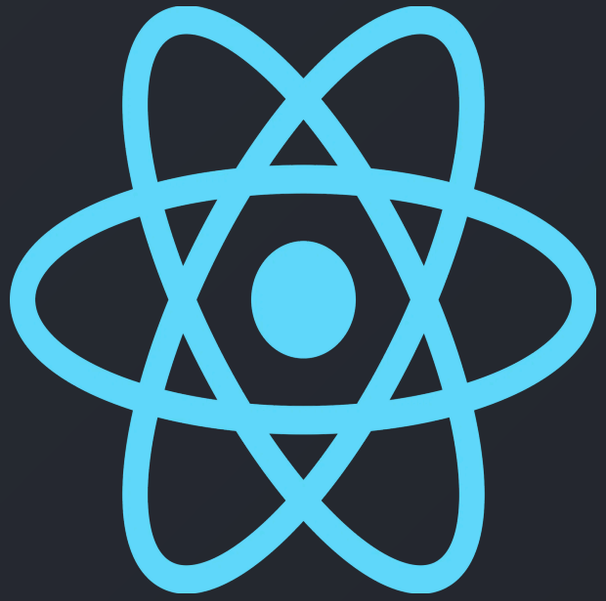


Introdução ao React

Construindo Interfaces de Usuário Modernas

Seu Guia Rápido para o Desenvolvimento Front-end com React

Conceitos Fundamentais: Componentes e JSX

Componentes

Componentes são os blocos de construção fundamentais em React. Eles permitem dividir a interface em partes independentes e reutilizáveis.

Existem dois tipos principais:

- Componentes Funcionais (recomendados)
- Componentes de Classe (legado)

JSX

JSX é uma extensão de sintaxe para JavaScript que parece HTML, mas permite inserir expressões JavaScript usando chaves { }.

O JSX é transformado em chamadas de função React.createElement() durante a compilação.

```jsx
import React from 'react';

// Componente Funcional com JSX
function HelloWorld() {
  const name = 'Mundo';

  return (
    <div>
      <h1>Olá, {name}!</h1>
      <p>Bem-vindo ao React</p>
    </div>
  );
}

export default HelloWorld;
```

Props: Passando Dados para Componentes

O que são Props?

Props (abreviação de "properties") são argumentos passados para componentes React, semelhantes aos atributos HTML.

Características das Props

- São somente leitura (imutáveis)
- Fluem de cima para baixo (pai para filho)
- Podem ser de qualquer tipo de dado
- Permitem componentes reutilizáveis

```jsx
import React from 'react';

// Componente filho que recebe props
function Greeting(props) {
  return (
    <div>
      <h1>Olá, {props.name}!</h1>
      <p>{props.message}</p>
    </div>
  );
}

// Componente pai que passa props
function App() {
  return (
    <div>
      <Greeting
        name="Maria"
        message="Bem-vinda ao React!"
      />
    </div>
  );
}
```

State: Gerenciando o Estado Interno

O que é State?

State (estado) é um objeto JavaScript que armazena dados mutáveis de um componente. Quando o state muda, o componente é renderizado novamente.

Hook useState

O hook useState permite adicionar estado a componentes funcionais:

1. Importar useState do React
2. Declarar variável de estado e função atualizadora
3. Usar a variável de estado no JSX
4. Chamar a função atualizadora para mudar o estado

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

function Counter() {
  // Declaração do state
  const [count, setCount] = useState(0);

  return (
    <div>
      <h2>Contador: {count}</h2>

      <button
        onClick={() => setCount(count + 1)}
        style={{
          backgroundColor: '#61DAFB',
          color: '#282C34',
          padding: '10px 20px',
          borderRadius: '4px'
        }}
      >
        Incrementar
      </button>

