



Formato de Tarea

1. Portada

- **Título de la tarea:** Desarrollo de Aplicación Web para el Manejo de Asincronismo y Promesas
- **Asignatura:** Desarrollo web Avanzado
- **Nombre del estudiante:** Anthony Villareal, Ariel Reyes
- **Fecha de entrega:** 08/11/2024
- **Nombre del profesor o docente:** Doris Chicaiza
- **Universidad:** Universidad de las Fuerzas Armadas ESPE

2. Objetivo General

Desarrollar una aplicación web que permita observar y comprender el manejo de asincronismo en JavaScript utilizando diferentes enfoques: Callback, Promesas, y Async/Await, y subir el proyecto a un repositorio en GitHub o GitLab.

3. Desarrollo

- **Conceptos base:** El manejo de asincronismo en JavaScript permite ejecutar funciones sin bloquear el hilo principal, lo cual es esencial en el desarrollo de aplicaciones web dinámicas. Existen diferentes métodos para implementar asincronismo: *Callbacks*, *Promesas*, y *Async/Await*. Estos métodos facilitan el control sobre operaciones que dependen del tiempo de espera, como consultas a bases de datos o acceso a archivos externos. La importancia de estos conceptos radica en optimizar el rendimiento de aplicaciones y mejorar la experiencia de usuario.
- **Herramientas de desarrollo:** En esta tarea, se utiliza Node.js como entorno de desarrollo, ya que permite ejecutar JavaScript en el servidor y facilita el uso de funciones asíncronas. También se emplea Git para el control de versiones y GitHub/GitLab como plataforma de repositorio para compartir el código.
- **Cuerpo o Desarrollo:**
 - **Callback:** En este método, se emplea una función de retorno que se ejecuta tras un retraso de dos segundos. La función `obtenerDatosConCallback` recibe un callback como parámetro, simulando una respuesta asíncronica y notificando cuando los datos se han recibido.
 - **Promesas:** Las promesas ofrecen una alternativa para manejar asincronismo sin la necesidad de callbacks anidados. En `obtenerDatosConPromesa`, se crea una promesa que resuelve exitosamente después de tres segundos. Este método permite una sintaxis más clara mediante el uso de `.then()` y `.catch()`.
 - **Async/Await:** `obtenerDatosConAsyncAwait` emplea la sintaxis de `async/await`, que mejora la legibilidad al permitir un estilo similar al código síncronico. Después de cuatro segundos, la función devuelve el resultado, y se maneja el posible error con un bloque `try/catch`.



- **Codigo Completo:**

```
const fs = require('fs');

function obtenerDatosConCallback(callback) {
  setTimeout(() => {
    console.log("Callback: Datos recibidos correctamente.");
    callback("Callback: Datos recibidos correctamente.");
  }, 2000);
}

function obtenerDatosConPromesa() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log("Promesa: Datos recibidos correctamente.");
      resolve("Promesa: Datos recibidos correctamente.");
    }, 3000);
  });
}

async function obtenerDatosConAsyncAwait() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log("Async/Await: Datos recibidos correctamente.");
      resolve("Async/Await: Datos recibidos correctamente.");
    }, 4000);
  });
}

obtenerDatosConCallback((datos) => {
  console.log("Resultado desde Callback:", datos);
});

obtenerDatosConPromesa()
  .then(datos => {
    console.log("Resultado desde Promesa:", datos);
  })
  .catch(error => {
    console.error("Error en Promesa:", error);
  });

(async () => {
  try {
    const datos = await obtenerDatosConAsyncAwait();
    console.log("Resultado desde Async/Await:", datos);
  } catch (error) {
    console.error("Error en Async/Await:", error);
  }
})();
```



- FrontEnd

Demostración de Asincronismo en JavaScript

1. Usando Callback

Ejecutar Callback

Resultado aparecerá aquí...

2. Usando Promesa

Ejecutar Promesa

Resultado aparecerá aquí...

3. Usando Async/Await

Ejecutar Async/Await

Resultado aparecerá aquí...

4. Conclusión

El uso de *Callback*, *Promesas*, y *Async/Await* ofrece diferentes formas de manejar la asincronía en JavaScript, cada una con ventajas específicas en función del contexto. *Callback* es una forma



básica, mientras que *Promesas* y *Async/Await* proporcionan una mayor legibilidad y manejo de errores. La tarea demuestra la flexibilidad de JavaScript en la gestión de operaciones asíncronas y destaca la importancia de elegir el método adecuado según el caso de uso.

5. Referencias

- Node.js Foundation. (n.d.). *Node.js documentation: Asynchronous programming*. Node.js. Recuperado el 7 de noviembre de 2024, de <https://nodejs.org>
- Mozilla Developer Network (MDN). (n.d.). *Using promises*. MDN Web Docs. Recuperado el 7 de noviembre de 2024, de https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

GitHub.

<https://github.com/ANTHONYNESTORVILLARREALMACIAS/Desarrollo-Web-Avanzado-AnthonyV-ArielR.git>