Alexander Ocoro

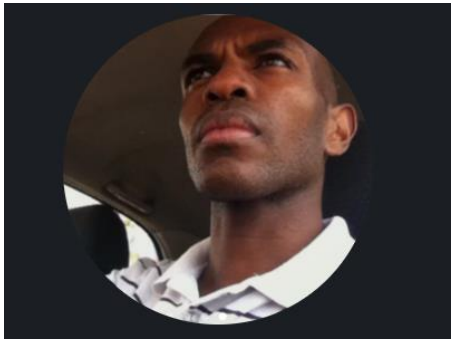
Mentor

TIC

TALENTO TECH VALLE

DESARROLLO *WEB*
FULL STACK
NIVEL BÁSICO

Warner Fdo. Valencia



Ejecutor

JavaScript Avanzado II

Objetivo: Aprender a manejar la programación asíncrona en JavaScript utilizando **Promesas** y **async/await**, y optimizar flujos con `Promise.all`.

Introducción a Promesas

Objetivo: Comprender qué son las Promesas y cómo ayudan a gestionar operaciones asíncronas.

Introducción a Promesas

Teoría: ¿Qué son las Promesas?

1. **¿Qué es la asincronía?**
    - La asincronía permite que el programa continúe ejecutándose mientras espera que una operación (como una llamada a una API o una base de datos) se complete.
2. **Definición de una Promesa**
    - Una Promesa representa el resultado eventual de una operación asíncrona.
    - Tiene tres estados:
        - **Pending (pendiente):** La operación aún no ha terminado.
        - **Fulfilled (resuelta):** La operación se completó con éxito.
        - **Rejected (rechazada):** La operación falló.
3. **Métodos de una Promesa**
    - `then`: Maneja el resultado si la promesa se resuelve.
    - `catch`: Maneja errores si la promesa se rechaza.
    - `finally`: Se ejecuta siempre, independientemente de si la promesa fue resuelta o rechazada.

Ejemplo Práctico: Crear y Manejar una Promesa Simple

1. Crear una promesa que simula una operación asíncrona:

```
const promesa = new Promise((resolve, reject) => {
  const exito = true; // Simula si la operación tiene éxito o
no.
  setTimeout(() => {
    if (exito) {
      resolve("¡Operación exitosa!");
    } else {
      reject("Hubo un error.");
    }
  }, 2000); // Espera 2 segundos
});

promesa
  .then((resultado) => console.log(resultado)) // "¡Operación
exitosa!"
  .catch((error) => console.error(error)) // Maneja el error
  .finally(() => console.log("Operación finalizada."));
```

Uso de Async/Await

Objetivo: Usar la sintaxis de `async/await` para trabajar con promesas de forma más clara y estructurada.

Teoría: ¿Qué son `async` y `await`?

1. `async`
    - Convierte una función en una función asíncrona, que siempre devuelve una promesa.
2. `await`
    - Detiene la ejecución de la función asíncrona hasta que la promesa sea resuelta o rechazada.
3. **Ventajas sobre `.then` / `.catch`**
    - Hace el código más legible y fácil de mantener, especialmente cuando se encadenan múltiples operaciones asíncronas.

Ejemplo Práctico: Convertir Promesas a `async/await`

1. Operación con `.then` / `.catch`:

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then((response) => response.json())
  .then((data) => console.log(data))
  .catch((error) => console.error("Error:", error));
```

2. Conversión a `async/await`:

```
async function obtenerDatos() {
  try {
    const response = await fetch(
      "https://jsonplaceholder.typicode.com/posts/1"
    );
    const data = await response.json();
    console.log(data);
  } catch (error) {
    console.error("Error:", error);
  }
}
obtenerDatos();
```