Integração Backend + Frontend com React

Desenvolvimento Web 2 · 4° Semestre · Análise e Desenvolvimento de Sistemas

Agenda da Aula

01

Introdução e Instalação

Configuração do ambiente React

02

Componentes Básicos

Estrutura e criação de componentes

03

Props e Comunicação

Passagem de dados entre componentes

04

Eventos e Interatividade

Manipulação de eventos do usuário

05

Estado com useState

Gerenciamento de estado dinâmico

06

Renderização Condicional

Exibição de conteúdo baseada em condições

07

Renderização de Listas

Exibição de dados em arrays

08

Navegação com React Router

Implementação de rotas na aplicação

Objetivos de Aprendizagem

Dominar fundamentos do React

Compreender a arquitetura baseada em componentes do React e como ela facilita o desenvolvimento de interfaces modernas e escaláveis.

Criar Componentes Reutilizáveis

Desenvolver componentes modulares que podem ser reutilizados em diferentes partes da aplicação, seguindo as melhores práticas.

Integrar Frontend e Backend

Estabelecer comunicação eficiente entre a interface React e os dados do backend, simulando cenários reais de desenvolvimento.

Por que React?

Popularidade no Mercado

React é uma das bibliotecas JavaScript mais utilizadas no mundo, com mais de 200 mil estrelas no GitHub e adoção massiva por empresas como Facebook, Netflix, Instagram e Airbnb.

Vantagens Técnicas

- Virtual DOM para performance otimizada
- Componentização e reutilização de código
- Ecossistema rico de bibliotecas
- Fácil integração com APIs REST
- Community support extenso

Instalação e Configuração

Instalação do React

Para começar nosso projeto React, utilizaremos o Create React App, que configura automaticamente um ambiente de desenvolvimento otimizado.

```
npx create-react-app meu-projeto-frontend
cd meu-projeto-frontend
npm start
```

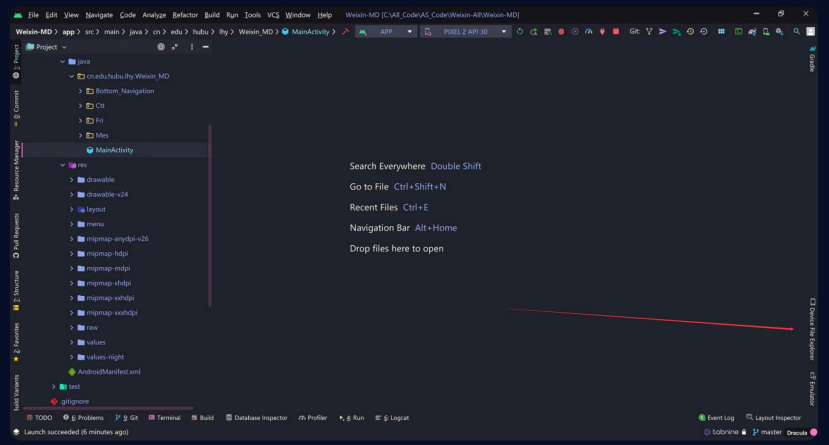
Dica: O comando npx executa o Create React App sem necessidade de instalação global, garantindo sempre a versão mais recente.

Estrutura do Projeto React

Arquivos e Pastas Principais

- **src/** - Código fonte da aplicação
- **public/** - Arquivos públicos e index.html
- **package.json** - Dependências e scripts
- **App.js** - Componente principal
- **index.js** - Ponto de entrada da aplicação

Esta estrutura organizada facilita a manutenção e escalabilidade do projeto, seguindo convenções estabelecidas pela comunidade React.



Primeiro Olhar no App.js

O arquivo App.js é o coração da nossa aplicação React. Vamos analisar sua estrutura básica:

```
import React from 'react';
import './App.css';

function App() {
  return (
    <div className="App">
      <header className="App-header">
        <h1>Minha Aplicação React</h1>
        <p>Bem-vindos ao desenvolvimento frontend!</p>
      </header>
    </div>
  );
}

export default App;
```

Note que utilizamos JSX, uma extensão de sintaxe que permite escrever HTML dentro do JavaScript de forma mais intuitiva.

