DESARROLLO DE SOFTWARE FRONTEND III

MARTINEZ COSIO JOSE ALFREDO

Aarón Hernández García

HttpClient - GET y Manejo de errores

05/08/2025

Investigación: Llamado a una API y Manejo de Errores en Angular

1. Crear un nuevo proyecto Angular

El primer paso es generar un nuevo proyecto utilizando Angular CLI. Este comando inicial crea la estructura base del proyecto, donde se organizan los componentes, servicios, módulos y demás archivos necesarios para el desarrollo.

Una vez creado el proyecto, se recomienda iniciar el servidor de desarrollo para asegurarse de que todo está funcionando correctamente antes de proceder.

2. Hacer el llamado a una API pública

Para fines de prueba y desarrollo, se utiliza una API pública como **JSONPlaceholder**, que permite acceder a datos simulados como usuarios, publicaciones, comentarios, etc.

El objetivo de esta actividad es realizar una **solicitud GET** que recupere datos desde el servidor y luego los muestre en la aplicación Angular, por medio de un servicio y un componente.

3. Manejo de errores HTTP

Es fundamental manejar posibles errores que puedan surgir durante una solicitud HTTP. Si no se controlan adecuadamente, pueden provocar fallos en la aplicación y afectar negativamente la experiencia del usuario.

Los errores HTTP más comunes que se deben contemplar en este contexto son:

- **400 (Bad Request):** La solicitud enviada al servidor es incorrecta o incompleta. Generalmente se debe a errores en los datos proporcionados por el cliente.
- **401 (Unauthorized):** Indica que el usuario no está autenticado. Suele ocurrir cuando se intenta acceder a recursos protegidos sin haber iniciado sesión.
- **403 (Forbidden):** El servidor entiende la solicitud, pero niega el acceso. Es un tema de permisos o restricciones en la cuenta del usuario.
- **500 (Internal Server Error):** El servidor encontró un error inesperado. No es culpa del cliente, y suele requerir que el usuario espere o intente más tarde.
- **504 (Gateway Timeout):** El servidor no respondió a tiempo. Esto puede deberse a una conexión lenta o problemas con el servidor remoto.

4. Personalización de mensajes de error

Para mejorar la usabilidad, es importante que los mensajes de error mostrados al usuario sean claros, personalizados y comprensibles. No es recomendable mostrar mensajes técnicos ni códigos de error directamente.

Ejemplos de mensajes personalizados:

- Para error 400: "Hubo un problema con tu solicitud. Revisa los datos ingresados."
- Para error 401: "Tu sesión ha expirado. Por favor, vuelve a iniciar sesión."
- Para error 403: "No tienes permiso para acceder a esta sección."
- Para error 500: "Ocurrió un error en el servidor. Intenta nuevamente más tarde."
- Para error 504: "El servidor tardó demasiado en responder. Revisa tu conexión a internet."

Este tipo de mensajes permite que el usuario entienda lo que pasó y sepa cómo proceder.

5. Pruebas

Una vez implementado el manejo de errores, se deben realizar pruebas para verificar que:

- Los mensajes se muestran correctamente ante cada tipo de error.
- La aplicación no se detiene o se rompe si ocurre un problema en la comunicación con el servidor.
- La experiencia del usuario sigue siendo fluida, incluso en situaciones no ideales.

Las pruebas pueden realizarse manipulando intencionalmente las URLs para provocar errores, utilizando herramientas como Postman o servicios simulados que permitan probar distintos códigos de estado HTTP.

6. Documentación del código

Toda la lógica debe estar adecuadamente documentada, especialmente en:

- El servicio encargado de hacer la solicitud HTTP.
- La función que detecta el tipo de error y elige el mensaje adecuado.
- El componente que recibe la respuesta o el mensaje de error y lo presenta en pantalla.

Una buena documentación del código facilita su mantenimiento y comprensión por parte de otros desarrolladores.