Introdução

Este eBook é projetado para fornecer a você uma introdução sólida ao mundo do React, uma das bibliotecas de front-end mais populares e poderosas de hoje. Ao longo deste guia, vamos explorar os conceitos-chave que formam a espinha dorsal do React, permitindo-lhe construir aplicações web robustas e interativas de maneira eficaz.

O React é conhecido por sua flexibilidade e eficiência, permitindo aos desenvolvedores criar interfaces de usuário ricas e responsivas com um código bem estruturado e reutilizável.

Através deste eBook, você será guiado passo a passo nos fundamentos do React, desde a configuração do seu ambiente de desenvolvimento, passando pela criação de componentes, até a manipulação de estados e props, e muito mais.

O foco deste eBook é fornecer uma compreensão clara e prática dos conceitos fundamentais do React, sem a necessidade de explorar bibliotecas externas ou delinear em testes. Dessa forma, você poderá se concentrar em aprender o núcleo do React e, uma vez que tenha uma compreensão firme dos conceitos básicos, você estará bem preparado para explorar as vastas possibilidades que o ecossistema React tem a oferecer.

Cada capítulo deste eBook é dedicado a um conceito chave do React, explicado de maneira clara e acompanhado por exemplos práticos que ajudarão a consolidar o seu aprendizado.

Eu, Matheus, fiz com muito carinho este material para você dar os seus primeiros passos em React na direção correta, se você tem interesse em aprender mais sobre React ou o mundo da programação, recomendo conhecer os nossos cursos, **clicando aqui** você acessa a página.

Além disso, escrevi um outro eBook muito mais completo que este, com exercícios e projetos, ele está disponível na Amazon, conheça ele **clicando aqui.**

Sugiro aproveitar os materiais indicados acima após a leitura deste, onde você ganhará um grande conhecimento, vamos começar?

Conceito 1: Configurando o Ambiente de Desenvolvimento com Vite

Agora que você já tem uma visão geral do que é React, é hora de arregaçar as mangas e começar a configurar nosso ambiente de desenvolvimento. Neste capítulo, vamos focar na configuração do ambiente de desenvolvimento usando o Vite, uma ferramenta de compilação moderna que permite iniciar projetos React de maneira muito rápida e eficiente.

O Vite oferece uma série de vantagens em relação a outras ferramentas de compilação, incluindo inicialização rápida, recarregamento de módulos otimizado e uma série de recursos integrados que facilitam o desenvolvimento com React. Vamos seguir os passos abaixo para configurar seu primeiro projeto React com Vite.

Instalação do Node.js

Antes de podermos começar com o Vite, você precisará ter o Node.js instalado em seu computador. O Node.js é um ambiente de execução JavaScript que nos permite executar código JavaScript no lado do servidor.

1. Vá para o site oficial do Node.js.
2. Baixe e instale a versão LTS (Long Term Support) seguindo as instruções na tela.

Criando seu Primeiro Projeto React com Vite

Com o Node.js instalado, agora podemos prosseguir e criar nosso projeto React.

Abra o terminal ou prompt de comando.

Execute o seguinte comando para criar um novo projeto React com Vite:

```
npx create-vite my-react-app --template react
```

Neste comando:

- npx é um executor de pacote que vem com o Node.js.
- create-vite é o comando que instrui o Vite a criar um novo projeto.
- my-react-app é o nome do seu novo projeto.
- --template react especifica que queremos usar o template React.

Agora, navegue até o diretório do seu novo projeto com o comando:

```
cd my-react-app
```

Instale as dependências do projeto com o comando:

```
npm install
```

Executando seu Projeto

Agora que nosso projeto está configurado, é hora de executá-lo e ver o React em ação.

No terminal, execute o seguinte comando:

```
npm run dev
```

Abra seu navegador web e vá para http://localhost:3000. Você deverá ver a página inicial do seu aplicativo React em execução.

Parabéns! Você configurou com sucesso seu ambiente de desenvolvimento React com Vite e está pronto para começar a construir seu primeiro aplicativo React. Nos próximos capítulos, vamos explorar os conceitos-chave do React e

como você pode utilizá-los para construir aplicações web interativas e dinâmicas.

Conceito 2: Componentes em React

Os componentes são o coração do React. Eles são as unidades básicas de código que compõem as interfaces de usuário em aplicações React. Cada componente é como um bloco de construção personalizado que você pode usar para construir sua interface de usuário. Neste capítulo, vamos explorar o que são componentes, como criá-los, e a diferença entre Componentes Funcionais e Componentes de Classe.

