

Conexión Front-Backend

Cordero Hernández,
Marco R.



En sesiones pasadas

- Event Listeners
- Objeto *event*
- Event Bubbling



CONTENIDOS

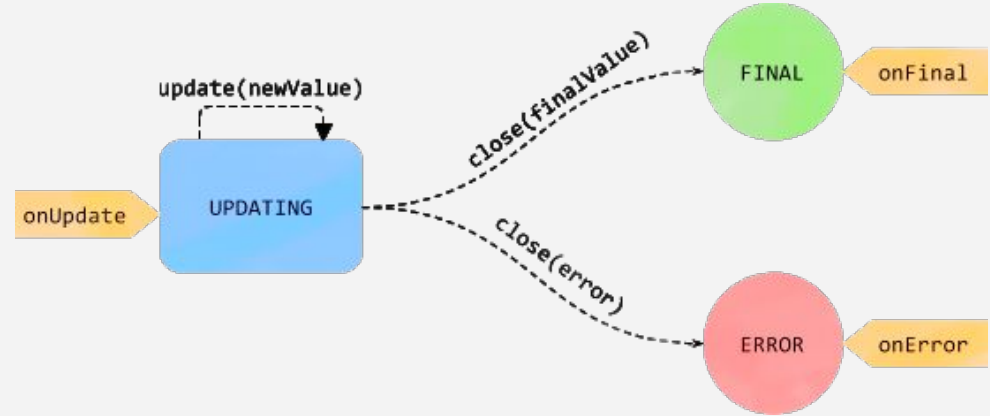
01	02
Repaso breve	Ejercicio integrador

01

REPASO BREVE

¿Recuerdas...

- Programación asíncrona?
- Async/Await?
- *fetch*?
- AJAX?



Async () => { Await }

{ fetch }

¿Recuerdas programación asíncrona?

JavaScript implementa un modelo de operación *asíncrono no bloqueante*, por lo que se pueden realizar peticiones al motor que lo implemente que pueden ejecutarse en otro momento del programa.

Para lograr esto, se puede hacer uso de lo siguiente:

- Función `setTimeout`
- Promesas (`resolve`, `reject`, `then`, `catch`)
- ***Async / Await***



¿Recuerdas *Async / Await*?

Usar la palabra ***async*** antes de una función (en su declaración) habilitará el uso de *promesas* dentro de su declaración sin necesidad de manejarlas directamente. Para ello, se emplea la palabra ***await*** (la cual solo puede ser usada dentro de funciones ***async***) antes de invocar algo que retornará una *promesa*.

Nota: Estas funciones *también regresarán promesas*



¿Recuerdas Async / Await y fetch?

```
async function leerDatosDeJSON(url) {  
    let response = await fetch(url);  
  
    if (response.status == 200) {  
        let json = await response.json();  
        return json;  
    }  
  
    throw new Error(response.status);  
}  
  
const urlJSON = 'https://jsonplaceholder.typicode.com/users';  
leerDatosDeJSON(urlJSON)  
    .then(json => console.log(json))  
    .catch(err => console.log(err));
```


¿Recuerdas AJAX?

Implementar **AJAX** permite:

- Lectura de datos de un *servidor web*
- Actualizar datos de una *página web* sin recargarla
- Enviar datos a un *servidor web* en segundo plano
- Enviar y recibir XML, texto plano o **JSON**

