A logo with blue and white text

Description automatically generated

**UNIVERSIDAD DE LAS FUERZAS ARMADAS “ESPE”**

Departamento de Ciencias de Computación

Ingeniería de Software

**“Consulta JavaScript”**

**Asignatura:**

Programación Web

**NRC: 2260**

**Integrantes:**

Guaman Pulupa Alexander Daniel

**Docente:** Ing. Diego Gambo

**Fecha:** 17 de enero de 2025

Tabla de contenidos

[1. Objetivo 1](#_Toc188006460)

[2. Desarrollo 1](#_Toc188006461)

[2.1 Funciones Callback 1](#_Toc188006462)

[2.2 Función set Interval 1](#_Toc188006463)

[2.3 Funciones set time out 2](#_Toc188006464)

[2.4 Promesas en Javascript 2](#_Toc188006465)

[2.5 Procesos Asíncronos 2](#_Toc188006466)

[3. Conclusiones 2](#_Toc188006467)

# 1. Objetivo

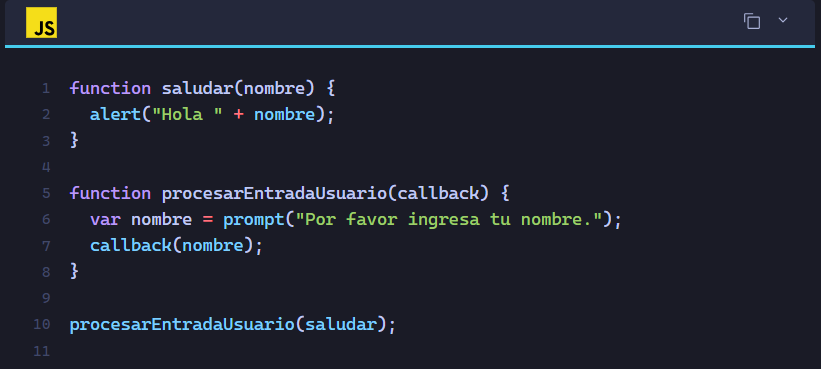
Entender y aplicar los conceptos clave de las funciones callback, temporizadores (setInterval y setTimeout), promesas y procesos asíncronos en JavaScript, con el fin de desarrollar aplicaciones que gestionen tareas asíncronas de manera eficiente y organizada.

# 2. Desarrollo

## 2.1 Funciones Callback

Una función de callback es una función que se pasa a otra función como un argumento, que luego se invoca dentro de la función externa para completar algún tipo de rutina o acción.

Ejemplo:

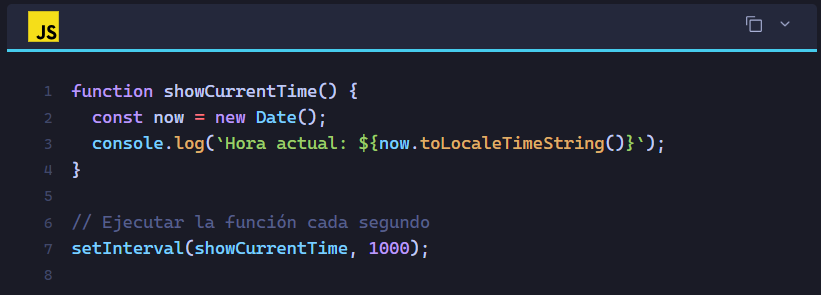


El ejemplo anterior es una callback sincrónica, ya que se ejecuta inmediatamente.

## 2.2 Función set Interval

La función setInterval ejecuta una función repetidamente tras un intervalo definido.

Ejemplo:

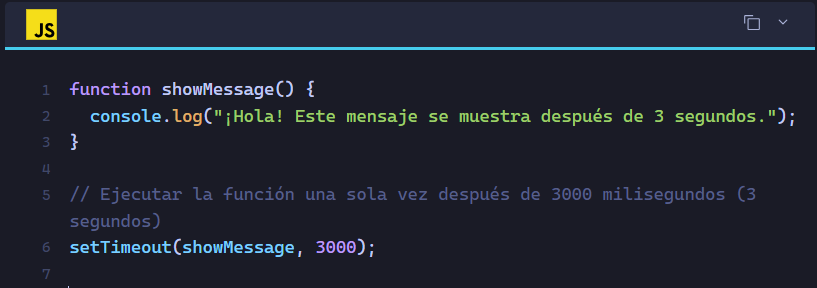


Simula un reloj que imprime la hora actual en tiempo real, actualizándose cada segundo

## 2.3 Funciones set time out

A diferencia de la función setInterval, la función setTimeout ejecuta una función una única vez tras un intervalo definido.