O que são Componentes?

Um componente em React encapsula um pedaço de interface de usuário. Eles podem ser tão simples quanto um botão ou tão complexos quanto uma tabela de dados completa. Cada componente é independente, reutilizável e pode conter outros componentes.

Criando Componentes

Criar um componente em React é bastante simples. Abaixo, vamos criar um componente simples que renderiza uma mensagem na tela.

Dentro da pasta src do seu projeto, crie um novo arquivo chamado Greeting.js. Abra Greeting.js e adicione o seguinte código:

```
import React from 'react';

function Greeting() {
  return (
    <div>
      Olá, bem-vindo ao React!
    </div>
  );
```

```
}

export default Greeting;
```

Agora, vamos importar e usar o componente Greeting em nosso componente principal App.js. Abra App.js e adicione o seguinte código:

```
import React from 'react';
import Greeting from './Greeting';

function App() {
  return (
    <div>
      <Greeting />
    </div>
  );
}

export default App;
```

Agora, quando você executa seu aplicativo, deverá ver a mensagem "Olá, bem-vindo ao React!" renderizada na tela.

Componentes Funcionais versus Componentes de Classe

Existem dois tipos principais de componentes em React: Componentes Funcionais e Componentes de Classe.

Componentes Funcionais

Os componentes funcionais são simples funções JavaScript que aceitam props como argumento e retornam elementos React. Eles são fáceis de escrever, entender e testar.

```
function Greeting(props) {
  return <div>Olá, {props.name}!</div>;
}
```

Componentes de Classe

Os componentes de classe são mais tradicionais e foram a principal forma de criar componentes antes da introdução dos Hooks em React. Eles exigem uma sintaxe mais verbosa comparada aos componentes funcionais.

```
import React, { Component } from 'react';

class Greeting extends Component {
  render() {
    return <div>Olá, {this.props.name}!</div>;
  }
}

export default Greeting;
```

Hoje em dia, a tendência é usar componentes funcionais juntamente com Hooks, pois oferecem uma maneira mais concisa e elegante de gerenciar estado e ciclo de vida do componente.

Conceito 3: Props (Propriedades)

No React, os componentes comunicam-se passando dados uns aos outros através de um mecanismo chamado props (abreviação de propriedades). Os props permitem que você passe informações de um componente pai para um componente filho. Eles são essenciais para a reutilização de componentes e para construir interfaces de usuário dinâmicas. Neste capítulo, vamos explorar o que são props, como usá-los, e como eles facilitam a comunicação entre componentes.

O que são Props?

Props são objetos que contêm dados a serem passados de um componente para outro. Eles são como argumentos de uma função, permitindo que você configure e passe dados entre componentes. Os props são imutáveis, o que

significa que um componente não pode alterar seus próprios props, mas apenas lê-los.

Passando e Acessando Props

Para passar props a um componente, você os inclui como atributos quando renderiza o componente. Vamos ver como isso é feito através de um exemplo.

Vamos começar criando um componente Greeting que aceita um prop name. Dentro da pasta src do seu projeto, crie um novo arquivo chamado Greeting.js e adicione o seguinte código:

```
import React from 'react';

function Greeting(props) {
  return (
    <div>
      Olá, {props.name}!
    </div>
  );
}

export default Greeting;
```

Agora, vamos renderizar o componente Greeting dentro do nosso componente App, passando um prop name para ele. Abra o arquivo App.js e atualize o código para o seguinte:

import React from 'react';
import Greeting from './Greeting';

```
function App() {
  return (
    <div>
      <Greeting name="Mundo" />
    </div>
  );
}
```

```
export default App;
```

Agora, quando você executa seu aplicativo, deverá ver a mensagem "Olá, Mundo!" renderizada na tela.

Props são Somente Leitura

É importante notar que os props são "somente leitura". Isso significa que um componente não deve modificar os valores dos props que recebe. Esta regra ajuda a garantir que os componentes sejam previsíveis e fáceis de entender.

Validando Props com PropTypes

React tem uma biblioteca adicional chamada PropTypes que permite especificar os tipos de props que um componente deve receber. Isso pode ajudar a capturar bugs e melhorar a legibilidade do seu código.