Componentes Básicos

O que são Componentes?

Definição

Componentes são **funções JavaScript** que retornam elementos JSX. Eles são os blocos de construção fundamentais de uma aplicação React.

Reutilização

Um componente criado uma vez pode ser utilizado múltiplas vezes em diferentes partes da aplicação, promovendo **código limpo** e **manutenível**.

Isolamento

Cada componente possui sua própria lógica e apresentação, facilitando o **debugging** e a **manutenção** da aplicação.

Criando Nosso Primeiro Componente

Vamos criar um componente simples chamado **Cabecalho** em um arquivo separado:

```
// src/components/Cabecalho.js
import React from 'react';

function Cabecalho() {
  return (
    <header style={{backgroundColor: '#EFF3F2', padding: '20px'}}>
      <h1>Sistema Acadêmico</h1>
      <p>Gerenciamento de Alunos e Cursos</p>
    </header>
  );
}

export default Cabecalho;
```

Convenção: Nomes de componentes sempre começam com letra maiúscula para diferenciá-los de elementos HTML nativos.

Utilizando o Componente no App.js

Agora vamos importar e utilizar nosso componente Cabecalho no componente principal:

```
// src/App.js
import React from 'react';
import Cabecalho from './components/Cabecalho';
import './App.css';

function App() {
  return (
    <div className="App">
      <Cabecalho />
      <main>
        <p>Conteúdo principal da aplicação</p>
      </main>
    </div>
  );
}

export default App;
```

Observe como o componente é utilizado como uma tag HTML personalizada, demonstrando a elegância e simplicidade do React.

Props

Comunicação entre Componentes

Entendendo Props

1

Componente Pai

Envia dados através de propriedades (props) para componentes filhos.

2

Componente Filho

Recebe e utiliza os dados enviados pelo componente pai através das props.

As props funcionam como **parâmetros de função**, permitindo que os componentes sejam dinâmicos e reutilizáveis. O fluxo de dados é sempre **unidirecional**: pai → filho.

Criando o Componente Aluno

Vamos criar um componente que recebe informações de um aluno através de props:

```
// src/components/Aluno.js
import React from 'react';

function Aluno(props) {
  return (
    <div>
      <h3>{props.nome}</h3>
      <p><strong>Matrícula:</strong> {props.matricula}</p>
      <p><strong>Curso:</strong> {props.curso}</p>
      <p><strong>Semestre:</strong> {props.semestre}º</p>
    </div>
  );
}

export default Aluno;
```

Note como utilizamos `{props.propriedade}` para acessar os dados passados pelo componente pai.

Utilizando Props no Componente Pai

Agora vamos utilizar o componente Aluno, passando diferentes dados através das props:

```
// src/App.js
import React from 'react';
import Cabecalho from './components/Cabecalho';
import Aluno from './components/Aluno';

function App() {
 return (
 <div className="App">
 <Cabecalho />
 <main>
 <Aluno nome="Maria Silva" matricula="2024001"
 curso="Análise e Desenvolvimento de Sistemas" semestre={4}
 />
 <Aluno nome="João Santos" matricula="2024002"
 curso="Ciência da Computação" semestre={6}
 />
 </main>
 </div>
 );
}
```

Destructuring de Props

Uma prática comum e mais limpa é usar destructuring para extrair as props:

Forma Tradicional

```
function Aluno(props) {
  return (
    <div>
      <h3>{props.nome}</h3>
      <p>{props.curso}</p>
    </div>
  );
}
```

Com Destructuring

```
function Aluno({nome, matricula, curso, semestre}) {
  return (
    <div>
      <h3>{nome}</h3>
      <p>{curso}</p>
    </div>
  );
}
```

O destructuring torna o código mais legível e facilita a identificação das props utilizadas pelo componente.

Eventos

Interatividade no React

Manipulação de Eventos no React

Sintaxe camelCase

Eventos em React seguem a convenção camelCase: `onClick`, `onChange`, `onSubmit`.

Funções JavaScript

Eventos recebem funções JavaScript como valor, não strings como no HTML tradicional.

SyntheticEvent

React encapsula eventos nativos em SyntheticEvents, garantindo compatibilidade entre navegadores.

