Introducción a la Asincronía: `fetch` y `async/await`

1. El problema: JavaScript es de "un solo hilo"

JavaScript es *single-threaded*, lo que significa que tiene un solo "cerebro" y solo puede procesar una tarea a la vez. Si este cerebro se detiene a esperar la respuesta de un servidor (que puede tardar segundos), toda la página web se congelaría, impidiendo clics o animaciones.

La Metáfora de la Cafetería

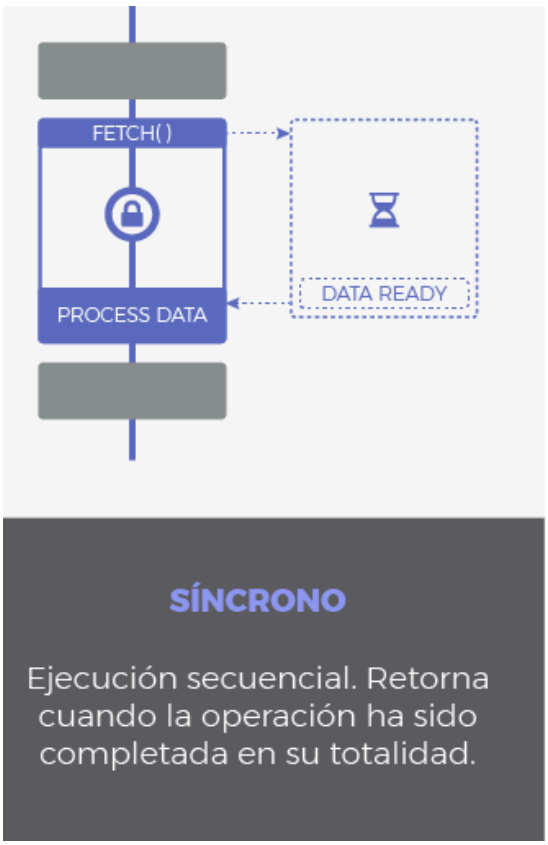
Para evitar este bloqueo, JavaScript usa un modelo **asíncrono no bloqueante**:

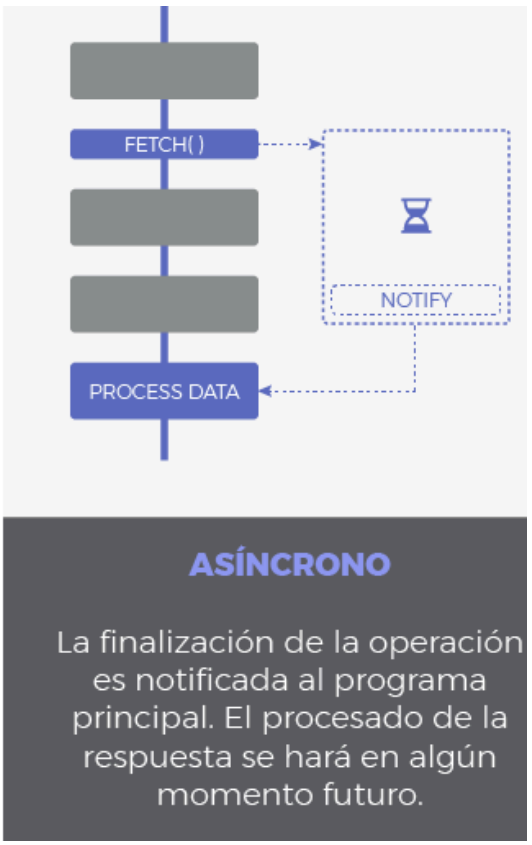
- **Estilo Síncrono (Antiguo):** Un cliente pide un café y el camarero se queda parado frente a la cafetera hasta que termina. Nadie más es atendido; la cola se congela.
- **Estilo Asíncrono (`fetch`):** El cliente pide un café, el camarero le entrega un **Ticket (Promesa)** y continúa atendiendo al siguiente cliente. Cuando el café está listo, se le notifica al cliente para que lo recoja.

