1. ¿Qué es *fetch* en JavaScript?

fetch es una **función** que sirve para **pedir datos a un servidor** (por ejemplo, a una API o a una base de datos online).

- Es como decirle al navegador: "Oye, ve a esta dirección (URL), trae lo que haya allí y avísame cuando termine".

Ejemplo mental:

- Tú (el navegador) pides una pizza (datos)
- El repartidor (internet) tarda un poco
- Cuando llega, te avisan

Ese "tarda un poco" es clave.

2. El problema: no es inmediato (asincronía)

Cuando usas fetch, **los datos NO llegan al instante**. El código **no se queda esperando**, sigue ejecutándose. Por eso fetch **no devuelve los datos directamente**, sino una Promesa.

3. ¿Qué es una Promesa?

Una **Promesa (Promise)** es un objeto que representa algo que:

- **Aún no ha terminado**
- **Puede salir bien**
- **O puede salir mal**

Piensa en una promesa como un **"ya te avisaré"**.

- Ejemplo real: "Te prometo que mañana te digo la nota del examen"

En programación:

- Mientras no llega → *pendiente*
- Si todo va bien → *resuelta*
- Si algo falla → *rechazada*

Importante: fetch() **DEVUELVE SIEMPRE UNA PROMESA**

```
const resultado = fetch("https://api.ejemplo.com/datos");
```

resultado **NO son los datos**, es una promesa de que llegarán.

4. ¿Cómo trabajamos con una Promesa?

Hay dos formas habituales:

- **.then()**
- **async / await**

Primero con .then() porque ayuda a entender mejor lo que pasa.

5. Primer .then() → el Response

```
fetch(url)
  .then(response => {
    // aquí tenemos un Response
  });
```

¿Qué es Response? El **Response** es la **respuesta del servidor**, pero **NO son todavía los datos listos**.

Es como:

- El cartero llega
- Te da el paquete
- **Aún no lo has abierto**

El Response contiene:

- Código de estado (200, 404, etc.)
- Cabeceras
- El cuerpo (body) **sin leer**

6. Leer los datos del Response

Normalmente los datos vienen en **JSON**, así que hacemos:

```
fetch(url)
  .then(response => response.json())
  .then(datos => {
    console.log(datos);
  });
```

OJO importante:

- response.json() **también devuelve una Promesa**

Por eso necesitamos **otro .then()**.

Resumen mental:

1. fetch → promesa
2. llega el Response
3. leer el JSON → otra promesa
4. llegan los datos reales

7. Mismo ejemplo con async / await (más claro)

```
async function obtenerDatos() {
  const response = await fetch(url);
  const datos = await response.json();
  console.log(datos);
}
```

Lee esto como si fuera español:

- "**Espera** a que el fetch termine"
- "**Espera** a que se lean los datos"
- "Ahora ya tengo los datos"

await **solo se puede usar dentro de funciones async**.

Ejercicios básicos:

Ejercicio 1: Entender qué devuelve fetch

- fetch **no devuelve los datos directamente**.

```
const resultado = fetch("https://jsonplaceholder.typicode.com/posts");
console.log(resultado);
```

Preguntas:

1. ¿Qué se muestra por consola?
2. ¿Son los datos?
3. ¿Por qué pone Promise?

Respuesta: fetch devuelve una **Promesa**, no los datos.

Ejercicio 2: Primer .then() (Response)

- Entender qué es un Response.

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(response => {
    console.log(response);
  });
```

Preguntas:

4. ¿Qué tipo de objeto es response?
5. ¿Aparecen ya los datos?
6. ¿Ves el status?

Idea clave: Aquí llega la **respuesta del servidor**, no los datos finales.

Ejercicio 3: Leer el JSON

- Ver que response.json() devuelve otra promesa.

```
fetch("https://jsonplaceholder.typicode.com/posts")
  .then(response => response.json())
  .then(datos => {
    console.log(datos);
  });
```

Preguntas:

1. ¿Qué tipo de dato es datos?
2. ¿Cuántos elementos tiene?
3. ¿Por qué hay dos .then()?

Idea clave: Leer el cuerpo de la respuesta **también lleva tiempo**.

Ejercicio 4: Mostrar solo un dato

- Trabajar con los datos recibidos.

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
                .then(response => response.json())
                .then(post => {
                  console.log(post.title);
                });
```

Preguntas:

1. ¿Qué se muestra por consola?
2. ¿Qué otras propiedades tiene post?

Ejercicio 5: Reescribir con async / await

- Comparar estilos (then vs async / await)

```
async function obtenerPost() {
  const response = await fetch("https://jsonplaceholder.typicode.com/posts/1");
  const post = await response.json();
  console.log(post.title);
}

    obtenerPost();
```

Preguntas:

1. ¿Dónde está la promesa?
2. ¿Qué línea "espera" a que lleguen los datos?
3. ¿Cuál versión te parece más clara?

Ejercicio 6: Manejar errores (nivel básico)

- Introducir errores sin entrar muy profundo.

```
fetch("https://jsonplaceholder.typicode.com/URL_INVENTADA")
  .then(response => {
    if (!response.ok) {
      throw new Error("Error en la petición");
    }
    return response.json();
  })
  .then(datos => {
    console.log(datos);
  })
  .catch(error => {
    console.log("Ha ocurrido un error:", error.message);
  });
```

Preguntas:

1. ¿Entra en el catch?
2. ¿Por qué?
3. ¿Qué significa response.ok?