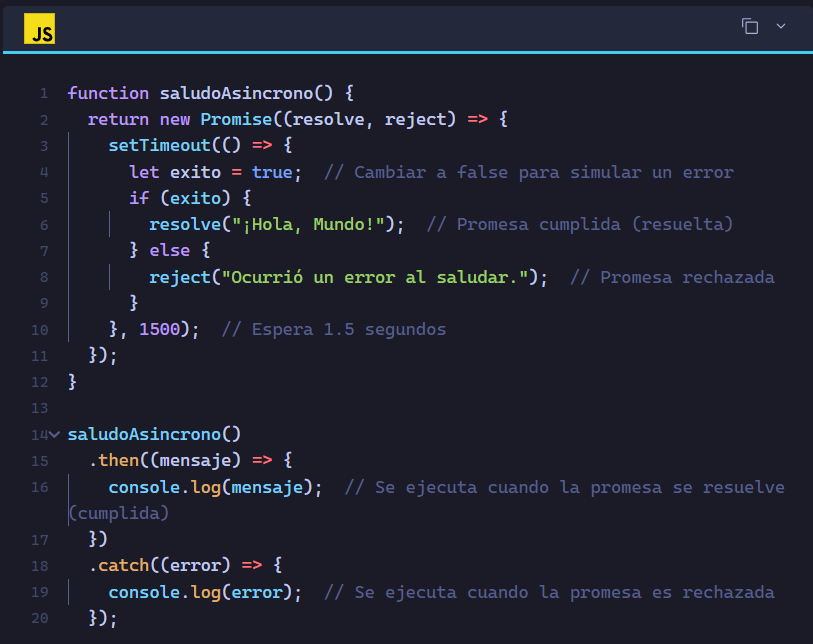
Ejemplo:



Después de 3 segundos, el mensaje se mostrara en la consola

## 2.4 Promesas en Javascript

Las **promesas** permiten manejar operaciones asíncronas de manera más estructurada, evitando. Una promesa representa una operación que puede estar en uno de tres estados: pendiente, cumplida, o rechazada.



**Pendiente**: La promesa está esperando a que se complete la tarea (en este ejemplo, 1.5s).

**Cumplida**: La promesa se resuelve exitosamente con resolve().

**Rechazada**: La promesa se rechaza debido a un error con reject().

Una promesa es un objeto que representa la eventual finalización o fracaso de una operación asíncrona.

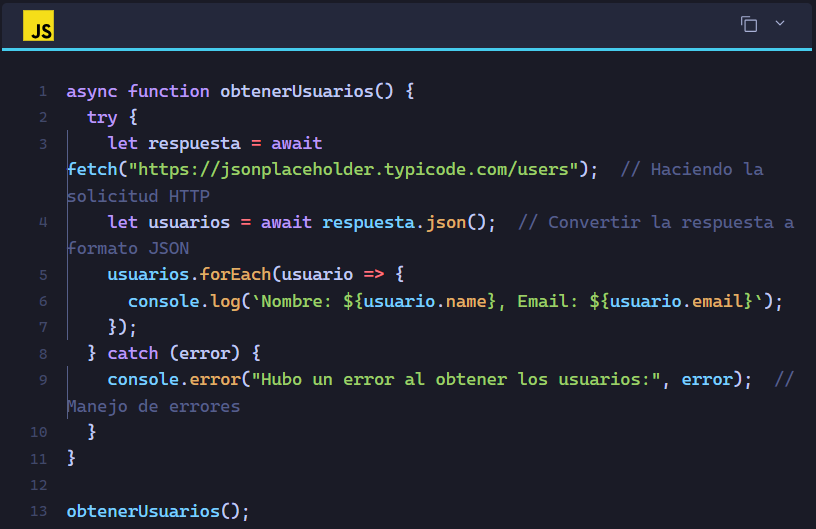
## 2.5 Procesos Asíncronos

Los **procesos asíncronos** en JavaScript se refieren a operaciones que no bloquean el flujo de ejecución del programa mientras esperan que se completen. Esto permite que otras tareas continúen ejecutándose mientras se espera el resultado de una operación (como una consulta a una base de datos, una solicitud HTTP, o la lectura de un archivo). Esto es esencial para mejorar el rendimiento y la eficiencia de las aplicaciones.

## 2.6 Funciones sync/await

**async** convierte una función en una función que devuelve una promesa.

**await** se utiliza dentro de funciones async para pausar la ejecución hasta que la promesa sea resuelta o rechazada



**fetch()**: Realiza una solicitud HTTP GET a la URL "https://jsonplaceholder.typicode.com/users", que es una API pública que devuelve una lista de usuarios.

**await**: Pausa la ejecución hasta que fetch devuelva una respuesta.

**Manejo de errores**: Si hay algún problema con la solicitud o la conversión a JSON, se captura en el bloque catch y se imprime un mensaje de error.

# 3. Conclusiones

Funciones Callback: Permiten ejecutar tareas dentro de otras funciones, especialmente para manejar operaciones asíncronas sin bloquear el flujo del programa.

setInterval y setTimeout: Son útiles para ejecutar funciones repetidamente o después de un retraso, permitiendo gestionar tareas temporizadas en JavaScript.

Promesas: Mejoran el manejo de operaciones asíncronas al representar su estado (pendiente, cumplida o rechazada), haciendo el código más limpio y controlado.

Procesos Asíncronos: Permiten ejecutar tareas sin bloquear el programa, mejorando la eficiencia y la capacidad de respuesta de las aplicaciones.

async/await: Facilitan el trabajo con promesas, haciendo que el código asíncrono sea más legible y manejable, especialmente al trabajar con errores.

**Referencias**

*setInterval() - Referencia de la API Web | MDN*. (n.d.). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/API/Window/setInterval

*setTimeout() - Referencia de la API Web | MDN*. (n.d.). MDN Web Docs. https://developer.mozilla.org/es/docs/Web/API/Window/setTimeout