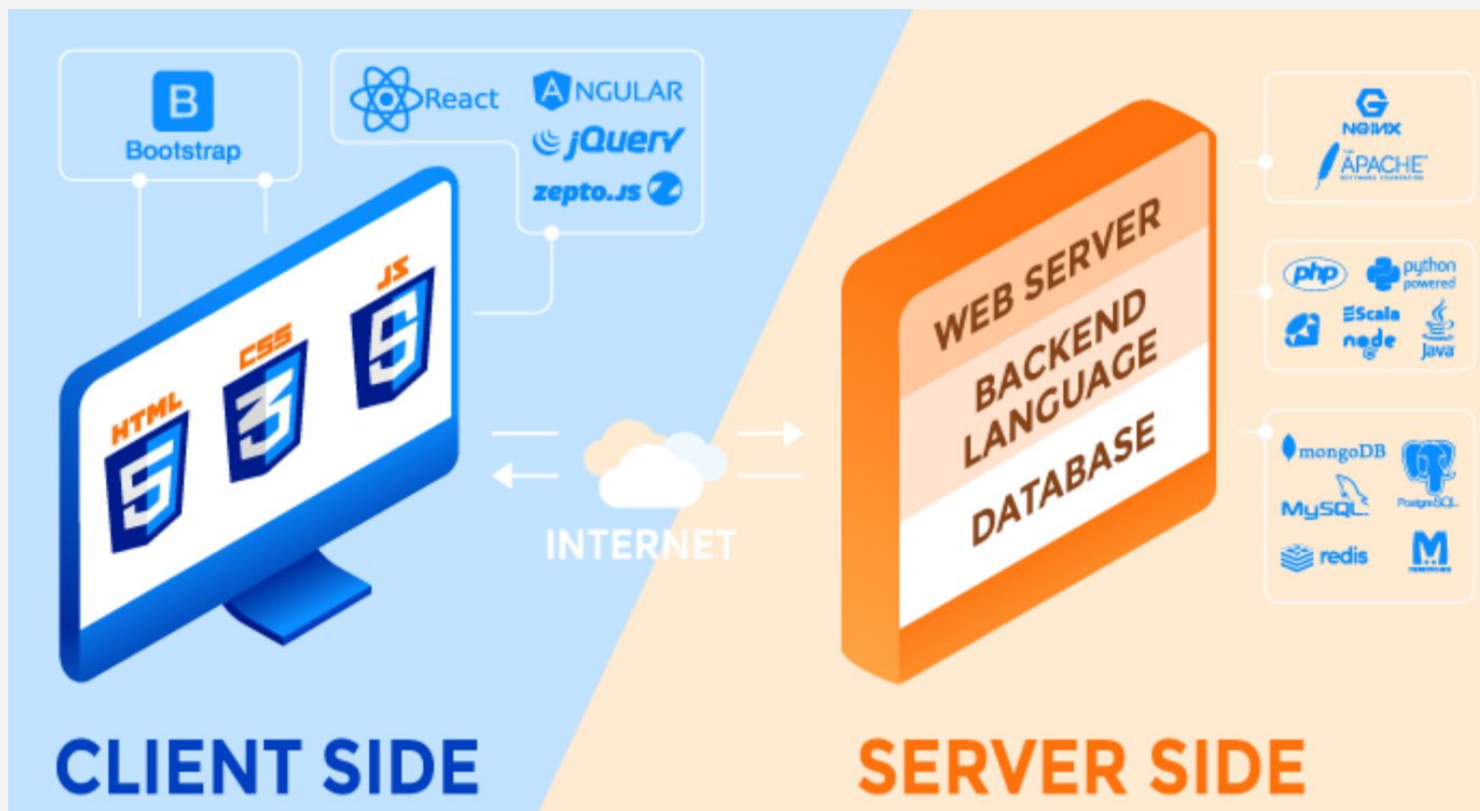
AJAX es una combinación del uso del objeto **XMLHttpRequest** (o la función ***fetch***) y manipulación del **DOM**.



¿Recuerdas AJAX?

```
function guardarEnJSON(datos) {  
    // 1. Crear objeto de tipo XMLHttpRequest  
    let xhr = new XMLHttpRequest();  
  
    // 2. Configurar método (PUT = Actualizar)  
    xhr.open('PUT', urlJSON);  
  
    // 3. Indicar tipo de datos (JSON)  
    xhr.setRequestHeader('Content-Type', 'application/json');  
  
    // 4. Enviar solicitud a la red  
    xhr.send([JSON.stringify(datos)]);  
  
    // 5. Definir comportamiento de recepción de respuesta  
    xhr.onload = () => {  
        if (xhr.status !== 200) { // Fallo de solicitud  
            console.log('Ocurrió un error...');  
            console.log(`${xhr.status}: ${xhr.statusText}`);  
        } else { // Solicitud exitosa  
            console.log(xhr.responseText);  
        }  
    };  
}
```

Conceptos revisados hasta ahora



02

EJERCICIO
INTEGRADOR



Configuración inicial

Dentro del repositorio de la clase encontrarás los archivos base para la implementación presente. El objetivo es replicar el comportamiento de una aplicación web en donde será posible administrar usuarios, esto logrado al realizar la conexión adecuada del *frontend* con el *backend*.

