**UNIVERSIDAD NACIONAL DE COSTA RICA**

**SEDE REGIONAL BRUNCA CAMPUS COTO**



**Asignación 2**

**UseContext**

**Estudiantes:**

Jocsan Ramirez Chaves

Abdías Ureña Soto

Sebastián Arias Avilés

**Carrera:** Ingeniería en sistemas de información.

**Profesor:** Juan Gamboa Abarca

**Curso:** Programación IV

**Código de carrera:** BA-INFORM (572201)

**13 abril 2023**

El hook useContext de React es una herramienta poderosa para compartir datos entre componentes sin la necesidad de pasar props de manera explícita. Es importante seguir buenas prácticas de diseño de componentes y tener en cuenta las desventajas y limitaciones del hook al decidir si utilizarlo en una aplicación determinada.

A continuación, se detallan sus ventajas, desventajas y algunas malas prácticas que deben evitarse:

Ventajas:

* Evita pasar props a través de múltiples capas de componentes.
* Permite acceder a datos globales en cualquier parte de la aplicación.
* Simplifica la comunicación entre componentes.
* Reduce la complejidad del código.

Desventajas:

* Aumenta la complejidad del código si se usa en exceso.
* Puede ser difícil de depurar si no se usa adecuadamente.
* A veces puede llevar a patrones de dependencia no deseados entre componentes.
* Puede causar problemas de rendimiento si se utiliza en exceso.

Malas prácticas:

* Usar demasiado useContext en componentes de bajo nivel.
* No seguir el principio de responsabilidad única al usar useContext.
* Crear contextos muy grandes que contienen datos irrelevantes para los componentes.
* Usar useContext para compartir datos que cambian con frecuencia y no se pueden cachear.

React.FC es un tipo genérico en TypeScript que se utiliza para definir componentes funcionales en React. La abreviatura "FC" significa "Functional Component" (Componente Funcional en español) y es un componente que se define como una función.

La ventaja de usar el tipo React.FC es que proporciona un tipo de retorno más preciso para el componente. Por ejemplo, en lugar de tener que especificar manualmente el tipo de retorno como JSX.Element, React.FC ya define el tipo de retorno para el componente como React.ReactElement | null.

Esto significa que cualquier cosa que se devuelva desde el componente debe ser un elemento de React válido o null.

Además, también proporciona un tipo para las propiedades que se pueden pasar al componente como argumento. Por ejemplo, React.FC<props> definiría las propiedades del componente como props: props. React.FC es un tipo que simplifica la definición de componentes funcionales en React y proporciona una verificación de tipos más precisa para las propiedades y el valor de retorno del componente.

En resumen, el hook useContext es una herramienta muy útil que puede mejorar significativamente la calidad del código y la eficiencia del desarrollo en aplicaciones grandes y complejas de React. Sin embargo, su uso debe ser adecuado y controlado para evitar problemas de rendimiento y complejidad.

**Guía**

1. Crear un nuevo proyecto de React usando el comando npx create-next-app my-app –typescript este comando instalara typescript de un solo con nextjs, el my-app puede ser remplazado con el nombre de carpeta que gusten.
2. Crear una carpeta llamada components.
3. Dentro de la carpeta components, crear un archivo llamado TodoForm.tsx será el archivo que como su nombre lo dice es un Formulario de Tareas, también crear un archivo SideBar.tsx el cual será usado para hacer el listado de las tareas y poder cambiar su estado de activo a inactivo.
4. Dentro de la misma carpeta de components, también crear un archivo llamado TodoContext.tsx, el contendrá el contexto y el diseño provider.
5. En TodoForm.tsx, importar los módulos de React y TodoContext.
6. Crear un componente TodoForm con una interfaz React.FC.
7. Dentro del componente TodoForm, importar los hooks useState y useContext de React.
8. Dentro del componente TodoForm, crear un estado local title usando useState.
9. Dentro del componente TodoForm, crear una función handleSubmit que maneje la presentación del formulario de edición y actualización de tareas.
10. Dentro del componente TodoForm, crear una función handleEdit que maneje la edición de tareas existentes. Exportar el componente TodoForm como default.

Enlace al repositorio: https://github.com/Abdias-Urena/Task-List