Aprimorando seu Projeto FrontChain com TypeScript e React

Por Escola Dnc

Introdução

Neste ebook, vamos explorar os avanços significativos feitos no projeto FrontChain, focando no uso de TypeScript com React para criar uma aplicação de lista de tarefas robusta e bem estruturada. Abordaremos a importância da tipagem, o gerenciamento de estados, e as melhores práticas para criar componentes funcionais eficientes.

Recapitulação do Progresso

Antes de mergulharmos nos novos conceitos, é importante reconhecer o quanto já foi alcançado em poucas aulas:

- Uma lista de tarefas estilizada foi implementada
- Funcionalidades de adicionar, marcar como concluída e remover tarefas estão operacionais
- O TypeScript está sendo utilizado para auxiliar nas validações

```
Importante: Se você estiver enfrentando dificuldades com os conceitos abordados até agora, não hesite em revisar as aulas anteriores. A compreensão sólida dos fundamentos é crucial para o progresso.
```

Tipagem de Componentes Funcionais

Importância da Tipagem

A tipagem adequada dos componentes funcionais é uma prática essencial no desenvolvimento com TypeScript e React. Ela proporciona:

- Maior clareza no código
- Prevenção de erros em tempo de desenvolvimento
- Melhor documentação implícita

Tipando Funções de Manipulação de Tarefas

Vamos analisar como tipar corretamente as principais funções do nosso aplicativo:

1. **Adicionar Tarefa**:

```
const adicionarTarefa = (novaTarefa: string): void => {  // Implementação};
```

2. **Remover Tarefa**:

```
const removerTarefa = (id: string): void => {  // Implementação};
```

3. **Marcar como Completo**:

```
const marcarComoCompleto = (id: string): void => {  // Implementação};
```

Observação: O uso do tipo de retorno `void` indica que estas funções não

Tipagem de Estados

A tipagem correta dos estados é fundamental para o gerenciamento eficiente da aplicação. Veja um exemplo:

```
const [estaCarregado, setEstaCarregado] = useState<boolean>(false);
```

Neste caso, estamos usando um estado genérico do tipo `boolean`, inicializado como `false`.

Gerenciamento de Estados

Criando Novos Estados

Um novo estado foi introduzido para controlar o carregamento da aplicação:

```
const [estaCarregado, setEstaCarregado] = useState<boolean>(false);
```

Este estado será utilizado para:

- Controlar a exibição de um indicador de carregamento
- Iniciar a busca de tarefas na memória quando necessário

Padronização de Nomenclatura

É crucial manter um padrão consistente na nomenclatura dos estados e funções. Por exemplo:

- Alteramos `setNewTodo` para `setNovoTodo` para manter a consistência com o idioma português usado no restante da aplicação.

```
const [novoTodo, setNovoTodo] = useState<string>('');
```

Aprimoramentos na Interface do Usuário

Contador de Tarefas

Foi implementado um contador que exibe o número total de tarefas:

```
<h1>Lista de Tarefas - {todos.length}</h1>
```

Indicador de Tarefas Concluídas

Uma função foi criada para obter e exibir a quantidade de tarefas concluídas:

```
const obterTarefasCompletas = (): Todo[] => {  return
todos.filter(todo => todo.completado);};
```

Esta função é utilizada para exibir uma estatística na interface:

```
<span>{obterTarefasCompletas().length} de {todos.length}</span>
```

Melhores Práticas e Dicas

1. **Consistência na Nomenclatura**: Mantenha um padrão consistente, seja em português ou inglês, em toda a aplicação.
2. **Tipagem Explícita**: Mesmo quando o TypeScript pode inferir o tipo, é uma boa prática declarar explicitamente o tipo de retorno das funções.
3. **Uso de Genéricos**: Aproveite os genéricos do TypeScript para criar estados mais flexíveis e reutilizáveis.
4. **Componentização**: Divida sua aplicação em componentes menores e reutilizáveis para melhorar a manutenibilidade.
5. **Tratamento de Erros**: Sempre considere casos de erro e valores indefinidos ao trabalhar com dados dinâmicos.

Conclusão

Neste ebook, exploramos como aprimorar um projeto React utilizando TypeScript, focando na tipagem correta de componentes funcionais, gerenciamento eficiente de estados e implementação de funcionalidades úteis na interface do usuário.

Lembre-se de que a prática constante e a revisão dos conceitos são fundamentais para o domínio dessas tecnologias. Continue experimentando, refatorando seu código e buscando maneiras de melhorar a estrutura e a eficiência de sua aplicação.

Na próxima etapa do projeto, você estará preparado para implementar funcionalidades mais avançadas e otimizar ainda mais seu código. Continue praticando e boa sorte em sua jornada de desenvolvimento!