El ejercicio se compone de dos partes:

1. **Frontend:** Debes completar los archivos correspondientes con sus funcionalidades
2. **Backend:** Todo ya está implementado, solo falta instalar los módulos del proyecto ¿Cómo se hacía eso?

Instrucciones – Paso 1

En el archivo ***ajax_handler.js*** implementa el método **loadUser**, el cual realizará una petición al servidor en la ruta ***/api/users*** para cargar los usuarios disponibles en la “base de datos”.

Dentro de ***index.js*** implementa el método **displayUsers** para mostrar el resultado de la petición realizada desde **loadUser**.

Ejecuta el frontend (***home.html***) usando LiveServer para verificar que se haya cargado la información de los usuarios

... ***¿Funcionó?***

★ Instrucciones – Paso 1 – Introducción


En el archivo ***ajax_handler.js*** implementa el método **loadUser**, el cual realizará una petición al servidor en la ruta ***/api/users*** para cargar los usuarios disponibles en la "base de datos".

Dentro de ***index.js*** implementa el método **displayUsers** para mostrar el resultado de la petición realizada desde **loadUser**.

Ejecuta el frontend (desde ***user_admin.html***) usando LiveServer para verificar que se haya cargado la información de los usuarios

... ***¿Funcionó?***

HTTP CORS

▶  XHR GET http://localhost:3000/ CORS Missing Allow Origin

❗ Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://localhost:3000/. (Reason: CORS header 'Access-Control-Allow-Origin' missing). Status code: 200. [\[Learn More\]](#)

TypeError: NetworkError when attempting to fetch resource.

[index.js:7:29](#)

HTTP CORS

El *intercambio de recursos de origen cruzado* (**Cross-Origin Resource Sharing**) involucra peticiones desde un dominio hacia otros. Por defecto y razones de seguridad, los servidores no permitirán la comunicación con servidores alternos al dominio actual, por lo que la obtención de recursos remotos no será posible.

Las políticas prohibitivas de comunicación se aplican tanto a nivel de host como a nivel de puerto (localhost:3000 \neq localhost:5500)



HTTP CORS – Solución

El internet no tendría sentido si estas políticas no pudieran evadirse. Para evitarlas, a nivel de servidor, las peticiones deberán contar con alguna combinación de los siguientes *encabezados*:

- **Access-Control-Allow-Origin** → * (todos los orígenes) o poner una URL de dominio a la cual se le dará acceso a los recursos (debe ser específica)
- **Access-Control-Request-Headers** → Enlista todos los headers especiales que se desea compartir entre orígenes, ejemplo: X-Auth, Accept, Content-Type, ...
- **Access-Control-Allow-Methods** → Enlista todos los métodos permitidos, ejemplo: GET, POST, PUT, ...



★ Instrucciones – Paso 2 – CORS

Ejecuta nuevamente el frontend y corroborar que aún está el error.

Instalar ***cors***

- `npm i cors`

Desde `server.js` importar el paquete instalado y úsalo como *middleware*

- `const cors = require('cors')`
- `app.use(cors())`

Modificar el origen para que solo funcione con localhost en el puerto 5500 (o el que tengas configurado con LiveServer) y *nada más*.

★ Instrucciones – Paso 2 – CORS

```
▶ XHR GET http://localhost:3000/api/users [HTTP/1.1 200 OK 47ms]
▼ Array(3) [ {...}, {...}, {...} ] index.js:6:17
  ▶ 0: Object { _uid: "9701318933", _firstName: "Juan", _lastName: "Perez", ... }
  ▶ 1: Object { _uid: "9901460520", _firstName: "Diego", _lastName: "Lopez", ... }
  ▶ 2: Object { _uid: "8345410339", _firstName: "Diana", _lastName: "Gomez", ... }
      length: 3
  ▶ <prototype>: Array []
```

★ Instrucciones – Paso 3 – Métodos faltantes

Una vez que carguen los usuarios correctamente, completa los métodos faltantes dentro de ***ajax_handler.js***, ***user_utils.js*** e ***index.js***

Verifica que las implementaciones sean correctas llamando *desde consola* la función ***addUser*** y agrega *Ximena* (usuario ya precargado).

Posteriormente, usa los botones de edición y borrado para corroborar que puedas corregir el apellido de *Ximena* y que puedas eliminarla de manera exitosa.

★★ EXTRA

¿Qué cambios serían necesarios para ver la app en un dispositivo móvil?