Aqui está um exemplo de como você pode usar PropTypes para validar os props no componente Greeting:

```
import React from 'react';
import PropTypes from 'prop-types';

function Greeting(props) {
  return (
    <div>
      Olá, {props.name}!
    </div>
  );
}

Greeting.propTypes = {
  name: PropTypes.string.isRequired
};

export default Greeting;
```

Características dos Props:

Imutabilidade:

Os props são imutáveis, o que significa que um componente não pode alterar os valores dos props que recebe. Esta imutabilidade ajuda a manter o fluxo de dados unidirecional e previsível na sua aplicação.

Passagem de Dados:

Você pode passar diferentes tipos de dados através de props, incluindo números, strings, objetos, arrays, e até mesmo funções.

Valores Default:

Em React, você pode definir valores default para os props usando defaultProps. Isso é útil quando um prop não é passado para um componente.

```
import React from 'react';

function Greeting(props) {
  return <div>Olá, {props.name}!</div>;
}

Greeting.defaultProps = {
  name: 'Mundo',
};

export default Greeting;
```

Utilizando Props para Renderização Condicional

Os props também podem ser usados para controlar a renderização condicional dentro de um componente. Por exemplo, você pode renderizar diferentes UIs baseadas no valor de um prop.

```jsx
import React from 'react';

function WelcomeMessage(props) {
  if (props.isLoggedIn) {
    return <div>Bem-vindo de volta!</div>;
  } else {
    return <div>Por favor, faça login.</div>;
  }
}

export default WelcomeMessage;
```

Passando Funções através de Props

Você pode passar funções como props para permitir a comunicação entre componentes. Isso é especialmente útil para lidar com eventos e compartilhar lógica entre componentes.

```jsx
import React from 'react';

function Button(props) {
  return <button onClick={props.onClick}>Clique em mim</button>;
}

function App() {
  function handleClick() {
    alert('Botão foi clicado!');
  }

  return (
    <div>
      <Button onClick={handleClick} />
    </div>
  );
}

export default App;
```

Desestruturação de Props

A desestruturação é uma característica do ES6 que permite desempacotar valores de arrays ou propriedades de objetos em variáveis distintas. Isso pode ser usado para tornar seu código mais limpo e legível ao trabalhar com props.

i

```
mport React from 'react';

function Greeting({ name }) {
  return <div>Olá, {name}!</div>;
}

export default Greeting;
```

Conceito 4: Estado (State)

O estado é um conceito fundamental no React que permite aos componentes manter e gerenciar dados dinâmicos. Diferente dos props, o estado é mutável e pode ser alterado ao longo do tempo, permitindo a criação de interfaces de usuário interativas. Neste capítulo, vamos explorar o que é o estado, como utilizá-lo e como ele se relaciona com a renderização de componentes.

O que é Estado?

O estado em um componente React é um objeto que contém dados que podem mudar ao longo do tempo. Cada vez que o estado de um componente muda, o componente é re-renderizado, e todos os seus descendentes também são re-renderizados.

Inicializando Estado com useState

O hook useState é a maneira mais comum de adicionar estado a um componente funcional no React. Vamos criar um contador simples para ilustrar como o useState funciona.

Dentro da pasta src do seu projeto, crie um novo arquivo chamado Counter.js.

Abra Counter.js e adicione o seguinte código:

```
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  function handleIncrement() {
    setCount(count + 1);
  }

  function handleDecrement() {
    setCount(count - 1);
  }

  return (
    <div>
      <button onClick={handleDecrement}>-</button>
      <span>{count}</span>
      <button onClick={handleIncrement}>+</button>
    </div>
  );
}

export default Counter;
```

Agora, importe e use o componente Counter em seu componente App. Abra App.js e atualize o código para o seguinte:

```
import React from 'react';
import Counter from './Counter';

function App() {
  return (
    <div>
      <Counter />
    </div>
  );
}

export default App;
```

Agora você tem um contador funcional! Cada vez que você clicar nos botões, o número no meio será incrementado ou decrementado.

Estado e Renderização

Quando o estado de um componente muda, o componente é re-renderizado. Isso significa que a função do componente é chamada novamente, e a nova renderização reflete o estado atual do componente.

No exemplo do contador acima, cada vez que o count muda, o componente Counter é re-renderizado, mostrando o novo valor na tela.

