Prueba Técnica de Revisión de código JavaScript– HICAPPS

Contenido

[1. Revisión del Código 4](#_Toc145718356)

[2. Sugerencias de Mejora 4](#_Toc145718357)

[3. Refactorización 5](#_Toc145718358)

[4. Documentación 6](#_Toc145718359)

**Código Muestra**

**Texto

Descripción generada automáticamente**

1. Revisión del Código:

Revise el fragmento de código proporcionado identificando errores, malas prácticas y áreas de mejora.

**Errores en función fetchData**: Actualmente si ocurre un error de red o cualquier otro error que no sea con estado 200, siempre estará llamando a *callback(new Error("An error occurred while fetching data"))*. Esto puede ocultar información sobre el error que se produce. Sería mejor mostrar información más precisa sobre el error que ocurre.

**Errores en función getUserData**: La función **getUserData** solamente muestra los errores de **fetchData**. Se puede mejorar mostrando un mensaje de error con mayor descripción en lugar de sólo transmitir el error original.

En lugar de *var* se recomienda utilizar **const** al momento de declarar variables. Esto nos ayuda a evitar problemas que se relacionen con el alcance de variables.

En lugar de utilizar *callbacks* para manejar errores, sería mejor considerar el uso de **Promise** para obtener un manejo de errores más moderno, más legible y fácil de mantener.

1. Sugerencias de Mejora:

Proporciona sugerencias concretas para corregir los errores identificados y mejorar el código en términos de eficiencia, legibilidad y mantenibilidad.

**Errores en función fetchData**:

En lugar usar *callback* con un error genérico, se puede entregar una descripción más clara sobre el error que se produce. Esto ayudará en el diagnóstico de problemas.

Usando la propiedad *xhr.statusText* para obtener la descripción con detalles del estado HTTP en caso de ocurrir un error.

**Errores en función getUserData**:

Se puede mejorar el manejo de errores entregando un mensaje de error con una mayor descripción en lugar de solamente mostrar el error original.

**Usar Promise**:

Considera utilizar *Promise* en lugar de *callbacks* para un manejo de errores más limpio y una mejor estructura de código.

**Utilizar const**:

Reemplazar *var* y usar ***const*** para la declaración de variables. Esto ayuda a prevenir problemas de alcance y hace que el código sea más seguro.

1. Refactorización:

Refactoriza el código según tus sugerencias de mejora, asegurando que el código resultante sea funcional, eficiente y fácil de entender.

Texto

Descripción generada automáticamente

1. Documentación:

Prepara una documentación que explique las razones detrás de tus sugerencias y mejoras, destacando cómo estas mejoran la calidad del código.

* **Utilizar *Promise* en lugar de *callbacks***:

**Razones**: *Promise* es una abstracción más legible para manejar asincronía en JavaScript. Otorgan un flujo de control más claro y evitan el "*callback hell*" (anidación excesiva de funciones de *callback*).

**Mejoras**: El código se vuelve más legible y fácil de entender, ya que sigue una estructura lineal en lugar de una anidación de funciones de *callback*.

* **Mejor control de errores**:

**Razones**: Entregar una información detallada sobre los errores ayuda en el diagnóstico de problemas en el código.

**Mejoras**: Cuando ocurre un error, el mensaje de error contiene una descripción relevante, como el estado HTTP en el caso de una solicitud fallida. Esto ayuda a los desarrolladores a identificar problemas de una manera más eficiente.

* **Uso de const en lugar de var**:

**Razones**: *const* tiene un alcance de bloque, lo que significa que son más precisos en cuanto al alcance de las variables y evitan problemas de redeclaración o reasignación no deseados.

**Mejoras**: Al utilizar const, el código se vuelve más seguro y menos propenso a errores relacionados con el alcance de las variables. Además, se fomenta una buena práctica de programación al declarar variables de manera más específica.

* **Estructura más clara y mantenible**:

**Razones**: La refactorización del código siguiendo buenas prácticas, como el uso de *Promise* y la mejora del manejo de errores, hace que el código sea más claro y fácil de mantener.

**Mejoras**: Los desarrolladores pueden leer, entender y mantener el código de manera más efectiva. Esto es esencial para el mantenimiento a largo plazo y la colaboración en proyectos.