Examen de React

Objetivo:

Crear una app que simule una **luz de semáforo** que cambia de color al presionar botones.

Instrucciones:

El examen consta de dos partes: Teórica y práctica; para la parte práctica se debe crear un contexto en donde se conteste las preguntas y otro contexto para la parte práctica, con base a la siguiente estructura:

```
src/
├── App.tsx
├── main.tsx
├── context/
│   └── SemaforoContext.tsx
├── components/
│   ├── LucesSemaforo.tsx
│   ├── BotonesSemaforo.tsx
│   └── ParteTeorica.tsx
```

- No se permite copiar el código de internet directamente.
- El trabajo es individual.
- Se debe cargar el proyecto en un git y poner el repositorio para descargarlo.
- Cree los componentes separados.
- Use Tailwind CSS para el diseño.
- Use `useState` y `useContext` para manejar el color actual.
- Puede mostrar los círculos del semáforo como `div` con fondo circular (`rounded-full`).

Parte 1: Teoría (4 puntos)

1. ¿Qué hace el hook `useState`? Dé un ejemplo.

`useState` es un hook de React que permite manejar el estado en componentes funcionales. Retorna un valor de estado y una función para actualizarlo.

Ejemplo:
```
const [contador, setContador] = useState(0);
```

2. ¿Qué es un **fragmento** (`<> </>`) y para qué sirve?

Un fragmento es una forma de agrupar varios elementos JSX sin agregar un nodo extra al DOM. Es útil cuando se necesita devolver múltiples elementos desde un componente sin usar un div adicional.

Ejemplo:

```
<>
  <h1>Título</h1>
  <p>Descripción</p>
</>
```

3. ¿Qué diferencia hay entre `useContext` y `useState`?

- `useState` se usa para crear y actualizar estados locales dentro de un componente.
- `useContext` se utiliza para acceder a valores globales (estado o funciones) definidos en un contexto compartido, útil para comunicación entre componentes sin prop drilling.

4. ¿Cuál es la estructura básica de un componente funcional?

```
const MiComponente = () => {
  return (
    <div>
      Contenido del componente
    </div>
  );
};

export default MiComponente;
```

Parte 2: Práctica (6 puntos)

 **Caso práctico: Simulador de Semáforo**

Crea un programa web con 3 botones:

- "Luz Roja"
- "Luz Amarilla"
- "Luz Verde"

Al hacer clic en un botón, debe cambiar la luz activa del semáforo. Solo una luz puede estar encendida a la vez. Las luces deben representarse con círculos grandes (`div`) de colores y estilos de Tailwind.

🔧 **Requisitos técnicos:**

1. Crear un contexto llamado `SemaforoContext` para guardar el color actual.
2. Crear un componente `LucesSemaforo` que muestre los 3 círculos con colores, y solo uno activo.
3. Crear un componente `BotonesSemaforo` con los 3 botones.
4. Usar `useContext` para que los botones cambien el color global.
5. Usar `useState` en el contexto para almacenar el estado del color.
6. Usar fragmentos para no usar `div` innecesarios.
7. Aplicar Tailwind para mostrar claramente qué luz está encendida (por ejemplo: `opacity-100` activa, `opacity-30` apagadas).

LINK GITHUB:

https://github.com/SOQI17/APPS-WEB/tree/525a67f93471f71500fecd79a43440cc0961a2d4/semaforo-appWEB/src
https://github.com/SOQI17/APPS-WEB.git