Funcionalidade Lazy de useState

useState também tem uma característica onde você pode passar uma função para o valor inicial, que só será executada na primeira renderização, isso é útil quando o cálculo do estado inicial é caro.

```
const [myState, setMyState] = useState(() => computeExpensiveValue());
```

Atualizações de Estado Assíncronas

As atualizações de estado são assíncronas em React. Isso significa que se você ler o estado imediatamente após chamar a função de atualização, você obterá o valor antigo. Uma solução comum é usar uma função callback dentro da função de atualização, que lhe dará o estado anterior como argumento.

```
setMyState(prevState => prevState + 1);
```

Uso de useState com Objetos e Arrays

Você também pode usar useState com objetos e arrays. No entanto, ao contrário do this.setState em componentes de classe, a função de atualização de useState não mescla objetos automaticamente. Você terá que fazer isso manualmente.

```
const [user, setUser] = useState({ name: '' });

// Para atualizar o nome do usuário, você precisaria fazer algo assim:
setUser(prevUser => ({
  ...prevUser,
  name: 'Novo Nome'
}));
```

Conceito 5: JSX (JavaScript XML)

JSX, ou JavaScript XML, é uma extensão de sintaxe para JavaScript que se parece muito com XML ou HTML. É uma maneira concisa e fácil de escrever componentes React, permitindo que você descreva a estrutura da UI de forma declarativa. Neste capítulo, vamos explorar o que é JSX, como ele é transformado em chamadas de função JavaScript regulares, e como você pode utilizá-lo para construir interfaces de usuário complexas.

O que é JSX?

JSX permite que você escreva elementos HTML e componentes personalizados de forma clara e legível, misturando marcação e lógica em um único arquivo. É uma forma sucinta de descrever a estrutura e os componentes da UI em React.

```
const element = <h1>Olá, mundo!</h1>;
```

Transformação de JSX

O JSX não é entendido pelos navegadores e requer uma etapa de transformação para convertê-lo em JavaScript puro. Ferramentas como Babel são usadas para

transformar JSX em chamadas de função React.createElement, que é o que o React usa por trás dos panos para criar elementos e componentes.

Por exemplo, o seguinte código JSX:

```
const element = <h1>Olá, mundo!</h1>;
```

É transformado em:

```
const element = React.createElement('h1', null, 'Olá, mundo!');
```

Expressões em JSX

Você pode incorporar qualquer expressão JavaScript válida dentro de chaves {} em JSX. Isso permite que você misture lógica JavaScript com a marcação JSX de uma maneira muito direta.

```
const name = 'Mundo';
const element = <h1>Olá, {name}!</h1>;
```

Atributos e Nomes de Componentes

Os nomes dos componentes em JSX devem começar com uma letra maiúscula, para distinguir componentes personalizados de elementos HTML nativos. Além disso, os atributos em JSX seguem uma convenção camelCase ao invés da notação kebab-case típica de HTML.

```
// Componente personalizado
const MyComponent = () => <div>Meu Componente</div>;

// Uso de atributos camelCase
const element = <div className="myClass">Texto</div>;
```

Auto-fechamento de Tags

Em JSX, as tags que não têm filhos podem ser auto-fechadas, similarmente ao XML e HTML.

```
const element = <img src="myImage.jpg" alt="My Image" />;
```

Comentários em JSX

Para adicionar comentários dentro do JSX, você pode usar a sintaxe {/* */}.

```
const element = (
  <div>
    {/* Este é um comentário */}
    Olá, mundo!
  </div>
);
```

Conceito 6: Manipulação de Eventos

Manipular eventos é crucial em qualquer aplicação interativa. No React, a manipulação de eventos é muito semelhante à manipulação de eventos no DOM, com algumas diferenças de sintaxe. Neste capítulo, vamos explorar como lidar com eventos de usuário como cliques e entradas de formulário, permitindo uma interação rica entre o usuário e a aplicação.

Sintaxe de Manipulação de Eventos

No React, os nomes dos eventos são escritos em camelCase, em vez de letras minúsculas, e você passa uma função como o manipulador de eventos, em vez de uma string.

```
// HTML
<button onclick="handleClick()">Clique em mim</button>

// React
<button onClick={handleClick}>Clique em mim</button>
```

Definindo Manipuladores de Eventos

Você pode definir um manipulador de eventos em um componente como uma função. Veja um exemplo de um componente que tem um botão com um manipulador de evento onClick:

```
import React from 'react';

function MyButton() {
  function handleClick() {
    alert('Botão foi clicado!');
  }

  return (
    <button onClick={handleClick}>
      Clique em mim
    </button>
  );
}

export default MyButton;
```

