebSocket("wss://example.com/stockticker");
```

- Estados de la conexión: CONNECTING, OPEN, CLOSING, CLOSED.
- Envío de mensajes: **send**()
- **¡Creación de protocolo particular!**

BIBLIOGRAFÍA

1. JavaScript: The Definitive Guide, 7th Edition
2. Eloquent JavaScript, 3th edition, Marijn Haverbeke
3. Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)
4. Modern c++ Programming with Test-Driven Development, Jeff Langr
5. Refactoring: Improve the design of existing Code, Martin Fowler
6. Game programming patterns, Robert Nystrom