Ejercicio Práctico: Integración Completa del Sistema del Hospital con ReactJS

Contexto:

En este ejercicio práctico, los estudiantes aplicarán todo lo aprendido sobre **ReactJS** para crear un **sistema completo** para el **hospital**. Implementarán componentes reutilizables, optimizarán el rendimiento, y manejarán interacciones complejas con datos a través de APIs. Todas las vistas del sistema del hospital (Home, Servicios, Equipo Médico, Citas) deberán integrar **componentes avanzados** y técnicas de optimización de ReactJS.

Duración: **3 horas**

Requisitos:

1. **Implementación de Vistas Complejas con ReactJS (1.5 puntos)**

- Crea y estructura tres vistas principales del sistema del hospital usando componentes avanzados:
  - **Vista Principal (Home)**: Incluye una lista de servicios destacados y una sección con información del hospital.
  - **Vista del Equipo Médico**: Muestra los perfiles de doctores utilizando componentes `DoctorCard` para cada miembro del equipo, permitiendo filtrar por especialidad.
  - **Vista de Citas**: Implementa un formulario para agendar citas con **validaciones** y uso de **Hooks** (`useState`, `useEffect`).

2. **Optimización del DOM Virtual y Uso de Fragmentos (1 punto)**

- Usa el **DOM Virtual** para gestionar eficientemente la actualización de datos en las diferentes vistas, asegurando que solo los elementos necesarios se actualicen.
- Implementa **Fragmentos** (`<React.Fragment>`) para evitar añadir nodos innecesarios en el DOM y mejorar la estructura del código en las diferentes secciones del sistema.

3. **Uso de Referencias y Callbacks (1.5 puntos)**

- Implementa **referencias** para interactuar con los elementos del DOM en una de las vistas, como:
    - Enfocar automáticamente en un campo de entrada cuando el usuario ingresa a la vista de **Citas**.
    - Usar referencias de callback para gestionar el desplazamiento a diferentes secciones de la vista **Home**.

4. **Manejo de Datos con API REST Simulada (1.5 puntos)**

- Simula la obtención de datos del equipo médico y servicios a través de una **API REST** utilizando `fetch` y maneja las respuestas de manera asíncrona con **async/await**.
    - Carga los datos en la vista correspondiente (Equipo Médico, Servicios) al montar el componente, utilizando **Hooks** como `useEffect`.

5. **Optimización de Rendimiento y Uso de Profiler (1 punto)**

- Usa **Profiler** para identificar posibles problemas de rendimiento y optimiza la renderización de componentes que manejan grandes volúmenes de datos, como la lista de doctores o servicios.

6. **Comprobación de Tipos con PropTypes (0.5 puntos)**

- Implementa **PropTypes** en todos los componentes para verificar los tipos de datos y asegurar que los valores pasados como props son válidos, evitando errores en la aplicación.

Herramientas a Utilizar:

- **ReactJS** para el manejo del DOM virtual, creación de componentes reutilizables y optimización.
- **Hooks** (`useState`, `useEffect`, etc.) para manejar el estado y los efectos secundarios de los componentes.
- **Referencias** para interactuar con elementos del DOM de manera directa.
- **Profiler** para optimización de rendimiento.
- **PropTypes** para verificar los tipos de datos en los componentes.

Entrega:

- **Formato de entrega**:
    - Opción 1: Enviar un **enlace al repositorio de GitHub** con el proyecto actualizado, incluyendo todas las vistas implementadas, las optimizaciones, y las validaciones de tipos.
    - Opción 2: Entregar un archivo **ZIP comprimido** con el proyecto React, asegurándose de incluir todos los componentes, el manejo de datos, y las optimizaciones realizadas.