Uso do this em Manipuladores de Eventos

Em JavaScript, a palavra-chave this dentro de métodos é undefined. No React, isso pode ser contornado ao usar arrow functions ou ao fazer o bind do método no construtor do componente.

```
class MyButton extends React.Component {
  constructor(props) {
    super(props);
    this.handleClick = this.handleClick.bind(this);
  }
```

```
  handleClick() {
    alert('Botão foi clicado!');
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Clique em mim
      </button>
    );
  }
}
```

Eventos de Formulário

Manipular eventos de formulário é muito comum em aplicações React. Por exemplo, você pode querer ter um manipulador de eventos que é acionado cada vez que o usuário muda o texto em um campo de input:

```
import React, { useState } from 'react';

function MyForm() {
  const [text, setText] = useState('');

  function handleChange(event) {
    setText(event.target.value);
  }

  return (
    <form>
      <input type="text" value={text} onChange={handleChange} />
    </form>
  );
}

export default MyForm;
```

Conceito 7: Listas e Chaves

Quando se trata de exibir uma coleção de itens no React, as listas e chaves desempenham um papel crucial. Este capítulo abordará como renderizar listas de elementos de forma eficiente e por que as chaves únicas são essenciais para o desempenho e a consistência do seu aplicativo.

Renderizando Listas

Em React, você pode construir coleções de elementos e incluí-las em JSX com a ajuda de map(). A função map() percorre os itens de uma array e retorna um novo array de elementos React.

Vamos ver um exemplo onde renderizamos uma lista de números:

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

No código acima, map() é chamado sobre a array numbers. Para cada item, retornamos um <li> elemento que será parte de uma lista <ul>.

A Importância das Chaves

Chaves ajudam o React a identificar quais itens mudaram, foram adicionados ou removidos. Chaves devem ser dadas aos elementos dentro da array para dar aos elementos uma identidade estável:

```
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
```

```
  </li>
);
```

Escolhendo a Chave Certa

A melhor maneira de escolher uma chave é usar uma string que identifique de forma única um item da lista entre seus irmãos. Muitas vezes, você usará IDs de seus dados como chaves:

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```

Quando você não tem IDs estáveis para itens renderizados, você pode usar o índice do item como uma chave como último recurso:

```
const todoItems = todos.map((todo, index) =>
  <li key={index}>
    {todo.text}
  </li>
);
```

No entanto, usar índices como chaves não é recomendado se a ordem dos itens pode mudar, pois isso pode impactar negativamente o desempenho e pode causar problemas com o estado do componente.

Extrair Componentes com Chaves

Chaves só fazem sentido no contexto do array circundante. Por exemplo, se você extrair um componente ListItem, você deve manter a chave no <ListItem /> dentro do array, e não no elemento <li> no próprio ListItem.

```
function ListItem(props) {
  // Correto! Não há necessidade de especificar a chave aqui:
  return <li>{props.value}</li>;
}
```

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // Correto! A chave deve ser especificada dentro do array.
    <ListItem key={number.toString()} value={number} />
  );
  return (
    <ul>{listItems}</ul>
  );
}
```

Conceito 8: Renderização Condicional

A renderização condicional no React permite que você exiba componentes ou elementos com base em uma condição específica. Isso é semelhante às condições em JavaScript puro: dependendo de uma expressão lógica, diferentes partes do código podem ser executadas. Este capítulo aborda como realizar a renderização condicional em componentes React, baseando-se em estados ou props.

Conceitos Básicos de Renderização Condicional

No React, você pode criar componentes que encapsulam o comportamento que você precisa. Então, você pode renderizar apenas alguns deles, dependendo do estado da sua aplicação.

Elementos Variáveis

Você pode usar variáveis para armazenar elementos. Isso pode ajudá-lo a renderizar condicionalmente uma parte do componente enquanto o resto do output não muda.

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  let greeting;
```

```
  if (isLoggedIn) {
    greeting = <WelcomeBack />;
  } else {
    greeting = <PleaseSignIn />;
  }

  return (
    <div>
      {greeting}
    </div>
  );
}
```

Operador Condicional Inline

Você pode usar o operador lógico && para incluir um componente somente se uma condição for verdadeira.

```
function Mailbox(props) {
  const unreadMessages = props.unreadMessages;
  return (
    <div>
      {unreadMessages.length > 0 &&
        <h2>
          Você tem {unreadMessages.length} mensagens não lidas.
        </h2>
      }
    </div>
  );
}
```

Outra opção é usar o operador ternário para escolher entre dois componentes diferentes:

```
const isLoggedIn = props.isLoggedIn;
return (
  <div>
    {isLoggedIn ? <LogoutButton onClick={this.handleLogoutClick} /> :
<LoginButton onClick={this.handleLoginClick} />}
```

```
  </div>
);
```

Prevenir o Render de Componentes

Às vezes, você quer que um componente não renderize nada. Para fazer isso, você pode fazer que o render retorne null, em vez de seu render output.

```
function WarningBanner(props) {
  if (!props.warn) {
    return null;
  }

  return (
    <div className="warning">
      Atenção!
    </div>
  );
}
```

Retornar null de um componente render não afetará o disparo dos métodos de ciclo de vida do componente.

