Desenvolvimento Front-end com React e TypeScript: Criando uma Lista de Tarefas

Por Escola Dnc

Introdução

Neste ebook, exploraremos o desenvolvimento de uma aplicação front-end utilizando React e TypeScript. Nosso projeto prático consiste na criação de uma lista de tarefas funcional, abordando aspectos importantes como a adição, remoção e marcação de tarefas como concluídas. Ao longo do processo, discutiremos a importância da tipagem em TypeScript e como ela pode melhorar a qualidade e a manutenibilidade do nosso código.

Estrutura Inicial do Projeto

Componentes Básicos

Nossa aplicação de lista de tarefas já possui uma estrutura inicial funcional. Até o momento, conseguimos:

- Adicionar novas tarefas à lista
- Marcar tarefas como concluídas ou não concluídas

No entanto, ainda existem algumas limitações e melhorias a serem feitas:

1. A interface do usuário não está visualmente atraente
2. Não há funcionalidade para remover tarefas da lista

Melhorando a Aparência com CSS

Implementação do CSS

Para melhorar a aparência da nossa aplicação, seguiremos os seguintes passos:

1. Localizar o arquivo `app.css` no projeto
2. Remover todo o conteúdo existente neste arquivo
3. Adicionar um novo conjunto de estilos CSS fornecido

```
Importante: O novo CSS será fornecido como parte dos materiais do curso. Certifique-se de copiar e colar corretamente o código fornecido no arquivo app.css.
```

Limpeza de Arquivos Desnecessários

Para evitar conflitos e manter nosso projeto organizado:

1. Apague o arquivo `index.css`, pois não será utilizado
2. Remova a importação do `index.css` no arquivo principal para evitar erros

Após essas alterações, nossa lista de tarefas terá uma aparência mais agradável e profissional.

Implementando a Funcionalidade de Remoção de Tarefas

Criação da Função de Remoção

Para adicionar a capacidade de remover tarefas, seguiremos estas etapas:

1. No arquivo principal (provavelmente `App.tsx`), crie uma nova função chamada `removerTarefa`
2. A função deve receber como parâmetro o `id` da tarefa a ser removida (tipo `string`)
3. Utilize o método `filter` para criar um novo array sem a tarefa que desejamos remover

Exemplo de implementação:

```
const removerTarefa = (id: string) => {  const tarefasAtualizadas = todos.filter(todo => todo.id !== id);
setTodos(tarefasAtualizadas);};
```

Adição do Botão de Remoção

Para permitir que o usuário remova tarefas, adicionaremos um botão a cada item da lista:

1. No componente que renderiza cada tarefa, adicione um novo elemento `<button>`

2. Configure o evento `onClick` do botão para chamar a função `removerTarefa`
3. Passe o `id` da tarefa como argumento para a função

Exemplo de implementação:

```
<button onClick={() => removerTarefa(todo.id)}>Remover</button>
```

Com essas alterações, os usuários poderão remover tarefas individualmente da lista.

Tipagem em TypeScript e Benefícios no Desenvolvimento React

Vantagens da Tipagem Estática

O uso de TypeScript em projetos React oferece diversas vantagens:

- **Autocompletar mais preciso**: O editor sugere propriedades e métodos corretos baseados nos tipos definidos
- **Detecção precoce de erros**: Erros de tipo são identificados durante o desenvolvimento, não em tempo de execução
- **Melhor documentação do código**: Os tipos servem como uma forma de documentação inline
- **Refatoração mais segura**: O compilador ajuda a identificar todos os lugares que precisam ser atualizados ao fazer mudanças

Exemplos Práticos de Tipagem

No nosso projeto, podemos ver exemplos de como a tipagem ajuda:

- Na definição de props para componentes
- Na tipagem de estados (por exemplo, o array de tarefas)

- Na definição de funções como `removerTarefa`, onde especificamos o tipo do parâmetro `id`

```
interface Todo {  id: string;  text: string;  completed:
boolean;}const [todos, setTodos] = useState<Todo[]>([]);const
removerTarefa = (id: string) => {  // implementação};
```

Próximos Passos: Persistência de Dados e useEffect

Limitação Atual

Atualmente, nossa aplicação tem uma limitação importante:

- As tarefas são perdidas ao atualizar a página, pois não estão sendo armazenadas de forma persistente

Introdução ao useEffect e Armazenamento Local

Na próxima etapa do desenvolvimento, abordaremos:

1. Como utilizar o hook `useEffect` do React para gerenciar efeitos colaterais
2. Implementação de persistência de dados utilizando o `localStorage` do navegador

Estes tópicos permitirão que nossa aplicação mantenha as tarefas mesmo após o recarregamento da página, melhorando significativamente a experiência do usuário.

Conclusão

Neste ebook, exploramos o desenvolvimento de uma aplicação de lista de tarefas utilizando React e TypeScript. Abordamos a estruturação inicial do projeto, melhorias na interface do usuário através de CSS, implementação da funcionalidade de remoção de tarefas e discutimos os benefícios da tipagem estática proporcionada pelo TypeScript no desenvolvimento React.

A combinação de React com TypeScript oferece um ambiente de desenvolvimento robusto e produtivo, permitindo a criação de aplicações front-end mais confiáveis e fáceis de manter. À medida que avançamos no projeto, continuaremos a explorar técnicas avançadas para melhorar nossa aplicação, incluindo a persistência de dados e o uso eficiente de hooks do React como o `useEffect`.

Lembre-se de que o desenvolvimento de software é um processo contínuo de aprendizado e aprimoramento. Continue praticando, explorando novas funcionalidades e aprofundando seu conhecimento em React e TypeScript para se tornar um desenvolvedor front-end mais eficiente e capacitado.