2. El "Cartero" de JavaScript: `fetch()`

`fetch` es la herramienta oficial para pedir datos a una dirección URL. Funciona en un proceso de **dos pasos**:

1. **La Petición:** Al ejecutar `fetch(url)`, recibes una **Promesa** (un ticket) que puede estar en tres estados: *Pending* (esperando), *Fulfilled* (éxito) o *Rejected* (error de red).
2. **La Respuesta y el Desempaquetado:** Primero obtienes un objeto `Response` con metadatos (como el estado 200). Para obtener los datos reales (como un JSON), debes usar el método `.json()`, que también es asíncrono.





3. El "Disfraz" Síncrono: `async/await`

Lanzado en ES2017, es "Azúcar Sintáctico" sobre las promesas. No cambia el funcionamiento interno, pero permite escribir código que parece síncrono (línea a línea), facilitando su lectura.

Las piezas clave:

- **`async` (El activador):** Se coloca antes de una función. Indica que la función manejará operaciones lentas y que **siempre devolverá una promesa**.
- **`await` (El freno de mano/Semáforo):** Solo se usa dentro de funciones `async`. Pausa la ejecución de **esa función específica** (sin congelar el navegador) hasta que la promesa se resuelve y entrega el dato ya desempaquetado.

4. Comparativa de Código

Forma Clásica (Promesas anidadas)

El código tiende a identarse hacia la derecha, creando una estructura difícil de seguir (*Callback Hell*):

```
function obtenerDatos() {
  fetch('api/usuario/1')
    .then(res => res.json())
    .then(usuario => {
      return fetch(`api/pedidos/${usuario.id}`);
    })
    .then(pedidos => console.log(pedidos))
    .catch(error => console.error(error));
}
```

Forma Moderna (`async/await`)

El código es plano y se lee como un libro:

```
async function obtenerDatos() {
  try {
    const resUser = await fetch('api/usuario/1'); // Pausa 1
    const usuario = await resUser.json();         // Pausa 2
    const resPedidos = await fetch(`api/pedidos/${usuario.id}`);
    const pedidos = await resPedidos.json();
    console.log(pedidos);
  } catch (error) {
    console.error("Algo falló:", error); // Un solo bloque para todo [cite: 391, 648]
  }
}
```

5. Manejo de Errores: `try / catch`

Con `async/await` se utiliza el bloque estándar `try/catch` de lenguajes como Java o PHP. **Importante:** `fetch` solo salta al `catch` por errores de red (sin internet o URL inexistente). Si el servidor responde con un error 404 o 500, debemos validarlo manualmente usando `if (!respuesta.ok)`.

6. Guía de supervivencia: Errores comunes

Error	Por qué ocurre	Cómo se arregla
Olvidar el async	Intentar usar await en una función normal.	Pon siempre async antes de function.
Olvidar el await	La variable guarda una "Promesa" (el paquete cerrado) en lugar de los datos.	Añade await antes del fetch y del .json().
No usar try/catch	Si la API falla o no hay internet, la web "explota" (error rojo en consola).	Envuelve siempre el código asíncrono en un bloque try/catch.
El .json() es lento	Olvidar que extraer los datos también tarda tiempo.	Usa const datos = await respuesta.json(); (con await).
Bucle forEach	El await dentro de un .forEach() no espera realmente.	Usa un bucle for...of si necesitas que las peticiones vayan en orden.

7. APIs Públicas para practicar

Para empezar a probar `fetch`, puedes usar estas URLs gratuitas:

- **JSONPlaceholder:** `https://jsonplaceholder.typicode.com/posts` (Ideal para blogs).
- **Rick and Morty API:** `https://rickandmortyapi.com/api/character`.
- **Dog API:** `https://dog.ceo/api/breeds/image/random`.
- **DummyJSON:** `https://dummyjson.com/products` (Simulación de tienda).

8. Ejemplo práctico

Revisar carpetas moodle