TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE:

- Consumo de API Rest con Fetch API.

Application Proggramming Interface o Interfaz de Programación de Aplicaciones, **API** por sus siglas en inglés, es un conjunto de protocolos utilizados para diseñar e integrar el software de las aplicaciones. Las **APIs** permiten interactuar a dos softwares de manera independiente al lenguaje o tecnología en el cual hayan sido diseñados, entregando la información de manera legible tanto para el humano como la máquina, en los formatos **XML** o **JSON**.

Las **API REST** utilizan peticiones de Protocolo de Transferencia de Hipertexto, **HTTP**, para procesar los requerimientos de datos. Existen varios métodos distintos siendo los más importantes **GET**, **POST**, **PUT** y **DELETE**.

- **GET:** Solicita una representación de un recurso especifico. Las peticiones GET solo deben recuperar datos.
- **POST**: Se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
- **PUT:** Reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- **DELETE:** elimina un recurso.

Además, trabajan con códigos de respuestas para indicar si se ha completado satisfactoriamente una solicitud o no. Un ejemplo de estos códigos es el código 400 o “Bad Request” que indica, literalmente, que una solicitud es incorrecta.

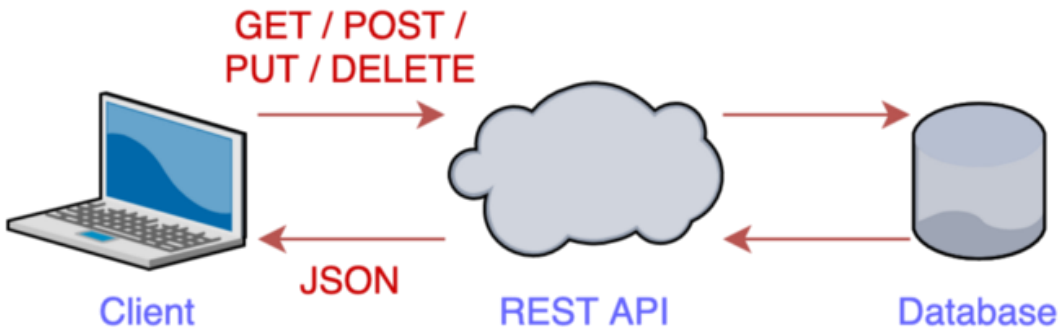

Ilustración 1 Petición HTTP

Fetch API

Cuando trabajamos con **JavaScript** tenemos distintas opciones para consumir una **API REST**, como por ejemplo el uso de **Ajax** con **JQuery**, una de las formas más populares de llevar a cabo esta función, pero existen otras alternativas como **Fetch API** y la Biblioteca **Axios**.

Fetch es una alternativa moderna que nos permite interactuar con **APIs** de una forma sencilla.

Para poder realizar una solicitud, **Fetch** solicita un único parámetro: la **URL** del recurso que queremos consumir.

```
1 Fetch('url')
```

Esta tecnología utiliza las promesas (*promise* en inglés) para ofrecer funciones más flexibles para realizar solicitudes a los servidores desde los navegadores. Además, ofrece una definición genérica de los objetos *Request* y *Response*, proporcionando coherencia, permitiendo su uso donde sea necesario en un futuro.

Ilustración 2 Imagen Fetch

¿QUÉ ES UNA PROMESA?

Una promesa es un objeto que representa la terminación o el fracaso de una operación asíncrona. En otras palabras, representa un valor que puede estar disponible ahora, en el futuro, o nunca.

Como su propio nombre indica, una promesa es algo que, en un principio pensamos que se cumplirá, pero en el futuro pueden ocurrir varias cosas, como encontrarse en un estado incierto, cumplida o no cumplida. Los estados en los que se puede encontrar una promesa son:

- Pendiente (*pending*): estado inicial, no cumplida o rechazada.
- Cumplida (*fulfilled*): significa que la operación se completó satisfactoriamente.
- Rechazada (*rejected*): significa que la operación falló.

Las promesas son objetos devueltos al cuál se adjuntan funciones *callback*, en lugar de pasar *callbacks* a una función.

PERO ¿QUÉ ES UN CALLBACK?

Para explicarlo de manera sencilla, un *callback* es una función que se ejecutará después de que otra función haya terminado de ejecutarse. Son necesarios porque **JavaScript**, al ser un lenguaje orientado a eventos, seguirá ejecutándose mientras escucha eventos, pero en ocasiones requeriremos que primero se ejecute cierta instrucción antes de continuar. Ahí es donde toman relevancia los *calback.*