**Informe de Desarrollo: Aplicación Pokémon Clicker**

**Framework elegido: React**

He elegido **React** para este proyecto debido a sus ventajas al trabajar con interfaces de usuario interactivas y dinámicas. React permite crear aplicaciones web mediante el uso de componentes reutilizables que gestionan su propio estado, lo cual es muy adecuado para el tipo de aplicación de "clicker" que se desea construir, donde el estado (como el puntaje) cambia con frecuencia.

* **Componentes**: Cada parte de la aplicación (como el contador de puntuación, el botón de clic y la lista de Pokémon obtenidos) se implementa como un componente independiente. Esto hace que el código sea más limpio, organizado y reutilizable.
* **Manejo del estado**: React proporciona un manejo eficiente del estado a través de useState. El puntaje y los Pokémon obtenidos se almacenan en el estado de la aplicación, lo que permite que la interfaz se actualice de manera reactiva cuando el usuario interactúa con la aplicación.
* **Rutas**: No se utiliza un sistema de rutas en esta aplicación debido a que es una aplicación de una sola página (SPA), pero React facilita la integración de un sistema de rutas si en algún momento se desea expandir la funcionalidad.

**Estructura de los componentes**:

He dividido la aplicación en varios componentes reutilizables para mantener el código limpio y organizado:

* **App.js**: El componente principal donde se maneja el estado global de la aplicación, como el puntaje y los Pokémon obtenidos. Es el punto de partida que reúne todos los componentes.
* **ClickerButton.js**: Un componente que contiene el botón para incrementar el puntaje. Cuando el usuario hace clic, se ejecuta una función que aumenta el puntaje y, si es necesario, obtiene un nuevo Pokémon.
* **Score.js**: Este componente muestra el puntaje actual del usuario. Se pasa como prop desde el componente principal App.
* **PokemonList.js**: Muestra una lista de los Pokémon que el usuario ha capturado hasta el momento, con imágenes y nombres. También es un componente reutilizable que toma como prop la lista de Pokémon capturados.

**Buenas prácticas de React**:

* Utilización de **props** para pasar información entre componentes.
* Uso de **hooks** como useState y useEffect para gestionar el estado y los efectos secundarios.
* Componentes **funcionales** que son más simples y fáciles de mantener.
* Gestión eficiente del estado para evitar la duplicación de datos en los componentes.

**3. Implementación de CRUD usando Local Storage**

La aplicación implementa la persistencia de datos utilizando el **Local Storage**. El objetivo de esta parte es permitir que los datos (como el puntaje y los Pokémon obtenidos) se conserven incluso después de que el usuario recargue la página. Esto se logra de la siguiente manera:

* **Guardar datos**: Cada vez que se actualiza el puntaje o la lista de Pokémon, estos se guardan en el Local Storage utilizando la función saveToLocal().
* **Leer datos**: Al cargar la aplicación, se lee el puntaje y la lista de Pokémon almacenados previamente usando la función loadFromLocal().
* **Actualizar datos**: Cuando el puntaje cambia o se capturan nuevos Pokémon, se actualizan tanto el estado de React como el Local Storage.

**Código (guardar puntaje y pokémon)**:

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**4. Integración con una API externa usando Fetch**

La aplicación se conecta a la **Pokémon API** (<https://pokeapi.co>) para obtener información sobre los Pokémon cuando el usuario alcanza ciertos puntajes. Usé el método fetch para hacer solicitudes a la API y obtener un Pokémon aleatorio.

* **Obtener Pokémon**: Cuando el usuario alcanza un puntaje que supera un umbral (por ejemplo, cada 10 puntos), la aplicación hace una solicitud a la API para obtener un Pokémon aleatorio.
* **Manejo de errores**: Se maneja el error de la API en caso de que no se pueda obtener un Pokémon. Se asegura de que el código no falle si la API no responde correctamente.

**Código**:

Texto

El contenido generado por IA puede ser incorrecto.

**Conclusión**

El código está comentado de manera que cada función tiene una explicación sobre su propósito y cómo contribuye a la aplicación. Las principales decisiones que tomé fueron:

* Uso de useState para almacenar el puntaje y los Pokémon.
* Uso de useEffect para guardar los datos en el Local Storage y asegurarse de que se mantengan entre recargas de página.
* Creación de una función getRandomPokemon para obtener un Pokémon aleatorio desde una API externa.
* Implementación de un sistema de "buffs" (mejoras) basado en los tipos de Pokémon obtenidos, de forma que cada tipo de Pokémon ofrece un aumento diferente en el puntaje.

La aplicación se presenta como una SPA donde el usuario hace clic en un botón para aumentar el puntaje, y por cada umbral alcanzado, obtiene un Pokémon aleatorio. La interfaz es sencilla y efectiva, con un contador de puntos, un botón de clic y una lista de Pokémon obtenidos.

Además, la aplicación guarda y recupera los datos en Local Storage, por lo que los cambios persisten incluso después de recargar la página. Se incluye un botón de "Resetear" que elimina todos los datos del juego y reinicia la experiencia.

**Características adicionales**:

* Un diseño temático de Pokémon, con colores inspirados en el Centro Pokémon.
* Un mensaje al resetear el juego que dice "Giovanni ha enviado al Team Rocket para quitarte todos tus Pokémon."

La aplicación cumple con todos los requisitos establecidos: la gestión del estado, la conexión con una API externa, la persistencia de datos en el Local Storage y la presentación de una interfaz sencilla y funcional. La aplicación es un ejemplo de cómo React facilita el desarrollo de aplicaciones interactivas, manteniendo el código modular y eficiente.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.