Reto IA 5

Contexto:

Soy desarrollador frontend junior trabajando en "EcoMarket", un e-commerce para productores locales. Necesito consumir una API REST, en base a mi conocimiento de HTTP y programación.

Decisiones:

- Utilice fetch nativo en lugar de axios.
- Utilice un servidor de mock local en lugar de uno web.
- Utilice el lenguaje de JavaScript.

Alternativas consideradas:

1- **Axios:** Es el estándar de la industria. Te permite configurar una "instancia base" con la URL de EcoMarket, añadir tokens de seguridad automáticamente y manejar errores en un solo lugar.
2- **MSW (Mock Service Worker):** Es la opción más moderna. Instalas una pequeña librería que intercepta las peticiones fetch en el navegador y devuelve datos falsos. No necesitas un servidor externo; todo vive dentro de tu proyecto de frontend.
3- **JSDoc:** Es el "punto medio". Sigues escribiendo JavaScript puro, pero añades comentarios especiales sobre tus funciones. VS Code leerá esos comentarios y te dará autocompletado y advertencias casi como si fuera TypeScript, sin necesidad de compilar nada.

Consecuencias:

1- EcoMarket crece y necesitas implementar un sistema de "retry" automático para peticiones fallidas por red inestable, además de adjuntar un token de autenticación dinámico a cada una de las 50 llamadas diferentes que hace tu app. Con fetch, te verás obligado a escribir un "wrapper" manual que podrías terminar convirtiendo en una versión mal hecha de Axios.
2- El equipo de EcoMarket crece y ahora tienes tres desarrolladores trabajando en diferentes sistemas operativos o versiones de Ubuntu. Uno de ellos no puede levantar el servidor local por un conflicto de puertos o dependencias de Node.js, lo que detiene el desarrollo del frontend por un problema de infraestructura "fuera de su alcance".

Además, tu entorno local no simula la latencia del mundo real ni los fallos de CORS que verás en el servidor real.

3- La API de EcoMarket evoluciona. Un endpoint que antes devolvía un objeto ahora devuelve un array. Como JavaScript es dinámico, tu cliente HTTP procesará la respuesta sin quejarse hasta que una función de UI intente acceder a una propiedad inexistente, lanzando un TypeError: can not read property 'X' of undefined en el navegador del cliente, y tendrás que rastrear el error a través de tres capas de archivos.

conclusión:

Después de plantearme la opiniones y consecuencias que arrojo la IA , decidí que voy a utilizar axios para el manejo de librerías y errores del código , ya que a pesar de que se engrose el código, a la larga va a ser mas sencillo definir las respuestas a errores que se lleguen a presentar.