¿Qué es una API?

Una **API** (Application Programming Interface) es un conjunto de reglas y herramientas que permite que diferentes aplicaciones o sistemas se comuniquen entre sí.

- En términos simples, es como un **camarero en un restaurante**: recibe tus pedidos (solicitudes), los lleva a la cocina (el servidor) y te trae la comida (los datos o resultados).
- Las APIs se utilizan mucho para conectar aplicaciones con servidores o servicios externos, como consultar datos de una base de datos, interactuar con redes sociales, obtener el clima, etc.

Ejemplo común de API:

- Un servicio de clima te proporciona una API para que puedas preguntar cosas como: "¿Cuál es la temperatura actual en mi ciudad?"
  La API devuelve los datos que puedes usar en tu aplicación.

¿Qué es `fetch()` en JavaScript?

`fetch()` es una función nativa de JavaScript que se usa para hacer solicitudes HTTP (como GET, POST, etc.) a APIs o servidores.

Pasos para usar APIs con `fetch()`:

1. **Conocer la URL de la API:**
   - Necesitas la URL donde se encuentra la API.
   - Ejemplo: `https://jsonplaceholder.typicode.com/posts` (una API falsa para pruebas).
2. **Hacer una solicitud HTTP:**
   - Usa `fetch()` para hacer una solicitud GET (obtener datos) o POST (enviar datos).
3. **Procesar la respuesta:**
   - La respuesta de la API se procesa como texto o JSON usando `.json()`.
4. **Manejar errores:**
   - Siempre maneja posibles errores, como problemas de conexión o respuestas inválidas.

---

1. Consultar datos de una API (GET)

Supongamos que queremos obtener una lista de publicaciones desde la API `JSONPlaceholder`.

```
// Función para obtener datos de la API
async function fetchPosts() {
  const apiUrl = "https://jsonplaceholder.typicode.com/posts";

  try {
    const response = await fetch(apiUrl); // Realiza la solicitud
    if (!response.ok) {
      throw new Error('Error en la solicitud');
    }
    const data = await response.json(); // Convierte la respuesta a JSON
    console.log(data); // Muestra los datos en consola

    // Ejemplo: Iterar sobre los datos y mostrar los títulos
    for(post of data) {
      console.log(`Título: ${post.title}`);
    };
  } catch (error) {
    console.error('Hubo un problema con la solicitud:', error);
  }
}

// Llamar a la función
fetchPosts();
```

Visualizar datos en una página HTML

Ahora mostramos los datos de la API en un elemento HTML (`<ul>`).

```
<!DOCTYPE html>
<html>
<head>
  <title>API Demo</title>
</head>
<body>
  <h1>Posts desde la API</h1>
  <ul id="post-list"></ul>

  <script>
    async function fetchAndDisplayPosts() {
      const apiUrl = "https://jsonplaceholder.typicode.com/posts";

      try {
        const response = await fetch(apiUrl); // Solicitud GET
        if (!response.ok) {
          throw new Error("Error en la solicitud");
        }
        const data = await response.json(); // Convierte la respuesta a JSON
```

```
        // Actualiza la lista en la página
        const list = document.getElementById("post-list");
       for(post of data){
          const listItem = document.createElement("li");
          listItem.textContent = post.title; // Muestra el título del post
          list.appendChild(listItem);
        };
      } catch (error) {
        console.error("Error al obtener los datos:", error);
      }
    }

    // Llama a la función
    fetchAndDisplayPosts();
  </script>
</body>
</html>
```

Explicación de `async` y `await`

1. **`async`:** Declara una función asíncrona que devuelve una promesa. Dentro de esta función puedes usar `await`.
2. **`await`:** Pausa la ejecución de la función hasta que la promesa se resuelva o rechace.