Componentes que Encapsulam Condições

Às vezes, você pode encapsular o comportamento condicional em um componente, para que o componente principal fique mais limpo.

```
function MyComponent(props) {
  return (
    <div>
      <ConditionalRenderingComponent isLoggedIn={props.isLoggedIn} />
      {/* ... outros componentes ... */}
    </div>
  );
}
```

Usando IIFE para Renderização Condicional Inline

Para expressões condicionais mais complexas ou para incluir lógica de loop, você pode usar uma IIFE (Immediately Invoked Function Expression) diretamente dentro do JSX.

```
<div>
  {
    (() => {
      if (condition) return <ComponentA />;
      if (otherCondition) return <ComponentB />;
      return <ComponentC />;
    })()
  }
</div>
```

A renderização condicional é uma técnica poderosa que permite criar interfaces de usuário dinâmicas e adaptativas. Ao entender e aplicar as práticas discutidas acima, você pode tornar seus componentes React mais reativos ao estado e às props, resultando em uma experiência de usuário mais rica e interativa.

Conceito 9: Lifting State Up (Elevação de estado)

No desenvolvimento de aplicações React, frequentemente encontramos a necessidade de compartilhar estado entre componentes. Com componentes funcionais, utilizamos hooks para gerenciar estados e, quando necessário, elevamos esse estado para um componente ancestral comum. Este capítulo explica como compartilhar o estado entre componentes funcionais e a importância de "lifting state up" para a propagação de dados eficiente.

Entendendo "Lifting State Up" em Componentes Funcionais

"Lifting State Up" em componentes funcionais envolve mover o estado para um componente pai para que possa ser acessado por vários componentes filhos através das props.

A Necessidade de Elevar o Estado

Elevar o estado é útil quando dois ou mais componentes precisam acessar ou modificar os mesmos dados. Ao manter o estado no componente pai, podemos garantir que o estado seja consistente em todos os componentes filhos.

Implementando o Lifting State Up com Hooks

Vamos usar o hook useState para ilustrar como o estado é elevado e compartilhado entre componentes.

Suponha que temos dois componentes que devem compartilhar e manipular um estado comum:

```
function TemperatureInput({ temperature, onTemperatureChange }) {
  // ...

  function handleChange(e) {
    onTemperatureChange(e.target.value);
  }

  return (
    <input value={temperature} onChange={handleChange} />
  );
}
```

O componente pai, que gerencia o estado compartilhado, poderia ser assim:

```
function Calculator() {
  const [temperature, setTemperature] = useState('');

  function handleTemperatureChange(newTemperature) {
    setTemperature(newTemperature);
  }

  return (
    <div>
      <TemperatureInput
        temperature={temperature}
```

```
        onTemperatureChange={handleTemperatureChange}
      />
      {/* Outros componentes que usam o estado temperature */}
    </div>
  );
}
```

Neste exemplo, Calculator mantém o estado temperature e passa esse estado e a função para atualizá-lo para o TemperatureInput via props.

Vantagens do Lifting State Up

Ao elevar o estado, você centraliza a sua lógica de estado em um local. Isso significa que o estado pode ser compartilhado entre componentes e facilita o rastreamento de mudanças, a depuração e a manutenção da aplicação.

Considerações ao Elevar o Estado

Embora elevar o estado possa ser útil, também pode levar a uma propagação excessiva de props se muitos níveis forem envolvidos. Para esses casos, pode ser mais adequado usar um padrão de gerenciamento de estado global, como Context ou Redux.

A técnica de lifting state up em componentes funcionais é uma parte fundamental do gerenciamento de estado em React. Ela permite que o estado seja compartilhado entre componentes de maneira eficiente e controlada.

Conceito 10: Continuando o seu aprendizado em React

Agora que você já aprendeu e entendeu os fundamentos do React, é importante seguir evoluindo seu conhecimento, para isso eu criei vários cursos que vão te ajudar neste processo, e outros ebooks também.

