Implementando Context API em uma Aplicação React

Por Escola Dnc

Introdução

Este ebook aborda a implementação do Context API em uma aplicação React, focando na criação de um estado global e no gerenciamento de idiomas. Vamos explorar como configurar o Context, utilizá-lo em diferentes componentes e criar funcionalidades de troca de idioma.

Configurando o Context API

Criando o Arquivo de Contexto

Para começar, criamos uma pasta chamada `Contexts` e dentro dela um arquivo `appcontext.jsx`. Neste arquivo, importamos as funções necessárias do React:

```
import { createContext, useState, useEffect } from 'react';import { getAPIData } from './services';
```

Definindo o Contexto e o Provider

Em seguida, criamos o contexto e o provider:

```
export const AppContext = createContext();export const AppProvider = ({ children }) => {  // Estados e lógica aqui  return (<AppContext.Provider value={/* valores */}>      {children}</AppContext.Provider>  );};
```

Criando Estados Globais

Dentro do provider, definimos os estados globais:

```
const [language, setLanguage] = useState('br');const [languages, setLanguages] = useState({});const [loading, setLoading] = useState(true);
```

- `language`: armazena o idioma atual
- `languages`: armazena os dados de idiomas da API
- `loading`: controla o estado de carregamento

Buscando Dados da API

Utilizamos o `useEffect` para buscar os dados de idiomas da API:

```
useEffect(() => {  const fetchLanguages = async () => {    try {
const texts = await getAPIData('webText');
setLanguages(texts);    } catch (error) {
console.error(error);    } finally {      setLoading(false);    }
};    fetchLanguages();}, []);
```

Utilizando o Context na Aplicação

Configurando o Provider no Arquivo Principal

No arquivo `main.jsx`, envolvemos nossa aplicação com o `AppProvider`:

```
import { AppProvider } from
'./Contexts/appcontext';ReactDOM.createRoot(document.getElementById('
<React.StrictMode>    <AppProvider>      <App />    </AppProvider>
</React.StrictMode>);
```

Acessando o Contexto nos Componentes

Para utilizar o contexto em um componente, usamos o hook `useContext`:

```
import { useContext } from 'react';import { AppContext } from
'./Contexts/appcontext';function MyComponent() {  const appContext
= useContext(AppContext);  // Usar appContext.language,
appContext.languages, etc.}
```

Implementando a Troca de Idioma

Criando a Função de Troca de Idioma

No componente onde desejamos implementar a troca de idioma (por exemplo, no footer):

```
const changeLanguage = (country) => {
appContext.setLanguage(country);};
```

Adicionando Botões de Troca de Idioma

```
<img   src="/br.png"   alt="Bandeira do Brasil"   onClick={() =>
changeLanguage('br')} /><img   src="/en.png"   alt="Bandeira dos
EUA"   onClick={() => changeLanguage('en')} />
```

Renderização Condicional Baseada no Idioma

Para exibir o conteúdo de acordo com o idioma selecionado:

```
<p>
{appContext.languages[appContext.language]?.general?.footer?.logoText
</p>
```

Este código busca o texto apropriado baseado no idioma atual.

Conclusão

A implementação do Context API em uma aplicação React oferece uma solução eficiente para o gerenciamento de estado global e a internacionalização. Ao centralizar o estado e a lógica de idiomas, conseguimos criar uma aplicação mais organizada e fácil de manter, permitindo a troca dinâmica de idiomas sem a necessidade de prop drilling.

```
Pontos-chave:- O Context API simplifica o gerenciamento de estado global- A implementação facilita a internacionalização da aplicação- O uso de useContext permite acesso fácil aos dados em qualquer componente- A renderização condicional baseada no idioma torna a aplicação multilíngue
```

Com esta estrutura, é possível expandir facilmente a aplicação, adicionando mais idiomas ou funcionalidades que dependam de um estado global compartilhado.