      <button
        onClick={() => setCount(count - 1)}
        style={{
          backgroundColor: '#61DAFB',
          color: '#282C34',
          padding: '10px 20px',
          borderRadius: '4px'
        }}
      >
        Decrementar
      </button>
    </div>
  );
}
```

Ciclo de Vida e useEffect

Ciclo de Vida dos Componentes

Componentes React passam por três fases principais durante sua existência:

Montagem | Atualização | Desmontagem

Hook useEffect

O useEffect permite executar efeitos colaterais em componentes funcionais, como:

- Chamadas de API
- Manipulação do DOM
- Configuração de timers
- Limpeza de recursos

O segundo parâmetro (array de dependências) controla quando o efeito é executado.

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

function Timer() {
  const [seconds, setSeconds] = useState(0);

  // Executa após cada renderização
  useEffect(() => {
    const interval = setInterval(() => {
      setSeconds(prev => prev + 1);
    }, 1000);

    // Função de limpeza
    return () => {
      clearInterval(interval);
    };
  }, []); // Array vazio = executa apenas na montagem

  return (
    <div>
      <h2>Timer: {seconds}s</h2>
    </div>
  );
}
```

Estrutura de um Projeto React

Create React App

A maneira mais fácil de iniciar um novo projeto React é usando o Create React App:

```
npx create-react-app meu-app
```

Arquivos Importantes

- **index.js**: Ponto de entrada da aplicação
- **App.js**: Componente principal
- **package.json**: Dependências e scripts

Scripts Disponíveis

- `npm start`: Inicia o servidor de desenvolvimento
- `npm run build`: Cria versão de produção

```
meu-app/
  ├── node_modules/
  ├── public/
  ├── index.html
  └── favicon.ico
  ├── src/
  ├── index.js
  ├── App.js
  └── components/
  └── package.json
```

```
// src/index.js - Ponto de entrada
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(
  document.getElementById('root')
);

root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

Trabalhando com Eventos

Eventos no React

Eventos no React são muito semelhantes aos eventos do DOM, mas com algumas diferenças sintáticas:

- Eventos são nomeados usando camelCase (onClick, onChange)
- Funções são passadas como manipuladores de eventos
- Você não pode retornar false para evitar o comportamento padrão

Eventos Comuns

onClick

Acionado quando um elemento é clicado

onChange

Acionado quando o valor de um elemento é alterado

onSubmit

Acionado quando um formulário é enviado

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

function ToggleButton() {
  const [isOn, setIsOn] = useState(false);

  const handleClick = () => {
    setIsOn(!isOn);
  };

  return (
    <div>
      <button
        onClick={handleClick}
        className={isOn ? 'active' : ''}
      >
        {isOn ? 'Desligar' : 'Ligar'}
      </button>

      {isOn && (
        <p>O botão está ligado!</p>
      )}
    </div>
  );
}
```

Renderização Condicional

O que é Renderização Condicional?

É a capacidade de exibir diferentes elementos ou componentes com base em condições específicas.

Técnicas Comuns

Operador Ternário

Ideal para condições simples de verdadeiro/falso:

`{condição ? <ElementoA /> : <ElementoB />}`

Operador &&

Para renderizar algo apenas quando a condição é verdadeira:

`{condição && <Elemento />}`

Declarações if/else

Para lógica mais complexa, use fora do JSX:

```
if (condição) { return <ElementoA />; }
else { return <ElementoB />; }
```

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

function LoginStatus() {
  const [isLoggedIn, setIsLoggedIn] = useState(false);

  const toggleLogin = () => {
    setIsLoggedIn(!isLoggedIn);
  };

  return (
    <div>
      <h2>Status do Usuário</h2>

      // Usando operador ternário
      {isLoggedIn
        ? <p className="text-green-400">Bem-vindo!</p>
        : <p className="text-red-400">Por favor, faça login.</p>
      }

      // Usando operador &&
      {isLoggedIn && (
        <button>Ver Perfil</button>
      )}

      <button onClick={toggleLogin}>
        {isLoggedIn ? 'Logout' : 'Login'}
      </button>
    </div>
  );
}
```

Renderizando Listas

Renderização de Coleções

Para renderizar listas de elementos em React, usamos o método `map()` para transformar arrays de dados em arrays de elementos JSX.

A Importância da Propriedade key

Cada elemento em uma lista precisa de uma propriedade `key` única para ajudar o React a identificar quais itens foram alterados.

> **Boas Práticas para Keys**
>
> - Use IDs estáveis dos seus dados
> - Evite usar índices do array como keys

