UD3 - Comunicación con el servidor

DAW2 - DWEC

Evolución comunicaciones

Web primigenia.

- Cada interacción requería recargar toda la página.
- Disponía de formularios y peticiones HTTP básicas.

Aparición de AJAX.

- Introducción del objeto XMLHttpRequest.
- Posibilidad de recargar datos sin tener que recargar la página.

Modernización Fetch API.

- Simplifica la comunicación con el servidor y se añade soporte nativo para JSON.

Fetch API

Herramienta moderna para realizar peticiones HTTP.

Basada en promesas => manejo asíncrono de las peticiones.

Gestión de errores y configuración de peticiones mejorados.

```
fetch("https://jsonplaceholder.typicode.com/posts")
    .then((response) => {
        if (!response.ok) {
            throw new Error(`Error HTTP: ${response.status}`);
        }
        return response.json();
    })
    .then((data) => console.log("Datos recibidos:", data))
    .catch((error) => console.error("Error en la petición:", error));
```

Obtener información con Fetch

Por defecto, fetch realiza una petición GET.

```
fetch("https://jsonplaceholder.typicode.com/users")
     .then((response) => response.json())
     .then((usuarios) => console.log(usuarios))
     .catch((error) => console.error("Error:", error));
```

“response.ok” indique que, habiendo recibido respuesta del servidor, la respuesta es correcta.

FIJATE, tenemos dos “.then” seguidos, dos promesas.

Enviar información con Fetch

Para enviar datos al servidor, configuramos la petición con un segundo parámetro.

```
fetch("https://jsonplaceholder.typicode.com/posts", {
    method: "POST",
    headers: {
        "Content-Type": "application/json",
    },
    body: JSON.stringify({ nombre: "Juan", edad: 30 })
})
    .then((response) => response.json())
    .then((data) => console.log("Datos enviados correctamente:", data))
    .catch((error) => console.error("Error al enviar los datos:",
error));
```

WebSockets

Protocolo para comunicaciones bidireccionales en tiempo real entre cliente y servidor.

Permite crear conexiones persistentes bidireccionales.

Necesita un servidor que soporte WebSockets, protocolos “ws://” y “wss://”

Aplicaciones típicas

- Chats en tiempo real.
- Actualizaciones en vivo: noticias, sensores (datos), etc...
- Juegos multijugador en línea.

Ejemplo WebSockets

```
const socket = new WebSocket("wss://echo.websocket.org");

socket.addEventListener("open", () => {
    console.log("Conexión establecida");
    socket.send("Hola, servidor");
});

socket.addEventListener("message", (event) => {
    console.log("Mensaje recibido:", event.data);
});

socket.addEventListener("close", () => {
    console.log("Conexión cerrada");
});
```

Preguntas