Introdução ao useContext com TypeScript e React

Por Escola Dnc

Introdução

Este ebook aborda o uso do hook useContext no React com TypeScript, focando na implementação de um tema escuro (Dark Mode) em uma aplicação. Vamos explorar como o useContext pode melhorar a eficiência na passagem de dados entre componentes e como implementá-lo corretamente usando TypeScript.

Contexto e Motivação para o useContext

O React tradicionalmente passa propriedades de componente pai para filho. No entanto, para certas funcionalidades globais, como um tema de aplicação, essa abordagem pode se tornar ineficiente.

Problemas com a Passagem Tradicional de Props

- **Prop Drilling**: Passar propriedades através de múltiplos níveis de componentes.
- **Complexidade**: Aumenta a complexidade do código.
- **Performance**: Pode afetar o desempenho ao atualizar muitos componentes desnecessariamente.

Vantagens do useContext

- **Acesso Direto**: Permite que componentes acessem dados globais diretamente.
- **Eficiência**: Evita atualizações desnecessárias em componentes intermediários.
- **Simplicidade**: Simplifica o gerenciamento de estados globais.

```
O useContext é ideal para dados que precisam ser acessados por
muitos componentes em diferentes níveis da árvore de componentes.
```

Implementação do useContext com TypeScript

Vamos criar um contexto de tema (Dark Mode) usando TypeScript e React.

Criando o Arquivo de Contexto

1. Crie um arquivo `theme-context.tsx` na pasta `src`.
2. Defina a interface para o contexto:

```
interface ThemeContextType {  theme: 'light' | 'dark';
toggleTheme: () => void;}
```

3. Crie o contexto:

```
const ThemeContext = createContext<ThemeContextType | undefined>
(undefined);
```

Criando o Provedor de Tema

1. Implemente o `ThemeProvider`:

```
export const ThemeProvider: React.FC<{ children: ReactNode }> = ({
children }) => {  const [theme, setTheme] = useState<'light' |
'dark'>('light');  const toggleTheme = () => {
setTheme(prevTheme => prevTheme === 'light' ? 'dark' : 'light');
};  return (    <ThemeContext.Provider value={{ theme, toggleTheme
}}>      {children}    </ThemeContext.Provider>  );};
```

2. Crie um hook personalizado para usar o contexto:

```
export const useTheme = () => {  const context =
useContext(ThemeContext);  if (!context) {    throw new
```

```
Error('useTheme deve ser utilizado com um ThemeProvider');  }
return context;};
```

Integrando o Contexto na Aplicação

1. No arquivo `main.tsx`, envolva o componente `App` com o `ThemeProvider`:

```
ReactDOM.createRoot(document.getElementById('root')!).render(
<React.StrictMode>    <ThemeProvider>      <App />
</ThemeProvider>  </React.StrictMode>);
```

2. No componente `App`, use o contexto:

```
function App() {  const { theme, toggleTheme } = useTheme();
return (    <div className={`container ${theme}`}>      {/*
Conteúdo do app */}      <button onClick={toggleTheme}>
Alterar para tema {theme === 'light' ? 'escuro' : 'claro'}
</button>    </div>  );}
```

Implementando o Dark Mode

Com o contexto configurado, podemos implementar a funcionalidade de Dark Mode.

Adicionando Estilos CSS

1. Crie classes CSS para os temas claro e escuro:

```
.container.light {  background-color: #ffffff;  color: #000000;}.container.dark {  background-color: #333333;  color: #ffffff;}
```

Usando o Tema no Componente

1. Aplique as classes dinamicamente baseado no tema atual:

```
<div className={`container ${theme}`}>  {/* Conteúdo do app */}</div>
```

2. Atualize o texto do botão de acordo com o tema:

```
<button onClick={toggleTheme}>  Alterar para tema {theme === 'light' ? 'escuro' : 'claro'}</button>
```

Dicas e Boas Práticas

- **Tipagem Correta**: Sempre defina interfaces claras para seus contextos.
- **Use Hooks Personalizados**: Crie hooks como `useTheme` para encapsular a lógica de uso do contexto.
- **Evite Overuse**: Use o Context API para estados verdadeiramente globais.
- **Teste Cuidadosamente**: Garanta que o tema persista corretamente entre recarregamentos de página.

```
Lembre-se: O useContext é poderoso, mas deve ser usado com
moderação para manter seu código organizado e eficiente.
```

Conclusão

O uso do `useContext` com TypeScript no React oferece uma solução elegante para gerenciar estados globais, como temas de aplicação. Ao implementar corretamente, você pode criar aplicações mais eficientes e fáceis de manter, evitando problemas comuns como prop drilling.

Este ebook cobriu os fundamentos da implementação do `useContext` para um sistema de tema, mas os princípios podem ser aplicados a muitos outros cenários onde o compartilhamento de estado global é necessário. Continue praticando e explorando outras aplicações do `useContext` para melhorar suas habilidades de desenvolvimento React com TypeScript.