```jsx
import React from 'react';

function ItemList() {
  const items = [
    { id: 1, name: 'React' },
    { id: 2, name: 'JavaScript' },
    { id: 3, name: 'HTML/CSS' }
  ];

  return (
    <div>
      <h2>Tecnologias Web</h2>

      // Renderizando lista com keys
      <ul>
        {items.map(item => (
          <li key={item.id}>
            {item.name}
          </li>
        ))}
      </ul>

      // Exemplo com componente
      <div className="mt-4">
        {items.map(item => (
          <TechItem
            key={item.id}
            name={item.name}
          />
        ))}
      </div>
    </div>
  );
}
```

Ecossistema e Próximos Passos

Ferramentas e Bibliotecas Populares

React Router

Navegação e roteamento para aplicações React

Redux

Gerenciamento de estado global

Material-UI / Chakra UI

Bibliotecas de componentes UI

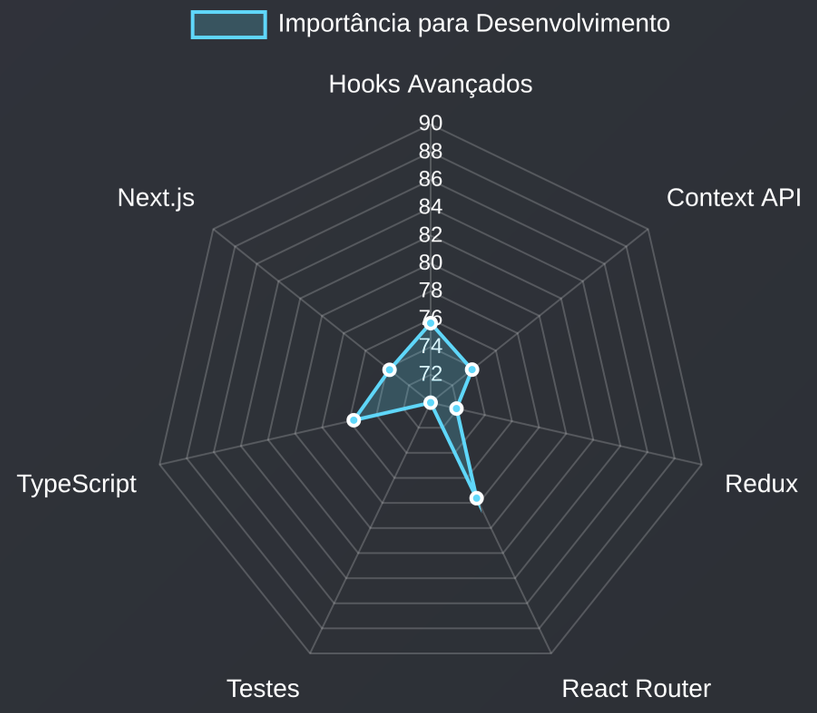
React Testing Library

Testes de componentes React

Recursos para Aprendizado

- Documentação oficial do React
- Cursos online (Udemy, Coursera, etc.)
- Comunidade React Brasil
- GitHub e projetos de código aberto

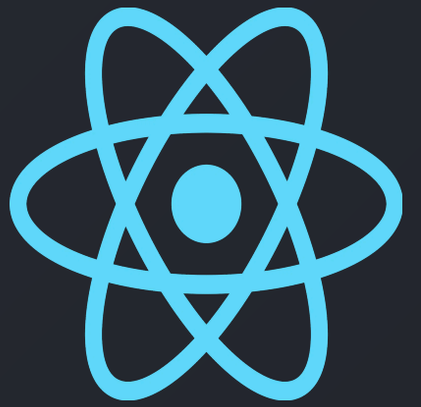
Próximos Tópicos para Aprender



Conclusão

Conceitos Fundamentais Abordados

- Componentes e JSX
- Props e passagem de dados
- State e gerenciamento de estado
- Ciclo de vida e useEffect
- Eventos e interatividade
- Renderização condicional e listas



Próximos Passos

Para continuar seu aprendizado em React, explore:

- Documentação oficial do React
- Projetos práticos para consolidar conhecimento
- Comunidade React e fóruns de discussão

Dicas para Praticar

- Comece com projetos pequenos e focados
- Recrie interfaces de sites que você conhece
- Explore bibliotecas complementares (React Router, Redux)
- Participe de desafios de código

Lembre-se: A prática constante é a chave para dominar o React!

Obrigado! Alguma pergunta?