Consumir API Rest con Fetch API

En esta sección vamos a hacer uso de la API Fetch en JavaScript para realizar una solicitud HTTP asíncrona a un servidor remoto.

Introducción a la API Fetch

La API Fetch es una herramienta integrada en JavaScript que facilita la realización de peticiones HTTP asíncronas para consumir y manipular datos desde servidores remotos. Esta API se encuentra integrada en los navegadores web modernos y no requiere la importación de librerías externas.

El método global **fetch** proporciona una forma sencilla y lógica de obtener recursos de manera asíncrona a través de la red.

Una petición HTTP es un mensaje que el cliente (por ejemplo, un navegador web) envía a un servidor para solicitar un recurso, como una página web o un archivo. Las peticiones HTTP pueden ser de diferentes tipos, como GET, POST, PUT, DELETE, entre otros, que se utilizan para diferentes propósitos, como consultar, enviar, modificar o eliminar datos.

Un ejemplo sencillo de cómo usar **fetch** para obtener y mostrar datos en formato JSON sería:

fetch('http://example.com/movies.json')

.then(response => response.json)

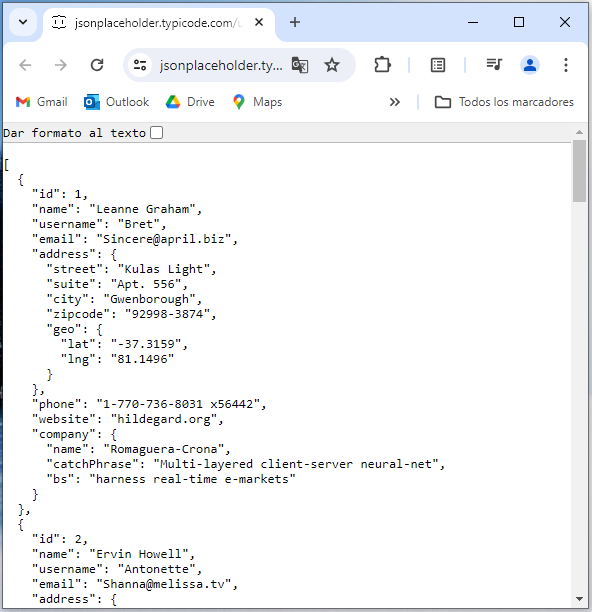
.then(data => console.log(data));

Este código realiza una petición GET a **http://example.com/movies.json**, convierte la respuesta a JSON y finalmente muestra los datos en la consola. Como se trata de una promesa, se maneja con **then**, permitiendo encadenar múltiples operaciones.

Consumir la API de JSON Placeholder con Fetch

JSON Placeholder (<https://jsonplaceholder.typicode.com>) es un servicio gratuito que ofrece múltiples endpoints para practicar y aprender a consumir APIs. Por ejemplo, el endpoint **/users** nos proporciona una lista de usuarios.

Al acceder a la URL: <https://jsonplaceholder.typicode.com/users> podemos ver un código en formato JSON que muestra un arreglo que contiene 10 objetos como elementos. Cada uno de estos objetos representa un usuario, y el valor del atributo **name** de cada objeto muestra el nombre del usuario.

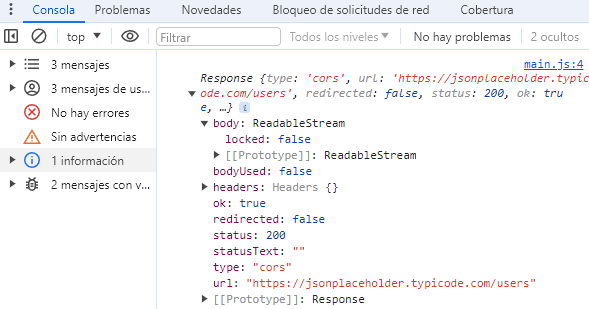


Ten en cuenta que método **fetch** se utiliza para realizar una solicitud HTTP a una URL específica, y toma como primer argumento la ruta URL en forma de cadena de texto (String).

**fetch** devuelve una promesa que eventualmente se resolverá con un objeto **Response** que representa la respuesta de la solicitud HTTP. Esta respuesta también es una promesa, lo que significa que puedes encadenar los métodos **then** y **catch** para manejarla cuando esté disponible.

En este caso, simplemente se imprime la respuesta en la consola.





Como se puede ver en la consola, el objeto response contiene toda la información de la respuesta HTTP, incluyendo el estado (**status**), encabezados (**headers**) y el cuerpo de la respuesta (**body**). Este último es un "**Readable Stream**" que debe ser convertido a JSON o texto para ser utilizado.

