Repaso LMSGI 2º trimestre

Mientras que en el backend hemos empezado a usar NodeJS y Express, en el front hemos estado trabajando con HTML, CSS y JavaScript Vanilla.

Nos hemos centrado en conocer las diferentes tecnologías de las que disponemos nativamente en el navegador, en concreto `XMLHttpRequest` y `fetch` para hacer peticiones a un servidor, y `JS Vanilla` para manipular el DOM con los datos que recibimos.

XMLHttpRequest

`XMLHttpRequest` nos permite hacer peticiones a un servidor desde JavaScript, fue el primér método que se añadió al navegador para hacer peticiones asíncronas:

```
const xhr = new XMLHttpRequest();
xhr.open('GET', 'https://eazywarez.glitch.me/htmx/hola', true);
xhr.onreadystatechange = function() {
  if (xhr.readyState === 4 && xhr.status === 200) {
    const respuesta = JSON.parse(xhr.responseText);
    document.getElementById('hola_mundo').innerHTML = respuesta;
  }
}
xhr.send();
```

Fetch API

En lugar de `XMLHttpRequest`, podemos usar `fetch` para hacer peticiones a un servidor de forma mas simple:

```
fetch('https://eazywarez.glitch.me/')
  .then(response => response.json())
  .then(data => {
    procesar(data)
  });
```

Mandar headers con fetch

Para interactuar con muchas APIs es necesario configurar un header, aqui podemos especificar cosas como el tipo de contenido que esperamos recibir, el que mandamos u otros datos como tokens de autenticación, API keys, etc.

```
fetch('https://eazywarez.glitch.me/', {
    headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
    'API-Key': '123456'
    }
})
```

Promesas

`fetch` devuelve una promesa, que es un objeto que representa un valor que puede estar disponible ahora, en el futuro o nunca.

```
const promesa = fetch('https://eazywarez.glitch.me/');
const respuesta = promesa.then(response => response.json());
const datos = respuesta.then(data => procesar(data));

function procesar(data) {
  const contenedor = document.getElementById('#contenedorEazy');
    contenedor.innerHTML = data;
}
```

Una promesa tiene dos métodos: `then` y `catch`. El método `then` se ejecuta si la promesa se cumple con `resolve`, y el método `catch` se ejecuta si la promesa no se cumple con `reject`.

```
const promesa = new Promise((resolve, reject) => {
  const valor = Math.random();
    if (valor > 0.5) {
        resolve('La promesa se ha cumplido');
    } else {
        reject('La promesa no se ha cumplido');
    }
});
promesa.then((valor) => {
  console.log(valor);
}).catch((error) => {
  console.log(error);
});
```

`<form>` y `event.preventDefault()`

Si queremos manejar el envio de datos de un formulario, podemos usar el evento `submit` y el método `preventDefault` para evitar que la página se recargue.

```
<form id="formulario">
  <input type="text" name="user" id="user">
  <input type="text" name="password" id="password">
  <button type="submit">Enviar</button>
</form>
```

Usando `event.preventDefault()`:

```
document.getElementById('formulario').addEventListener('submit', function(event) {
  event.preventDefault(); // Evita que la página se recargue
  //capturamos el input que lanzó el evento
  const whoTriggered = event.target;
  const user = document.getElementById('user').value;
  const password = document.getElementById('password').value;
  fetch('/login', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Accept': 'application/json',
      'API-Key': '123456'
    },
    body: JSON.stringify({user, password})
  });
});
```

HTMX

HTMX es una librería que nos permite hacer peticiones a un servidor y actualizar el contenido de la página sin recargarla.

```
<p hx-trigger="load" hx-get="https://eazywarez.glitch.me"></p>
```

Este código hace una petición a `https://eazywarez.glitch.me` cuando la página se carga y actualiza el contenido de la etiqueta `p` con la respuesta.

Este código hace una petición a `https://eazywarez.glitch.me` cuando la página se carga y actualiza el contenido de la etiqueta que tenga el id `contenido` con la respuesta que nos da el servidor.

```
<div hx-trigger="load" hx-get="https://eazywarez.glitch.me" hx-target="#contenido" hx-swap="innerHTML"></div>
<p id="contenido"></p>
```

Manipulación de DOM

Ya hemos visto las principales formas de manipulación del DOM con JavaScript Vanilla, vamos a recordarlas.

Imagina que nuestro servidor nos devuelve un objeto con la siguiente estructura:

```
{
  "nombre": "EazyWarez",
  "descripcion": "Una empresa de software",
  "empleados": [
    {
      "nombre": "Juan",
      "edad": 32
    },
    {
      "nombre": "Maria",
      "edad": 30
    }
  ]
}
```

Manipulación de DOM #1

Manipulación directa

1. Crea tu estructura HTML completa.

```
<div id="contenedor">
  <p id="nombre"></p>
  <p id="descripcion"></p>
  <ul id="empleados"></ul>
</div>
```

2. Captura o crea tus variables con JS.

```
const { nombre, descripcion, empleados } = data;
```

3. Rellena tus contenedores con estas variables.

```
const nombre = document.getElementById('nombre');
const descripcion = document.getElementById('descripcion');
const empleados = document.getElementById('empleados');
nombre.innerHTML = data.nombre;
descripcion.innerHTML = data.descripcion;
data.empleados.forEach((empleado) => {
  const li = document.createElement('li');
  li.innerHTML = `${empleado.nombre} - ${empleado.edad}`;
  empleados.appendChild(li);
});
```

Manipulación de DOM #2

Componentes reusables

Ejemplo en codepen

1. No necesitas estructura en el html, solo un contenedor donde enviarás tu componente.

```
<div id="whooops"></div>
```

En lugar de manipular de forma global, encapsula tu componente en una función a la que le pasas los datos para rellenar los elementos y su contenedor asociado.

Dentro de tu función, puedes usar funciones y métodos de JS para crear y manipular elementos del DOM como `createElement`, `appendChild`, `innerHTML`, etc.

```
function crearTarjeta(titulo, desc, emoji, contenedor){
  const container = document.querySelector(contenedor)
  const card = document.createElement("div")
  card.className = "card"
  container.appendChild(card)
  const template = `
  <h2 class="card__titulo">${titulo}</h2>
  <p class="card__emoji">${emoji}</p>
  <p class="card__desc">${desc}</p>`
  card.innerHTML = template;
}
crearTarjeta("Investiga", "Entiende bien tu problema y que te piden", "🔬", "#whooops")
```

En el ejemplo usamos `template strings`.

Necesitarás su CSS asociado:

```css
.card{
  background: black;
  color: white;
  width: 20vw;
  height: 20vw;
  display: flex;
  flex-direction: column;
  justify-content: space-around;
  align-items: center;
  border-radius: 1rem;
  container-type: inline-size;
  padding: 1rem;
}
.card__titulo{
  font-size: 18cqw;
  font-weight: 600;
}
.card__emoji{
  font-size: 25cqw;
}
.card__desc{
  font-size: 8cqw;
  text-align: center;
}
```