Aprendizaje Basado en Proyectos: Mejorando la Web del Hospital con APIs, TypeScript, y Seguridad

Contexto:

En este proyecto, los estudiantes aplicarán las herramientas avanzadas que han aprendido para mejorar y optimizar la **web del hospital**. Utilizarán **Fetch API** o **Axios** para gestionar el consumo de datos de APIs, integrarán **TypeScript** para asegurar el tipado y estructura del código, reforzarán la **seguridad del front-end** y gestionarán los errores de manera eficiente. El objetivo es que la web del hospital funcione de manera fluida, segura y con una estructura bien organizada.

Duración: **4 horas**

Requisitos:

1. **Consumo de APIs usando Fetch API o Axios (2 puntos)**

- Implementa el consumo de datos mediante **Fetch API** o **Axios** para interactuar con la base de datos del hospital:
    - Gestiona los datos de pacientes, citas y doctores mediante solicitudes **GET**, **POST**, **PUT**, y **DELETE**.
    - Muestra los datos obtenidos en la interfaz React de manera dinámica.
    - Maneja correctamente los errores de las peticiones y muestra mensajes claros al usuario si ocurre un problema.

2. **Integración de TypeScript en Componentes Clave (2 puntos)**

- Refactoriza los componentes principales de la aplicación utilizando **TypeScript**:
    - Define correctamente los **tipos de datos**, **props**, y **estados** dentro de los componentes.
    - Usa **TypeScript** para mejorar la estructura y el tipado del código, evitando errores en tiempo de ejecución.
    - Aplica interfaces y clases para modelar correctamente los datos de los usuarios y del sistema.

3. **Mejoras en la Seguridad del Front-End (2 puntos)**

- Implementa medidas de seguridad en la aplicación React:
  - Utiliza **React Router DOM** para proteger rutas y permitir el acceso solo a usuarios autenticados.
  - Protege las peticiones a la API mediante **JWT** para asegurar que solo usuarios con permisos puedan acceder a los datos sensibles.
  - Asegura la validación de formularios para evitar **XSS** y otros ataques comunes.
  - Integra encriptación para proteger la información confidencial antes de enviarla al servidor.

4. **Optimización con Hooks y Manejo de Errores (1 punto)**

- Utiliza **Hooks** como **useState** y **useEffect** para gestionar el estado y los efectos secundarios dentro de la aplicación:
  - Implementa un **Hook personalizado** que maneje la lógica repetitiva de la aplicación, como la autenticación o la gestión de formularios.
  - Asegura que los errores durante las peticiones a la API o en la interfaz se gestionen correctamente, mostrando mensajes al usuario en caso de error.
  - Optimiza el rendimiento de la aplicación utilizando dependencias correctamente en **useEffect** y evitando renderizados innecesarios.

Herramientas a Utilizar:

- **ReactJS** con **TypeScript** para el desarrollo y tipado de los componentes.
- **Fetch API** o **Axios** para el consumo de datos.
- **React Router DOM** y **JWT** para la protección de rutas y autenticación.
- **Hooks (useState, useEffect)** para la gestión de estado y efectos secundarios.

Entrega:

- **Formato de entrega**:
  - Opción 1: Enviar un **enlace al repositorio de GitHub** con el proyecto mejorado, incluyendo las funcionalidades descritas (consumo de APIs, TypeScript, seguridad, y manejo de errores).
  - Opción 2: Entregar un archivo **ZIP comprimido** con el proyecto completo y la integración de las mejoras solicitadas.