Taller: Programación Funcional, Orientada a Eventos y POO en el Proyecto del Hospital

Contexto:

En este taller, los estudiantes deberán aplicar los tres paradigmas de programación en **JavaScript**: **funcional**, **orientada a eventos** y **orientada a objetos**. Estos conceptos se utilizarán para mejorar la funcionalidad y la estructura del **sitio web del hospital**, añadiendo eventos, asincronía, y estructuras basadas en clases para modelar la información del sitio.

Duración: **2 horas**

Requisitos:

1. **Programación Funcional en JavaScript (2 puntos)**

- Implementa una función que use **currying** para calcular el costo total de los servicios de un paciente en función del número de consultas realizadas y el precio de cada consulta.
- Usa la **función flecha** para simplificar la sintaxis en una función que calcule el tiempo promedio de espera de los pacientes.
- Implementa la **recursión** para calcular de forma recursiva el total de horas de consulta de un doctor a lo largo de la semana.
- Integra **composición de funciones** para aplicar descuentos a los costos de consultas en base a la cantidad de consultas realizadas.

2. **Programación Orientada a Eventos y Programación Asíncrona (2.5 puntos)**

- Captura eventos del usuario en la página de **Contacto**:
    - Implementa un **listener** para capturar el envío del formulario y muestra un mensaje de confirmación.
    - Dispara un evento personalizado que simule la llegada de un nuevo paciente, mostrando una notificación en la página.
- Implementa una función **async/await** para simular una llamada a una API REST que obtenga los **datos de los doctores**. Usa **Promise** para manejar los casos de éxito o fallo.

- Implementa el manejo de errores utilizando **try/catch** en las funciones asíncronas, y define un **callback** que se invoque al fallar una petición simulada.

3. **Programación Orientada a Objetos en JavaScript (2.5 puntos)**

- Implementa una **clase** `Doctor` con las propiedades `nombre`, `especialidad`, y `años de experiencia`.
    - Añade un método para mostrar la información de cada doctor y otro para calcular el total de pacientes atendidos por el doctor.
- Crea una subclase de `Doctor`, por ejemplo `Cirujano`, que extienda las funcionalidades de la clase base.
- Implementa el **encapsulamiento** en la clase, protegiendo la propiedad de `años de experiencia` mediante un getter y un setter.
- Usa el **polimorfismo** para sobrescribir un método en la subclase `Cirujano` que calcule el número de operaciones realizadas en lugar de consultas.

Herramientas a Utilizar:

- **JavaScript** para implementar funciones funcionales, eventos, asincronía y clases.
- **README**: Actualiza el archivo README con:
    - Explicaciones sobre la programación funcional aplicada.
    - Descripción de los eventos y el uso de asincronía.
    - Explicación de la implementación de clases y el uso de herencia, encapsulación, y polimorfismo.

Entrega:

- **Formato de entrega**:
    - Opción 1: Enviar un **enlace al repositorio de GitHub** con el proyecto actualizado, incluyendo los archivos HTML, CSS, JS y README.
    - Opción 2: Entregar un archivo **ZIP comprimido** con el proyecto completo (HTML, CSS, JS, README, etc.).