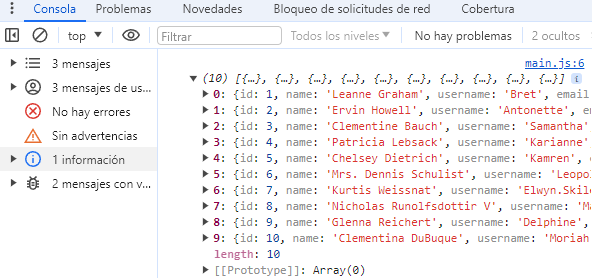
Nota: El estado **200** (**status: 200**) indica que todo está bien.

Mostrar los usuarios en la consola

El método **json** es una promesa que devuelve otra promesa, lo que significa que para manejar la conversión de la respuesta a JSON, se debe anidar otro **then** dentro del primero.

En este caso, el primer **then** maneja la respuesta inicial del **fetch**, mientras que el segundo **then** maneja la conversión de la respuesta en JSON y permite trabajar con los datos de forma efectiva:





Al realizar una solicitud GET a la URL **https://jsonplaceholder.typicode.com/users** con **fetch**, obtenemos una promesa que representa la respuesta HTTP.

Utilizamos el método **then** para manejar esta respuesta. Dentro de este **then**, llamamos al método **json** de la respuesta para convertir el cuerpo de la respuesta a JSON. Como **json** también devuelve una promesa, la anidamos para obtener los datos en formato JSON. Finalmente, mostramos la lista de usuarios en la consola utilizando **console.log(data)**.

Simplificar el código

Podemos simplificar el código eliminando las llaves y encadenando los métodos **then** definididos, ya que cada uno devuelve una promesa. Esto hace que el código sea más limpio y fácil de leer.

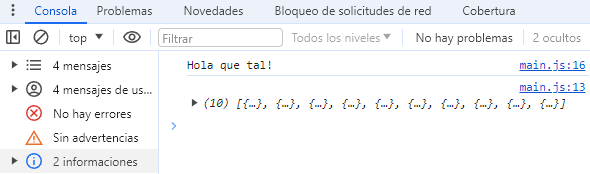
En este ejemplo, la primera llamada a **then** maneja la respuesta inicial y la convierte en JSON utilizando **response.json**. La segunda llamada a **then** maneja los datos JSON y los imprime en la consola.



Demostrar el retraso de la petición

Para demostrar que la petición es asíncrona y tiene un retraso (delay), podemos agregar un mensaje después de la petición:





Podemos ver que el mensaje **"Hola que tal!"** se define primero antes de la promesa, esto indica que hay un delay. Primero se ejecuta de forma lineal, pero debido al delay de la petición asincrónica, se ejecuta después.

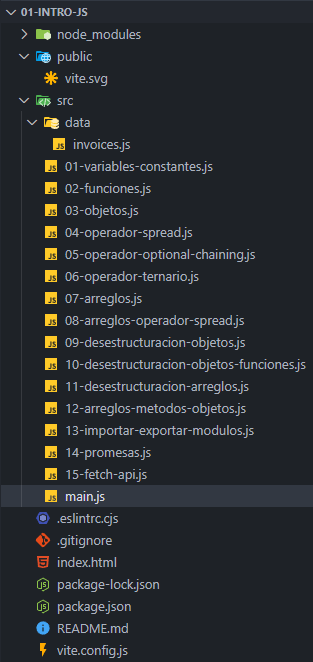
Antes de pasar al siguiente tema

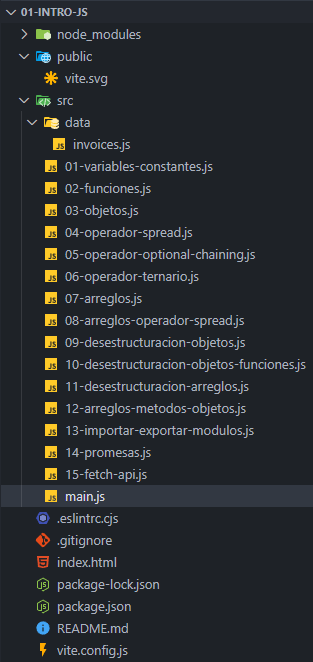
Realiza estas acciones en Visual Studio Code:

1. Abre el panel izquierdo y busca el archivo **main.js**.
2. Haz clic derecho en el archivo y selecciona **Cambiar nombre**.
3. Cambia el nombre del archivo a **15-fetch-api.js** y presiona **Enter**.
4. Ahora, crea un nuevo archivo, dentro la carpeta raíz **src**, haciendo clic en el botón **Nuevo archivo** en la parte superior del panel izquierdo.



1. Nombre el nuevo archivo como **main.js** y presiona **Enter**.





A partir de este momento, se realizarán cambios en el archivo **main.js** que has creado.