Obs: Os materiais abaixo estão com descontos promocionais aplicados aos seus links, é só clicar que você aproveita a promoção.

Confira as minhas recomendações por ordem de crescente de aprendizado:

Cursos

React do Zero a Maestria (c/ hooks, router, API, Projetos)

O curso "React do Zero a Maestria" é uma imersão completa na biblioteca React, ideal para quem deseja não apenas aprender a criar interfaces dinâmicas, mas também para aqueles que querem entender a integração do React em um ecossistema de desenvolvimento completo com tecnologias como Firebase, Node.js, MongoDB e Redux.

Por que cada elemento deste curso é ótimo para o aprendizado do React?

Criação de Aplicações Completas com React: Desenvolver um projeto do início ao fim solidifica o entendimento de conceitos e práticas, preparando os alunos para cenários do mundo real.

Gerenciamento de Páginas com React Router: Dominar o roteamento é essencial para a criação de aplicações de página única (SPAs), que oferecem experiências mais fluidas e modernas para os usuários.

Utilização de Todos os Hooks do React: Os Hooks são uma evolução no gerenciamento de estados e efeitos colaterais em componentes funcionais, representando as práticas atuais na escrita de componentes React.

Gerenciamento de Contexto com Context API: Aprender a Context API permite gerenciar o estado global de maneira mais eficiente, sem prop drilling excessivo.

Integração de React com Firebase: O Firebase é uma plataforma poderosa que oferece funcionalidades como banco de dados em tempo real e autenticação, habilidades valiosas no desenvolvimento de aplicações modernas.

MERN Stack: Compreender como o React se integra no stack MERN (MongoDB, Express, React, Node.js) é crucial para desenvolvedores que desejam criar aplicações full-stack robustas.

Além da instrução em vídeo, o curso oferece exercícios práticos, leituras complementares e recursos para download, proporcionando uma experiência de aprendizado abrangente, e certificado de conclusão.

Este curso é particularmente importante porque não assume apenas que os alunos vão construir o front-end, mas também se aprofunda na integração do React com o back-end, operações com banco de dados e autenticação. Ao final, os alunos estarão prontos para enfrentar o mercado de trabalho com uma habilidade altamente demandada.

Ideal para desenvolvedores front-end que querem evoluir para full-stack, bem como para profissionais de back-end que desejam compreender o front-end, o curso é um recurso abrangente para quem quer dominar React e estar preparado para contribuir em qualquer parte da stack de uma aplicação web moderna.

20+ Projetos em React JS aprenda Redux, Bootstrap, APIs

O curso "20+ Projetos em React JS aprenda Redux, Bootstrap, APIs" é uma experiência de aprendizado prático, destinada a desenvolvedores front-end que já possuem conhecimentos fundamentais em HTML, CSS, JavaScript e React. Este curso é ideal para aqueles que desejam aprofundar suas habilidades técnicas e construir um portfólio robusto com projetos reais.

O que você vai aprender com os projetos do curso:

Projetos Práticos com React JS: Aprender React por meio de projetos práticos é uma das maneiras mais eficazes de entender como a teoria se aplica ao mundo real, além de ser uma excelente oportunidade para enfrentar desafios comuns do desenvolvimento front-end.

Integração com TypeScript: O TypeScript tem se tornado cada vez mais popular por proporcionar um desenvolvimento mais seguro e escalável, e aprender a integrá-lo com React é essencial para desenvolvedores que desejam se manter atualizados com as práticas modernas.

Uso de Bibliotecas Complementares: As bibliotecas mencionadas, como Bootstrap, Chakra UI, Axios e Redux, são ferramentas amplamente utilizadas no mercado. Conhecê-las é quase um requisito na indústria atualmente, e saber aplicá-las em um contexto de projeto é uma habilidade altamente valorizada.

Fluxo do Redux com React JS: Redux é uma biblioteca para gerenciamento de estado previsível e é crucial para aplicações React de grande porte. Entender seu fluxo e como ele interage com o React pode diferenciar um desenvolvedor médio de um especialista.

Resolução de Problemas com Código de Qualidade: Resolver problemas complexos com código limpo e eficiente é uma habilidade que os empregadores valorizam. Este curso foca em ensinar como abordar desafios de codificação de maneira estratégica e pensada.

O curso oferece um ambiente de aprendizado altamente interativo e dinâmico, pois é inteiramente baseado em projetos, proporcionando experiência prática que é diretamente transferível para o ambiente de trabalho. Além disso, o suporte especializado e o acesso a

futuras atualizações garantem que os alunos tenham uma experiência contínua de aprendizado.

