8.6 Clase en vivo - Profesores

Status	Done

Unidad 8: Manejo del Asincronismo y Errores

Objetivos de clase:

- Entender el asincronismo en JavaScript
- Aprender a manejar peticiones
- Comprender sobre el control de errores

Repaso Unidad 8

Duración: 5 minutos

Consigna: Repaso de lo visto en la Unidad 8

Actividad: Comprendiendo el Call Stack y el Event Loop

Ejemplo para el Profesor:

- Explica la pila de ejecución con el siguiente código:

```
function firstFunction() {
    console.log("First");
}

function secondFunction() {
    console.log("Second");
    setTimeout(() => console.log("Async Action"), 0);
```

```
    console.log("Third");
}

firstFunction();
secondFunction();
```

- Muestra cómo la acción asincrónica es manejada por el Event Loop.

Posible Resolución:

- Salida esperada:

```
First
Second
Third
Async Action
```

Actividad: Implementando `setTimeout` y `setInterval`

Ejemplo para el Profesor:

- Explica cómo funciona `setTimeout` con este código:

```
setTimeout(() => {
    console.log("Mensaje después de 3 segundos");
}, 3000);
```

- Explica cómo detener `setInterval`:

```
const intervalId = setInterval(() => {
    console.log("Mensaje cada segundo");
}, 1000);

setTimeout(() => {
    clearInterval(intervalId);
```

```
    console.log("Intervalo detenido");
}, 5000);
```

Posible Resolución:

- Salida esperada:

```
Mensaje cada segundo
Mensaje cada segundo
Intervalo detenido
```

Actividad: Fetch y manejo de Promesas

Ejemplo para el Profesor:

- Explica cómo hacer una petición con `fetch`:

```
fetch('https://jsonplaceholder.typicode.com/posts')
    .then(response => response.json())
    .then(data => {
        data.forEach(post => console.log(post.title));
    })
    .catch(error => console.error('Error:', error));
```

Posible Resolución:

- Salida esperada: (Ejemplo de varios títulos)

```
sunt aut facere repellat provident occaecati excepturi o
ptio reprehenderit
qui est esse
ea molestias quasi exercitationem repellat qui ipsa sit
aut
```

Actividad: Async/Await

Ejemplo para el Profesor:

- Explica cómo reescribir la función usando `async/await`:

```
const fetchPosts = async () => {
    try {
        const response = await fetch('https://jsonplaceh
older.typicode.com/posts');
        const data = await response.json();
        data.forEach(post => console.log(post.title));
    } catch (error) {
        console.error('Error:', error);
    }
}

fetchPosts();
```

Posible Resolución:

- Salida esperada: Similar a la actividad anterior.

Actividad: Find the Bug - Control de Errores con `try-catch-finally`

Ejemplo para el Profesor:

- Muestra el error común de no usar `await` en `fetch`:

```
const getData = async () => {
    try {
        const response = await fetch('https://jsonplaceh
older.typicode.com/posts');
        const data = await response.json();
        console.log(data);
    } catch (error) {
```

```
        console.error('Algo salió mal', error);
    } finally {
        console.log('Petición finalizada');
    }
}

getData();
```

Posible Resolución:

- Solución:

```
const getData = async () => {
    try {
        const response = await fetch('https://jsonplaceh
older.typicode.com/posts');
        const data = await response.json();
        console.log(data);
    } catch (error) {
        console.error('Algo salió mal', error);
    } finally {
        console.log('Petición finalizada');
    }
}

getData();
```

Preclase - 30 minutos previos

Repaso de Conceptos Clave (10 minutos)

- **Programación Asincrónica:** Breve revisión de los modelos de programación sincrónica y asincrónica, y cómo JavaScript maneja múltiples tareas con el Event Loop.
- **Temporizadores (`setTimeout` y `setInterval`):** Explicación de cómo funcionan estos métodos para programar tareas con retrasos y repeticiones.

- **Manejo de Errores (`try-catch-finally`):** Cómo controlar errores en operaciones asincrónicas, asegurando que el código maneje fallos de manera ordenada.
- **Peticiones con `fetch`:** Introducción al uso de `fetch` para realizar peticiones HTTP, entendiendo cómo maneja las respuestas a través de promesas.
- **Uso de `Async - Await`:** Explicación sobre cómo simplificar el manejo de promesas y mejorar la legibilidad del código asincrónico.

Espacio para Preguntas y Respuestas (15 minutos)

- Abrir el espacio para que los estudiantes hagan preguntas sobre conceptos clave como la gestión de tareas asincrónicas con `async/await` o el manejo de errores con `try-catch`.
- Resolver dudas con ejemplos en vivo, como capturar errores en una petición `fetch` o usar `await` para pausar la ejecución hasta que una promesa se resuelva.
- Si no hay preguntas, sugerir temas comunes como cómo funcionan `setTimeout` y `setInterval` en el contexto del Event Loop, o la diferencia entre `fetch` y otras formas de realizar peticiones HTTP.

Demostraciones Prácticas (Opcional, dependiendo de las preguntas)

- Realizar demostraciones en vivo si las preguntas lo requieren, por ejemplo, cómo realizar una petición `fetch` y procesar la respuesta de forma asincrónica.
- Ejemplos posibles: Mostrar cómo usar `async/await` para manejar peticiones a una API y cómo utilizar `try-catch` para capturar errores en tiempo de ejecución.