Melhorando uma Aplicação React com Context API e Web Storage

Por Escola Dnc

Introdução

Neste ebook, vamos explorar técnicas avançadas para aprimorar uma aplicação React, focando no uso da Context API para gerenciamento de estado global e na implementação de internacionalização (i18n). Também abordaremos o uso de Web Storage para melhorar a experiência do usuário. Essas técnicas são fundamentais para criar aplicações web modernas, responsivas e centradas no usuário.

Implementando um Loading Spinner

Criando o Componente Loading Spinner

Para melhorar a experiência do usuário durante o carregamento de dados, vamos implementar um loading spinner personalizado:

1. Crie um novo componente chamado `LoadingSpinner`
2. Adicione os arquivos `LoadingSpinner.jsx` e `LoadingSpinner.css`
3. Estruture o componente básico:

```
import React from 'react';import './LoadingSpinner.css';import loadingSpinnerGif from '../assets/loading-spinner.gif';const LoadingSpinner = () => {  return (    <div className="loading-overlay-container">      <img src={loadingSpinnerGif} alt="Loading" height="80" />    </div>  );};export default LoadingSpinner;
```

Estilizando o Loading Spinner

Adicione estilos ao seu loading spinner para garantir uma aparência agradável e consistente:

```
.loading-overlay-container {  background: white;  height: 100vh;  position: fixed;  width: 100vw;  z-index: 1;  display: flex;  align-items: center;  justify-content: center;}
```

Utilizando o Loading Spinner

Importe e utilize o componente `LoadingSpinner` em seu `App.jsx`:

```
import LoadingSpinner from './components/LoadingSpinner';// ...
{isLoading ? <LoadingSpinner /> : /* resto do conteúdo */}
```

Dica: Para criar spinners personalizados, você pode usar ferramentas online como o loading.io.

Implementando Internacionalização com Context API

Configurando o Context

1. Crie um arquivo `AppContext.js` para gerenciar o estado global da aplicação
2. Implemente a lógica para alternar entre idiomas

```
import React, { createContext, useState, useEffect } from 'react';export const AppContext = createContext();export const AppProvider = ({ children }) => {  const [language, setLanguage] = useState('pt');  const [translations, setTranslations] = useState({});  useEffect(() => {    // Carregar traduções baseadas no idioma selecionado    fetchTranslations(language);  }, [language]);  const toggleLanguage = () => {    setLanguage(prevLang => prevLang === 'pt' ? 'en' : 'pt');  };  return (    <AppContext.Provider value={{ language, toggleLanguage, translations }}>      {children}    </AppContext.Provider>  );};
```

Aplicando Traduções nos Componentes

Utilize o `useContext` hook para acessar as traduções em seus componentes:

```
import React, { useContext } from 'react';import { AppContext } from './AppContext';const Header = () => {  const { translations } = useContext(AppContext);  return (    <header>      <nav>        <ul>          <li>{translations.header.home}</li>          <li>{translations.header.about}</li>          <li>{translations.header.projects}</li>          <li>{translations.header.contact}</li>        </ul>      </nav>    </header>  );};
```

Implementando o Seletor de Idiomas

Crie um componente para permitir que o usuário alterne entre idiomas:

```
import React, { useContext } from 'react';import { AppContext } from './AppContext';const LanguageSelector = () => {  const { language, toggleLanguage } = useContext(AppContext);  return (<button onClick={toggleLanguage}>      {language === 'pt' ? 'EN' : 'PT'}    </button>  );};
```

Atualizando Componentes com Traduções

Hero Component

```
import React, { useContext } from 'react';import { AppContext } from './AppContext';const Hero = () => {  const { translations } = useContext(AppContext);  return (    <section className="hero">      <h1>{translations.hero.title}</h1>      <p>{translations.hero.subtitle}</p>      <a href="#" className="cta-button">{translations.hero.cta}</a>    </section>  );};
```

About Component

```
import React, { useContext } from 'react';import { AppContext } from './AppContext';const About = () => {  const { translations } = useContext(AppContext);  return (    <section className="about">      <h2>{translations.about.title}</h2>      <p>{translations.about.p1}</p>      <p>{translations.about.p2}</p>      <p>{translations.about.p3}</p>    </section>  );};
```

Contact Form

```
import React, { useContext } from 'react';import { AppContext } from './AppContext';const ContactForm = () => {  const { translations } = useContext(AppContext);  return (    <form>      <h2>{translations.contact.title}</h2>      <input type="text" placeholder={translations.contact.pl1} />      <input type="email"
```

```
placeholder={translations.contact.pl2} />      <textarea
placeholder={translations.contact.pl3}></textarea>      <button
type="submit">{translations.general.send}</button>    </form>  );};
```

Utilizando Web Storage para Melhorar a Experiência do Usuário

Armazenando Preferências de Idioma

Utilize o `localStorage` para persistir a preferência de idioma do usuário:

```
const [language, setLanguage] = useState(() => {  return
localStorage.getItem('preferredLanguage') || 'pt';});useEffect(()
=> {  localStorage.setItem('preferredLanguage', language);},
[language]);
```

Caching de Dados

Implemente um sistema de cache para dados frequentemente acessados:

```
const [projects, setProjects] = useState(() => {  const
cachedProjects = localStorage.getItem('cachedProjects');  return
cachedProjects ? JSON.parse(cachedProjects) : [];});useEffect(() =>
{  const fetchProjects = async () => {    const response = await
fetch('/api/projects');    const data = await response.json();
setProjects(data);    localStorage.setItem('cachedProjects',
JSON.stringify(data));  };  if (projects.length === 0) {
fetchProjects();  }}, []);
```

Armazenando Progresso do Formulário

Salve o progresso do usuário em formulários longos:

```
const [formData, setFormData] = useState(() => {  const savedForm = sessionStorage.getItem('contactFormData');  return savedForm ? JSON.parse(savedForm) : { name: '', email: '', message: '' };});const handleInputChange = (e) => {  const { name, value } = e.target;  setFormData(prevData => {    const newData = { ...prevData, [name]: value };
sessionStorage.setItem('contactFormData', JSON.stringify(newData));
return newData;  });};
```

Conclusão

Neste ebook, exploramos técnicas avançadas para melhorar significativamente uma aplicação React. Implementamos um loading spinner personalizado para melhorar o feedback visual durante o carregamento de dados. Utilizamos a Context API para gerenciar o estado global da aplicação e implementar a internacionalização, permitindo que os usuários alternem facilmente entre diferentes idiomas.

Além disso, aproveitamos o poder do Web Storage (localStorage e sessionStorage) para persistir preferências do usuário, implementar caching de dados e salvar o progresso em formulários. Essas técnicas não apenas melhoram o desempenho da aplicação, mas também proporcionam uma experiência de usuário mais fluida e personalizada.

Ao aplicar esses conceitos em seus projetos, você estará criando aplicações web mais robustas, responsivas e centradas no usuário. Continue explorando e experimentando com essas técnicas para levar suas habilidades de desenvolvimento React ao próximo nível.