Este curso é apropriado para desenvolvedores que querem migrar de outras frameworks para React ou programadores que estão começando no React e desejam avançar rapidamente. Com a finalização deste curso, os alunos terão um portfólio diversificado e prático que demonstrará suas habilidades avançadas em React JS para possíveis empregadores.

Ebook

Fundamentos do React JS: Aprenda a bilioteca mais famosa de front-end através de exercícios e projetos

O eBook "Fundamentos do React JS" é uma leitura valiosa para estudantes e desenvolvedores que buscam uma compreensão abrangente da biblioteca de front-end React.

Aqui estão as razões pelas quais cada tópico abordado no livro é crucial para o aprendizado efetivo de React:

Fundamentos do React: Dominar os fundamentos é essencial para qualquer aprendizado. Ao entender os princípios básicos do React, incluindo JSX, componentes, props e estado, os alunos estabelecem uma base sólida que permite compreender conceitos mais complexos.

Ciclo de Vida e Manipulação de Eventos: Compreender o ciclo de vida de um componente e a manipulação de eventos é fundamental para criar interfaces interativas e dinâmicas, habilidades cruciais para um desenvolvedor front-end moderno.

Padrões de Projeto em React: Explorar padrões de design não só ajuda a estruturar aplicações eficientemente, mas também facilita a colaboração e a manutenção do código em projetos maiores.

Estado e Gerenciamento de Dados: Proficiência em gerenciamento de estado e o uso de Context API e Hooks é vital para gerenciar o fluxo de dados dentro de aplicações, permitindo que os alunos criem apps robustos e reativos.

Roteamento e Navegação: A capacidade de implementar navegação complexa é uma habilidade prática que melhora a usabilidade e a estrutura das aplicações web.

Otimização de Desempenho: Aprender estratégias de otimização é crucial para construir aplicações rápidas e eficientes, melhorando a experiência do usuário final.

Melhores Práticas e Padrões de Código: Adotar as melhores práticas e padrões de código não só melhora a qualidade do desenvolvimento, mas também a longevidade e escalabilidade do código produzido.

Recursos e Ferramentas: Ter acesso a uma variedade de recursos e ferramentas enriquece o processo de aprendizado, fornecendo ao aluno um caminho claro para aprofundamento e prática contínua.

Além do conteúdo teórico, o eBook oferece exemplos de código práticos, exercícios e desafios para testar o conhecimento adquirido, recursos adicionais para aprofundamento e acesso a uma comunidade de suporte, o que é extremamente valioso para o aprendizado contínuo e aplicação prática do conhecimento.

Ao fim do estudo deste eBook, o aluno estará bem preparado para construir aplicações React por conta própria e terá a confiança necessária para enfrentar desafios mais complexos na programação front-end.

Conclusão: O Início da Sua Aventura com React

E aqui estamos, no final deste guia inicial que criei para você. Espero que cada página, cada exemplo e cada conceito tenham servido como tijolos fundamentais na construção do seu entendimento sobre o React.

Sei que pode parecer muita informação de uma vez, mas lembre-se de que cada grande desenvolvedor começou exatamente onde você está agora - no começo. Eu também estive lá, descobrindo os mistérios do JSX, desvendando o estado e as props, e me maravilhando com a reatividade que o React proporciona.

Olhando para trás, para os meus primeiros dias mexendo com esses conceitos, posso dizer com confiança que a prática contínua e a curiosidade incansável são suas maiores aliadas. Não se acanhe diante dos desafios; eles são apenas oportunidades disfarçadas que irão fortalecer suas habilidades e compreensão.

Quero que você veja este eBook não como um ponto de chegada, mas como um ponto de partida vibrante. Há um universo inteiro de projetos esperando por você, problemas intrigantes para resolver e, claro, uma comunidade incrível de desenvolvedores React com os quais você pode aprender e crescer.

Estou ansioso para ver onde sua jornada React vai levar você. E lembre-se, estou torcendo por você a cada passo do caminho. Se você tiver dúvidas ou quiser mostrar o que criou, não hesite em me procurar. Afinal, codificar é mais do que escrever linhas de código; é conectar-se, criar e colaborar.

Com um sorriso no rosto e um teclado à disposição, digo: Vamos nessa! Sua aventura com React está apenas começando, e eu não poderia estar mais empolgado por você.

Com carinho e desejo de muito código para você, Matheus Battisti - Hora de Codar!