Exemplo Prático: Botão com Evento

Vamos criar um componente com um botão que responde a cliques:

```
// src/components/BotaoInterativo.js
import React from 'react';

function BotaoInterativo() {
 const handleClick = () => {
 alert('Botão foi clicado!');
 console.log('Evento de clique executado');
 };

 const handleMouseEnter = () => {
 console.log('Mouse entrou no botão');
 };

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

export default BotaoInterativo;
```

Eventos com Parâmetros

Frequentemente precisamos passar parâmetros para as funções de evento. Veja como fazer isso:

```
// src/components/ListaBotoes.js
import React from 'react';

function ListaBotoes() {
  const handleClick = (nomeAluno) => {
    alert(`Aluno selecionado: ${nomeAluno}`);
  };

  return (
    <div>
      <h3>Lista de Alunos</h3>
      <button onClick={() => handleClick('Maria Silva')}>
        Selecionar Maria
      </button>
      <button onClick={() => handleClick('João Santos')}>
        Selecionar João
      </button>
      <button onClick={() => handleClick('Ana Costa')}>
        Selecionar Ana
      </button>
    </div>
  );
}

export default ListaBotoes;
```

Atenção: Use arrow functions quando precisar passar parâmetros para evitar execução imediata da função.

useState

Gerenciamento de Estado

Estado vs Props

Props (Propriedades)

- Dados **externos** ao componente
- Recebidos do componente pai
- São **imutáveis** (read-only)
- Permitem comunicação pai → filho

Estado (State)

- Dados **internos** do componente
- Gerenciados pelo próprio componente
- São **mutáveis** e dinâmicos
- Controlam comportamento e renderização

Introdução ao useState Hook

O `useState` é um Hook que permite adicionar estado a componentes funcionais:

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

function Contador() {
 // Declaração do estado
 const [contador, setContador] = useState(0);

 const incrementar = () => {
 setContador(contador + 1);
 };

 const decrementar = () => {
 setContador(contador - 1);
 };

 return (
 <div style={{textAlign: 'center', padding: '20px'}}>
 <h2>Contador: {contador}</h2>
 <button onClick={incrementar}>+1</button>
 <button onClick={decrementar}>-1</button>
 </div>
 );
}

export default Contador;
```

Anatomia do useState

01

Importação

`import { useState } from 'react'` - Importamos o Hook do React

02

Declaração

`const [estado, setEstado] = useState(valorInicial)` - Declaramos o estado com valor inicial

03

Leitura

`{estado}` - Acessamos o valor atual do estado no JSX

04

Atualização

`setEstado(novoValor)` - Atualizamos o estado, causando re-renderização

Exemplo Prático: Formulário de Aluno

Vamos criar um formulário que gerencia o estado dos campos de entrada:

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

function FormularioAluno() {
 const [nome, setNome] = useState('');
 const [matricula, setMatricula] = useState('');
 const [curso, setCurso] = useState('');

 const handleSubmit = (e) => {
 e.preventDefault();
 console.log('Dados do aluno:', { nome, matricula, curso });
 alert(`Aluno ${nome} cadastrado com sucesso!`);
 };

...

}
```

Exemplo Prático: Formulário de Aluno

Vamos criar um formulário que gerencia o estado dos campos de entrada:

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

function FormularioAluno() {
...

 return (
 <form onSubmit={handleSubmit} style={{padding: '20px'}}>
 <div>
 <label>Nome: </label>
 <input
 type="text"
 value={nome}
 onChange={(e) => setNome(e.target.value)}
 />
 </div>
 <div>
 <label>Matrícula: </label>
 <input
 type="text"
 value={matricula}
 onChange={(e) => setMatricula(e.target.value)}
 />
 </div>
 <button type="submit">Cadastrar</button>
 </form>
 );
}
```

Estados Complexos: Objetos

Podemos usar useState com objetos para gerenciar múltiplas propriedades:

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

function PerfilAluno() {
 const [aluno, setAluno] = useState({
 nome: '',
 email: '',
 idade: '',
 curso: 'ADS'
 });

 const handleInputChange = (campo, valor) => {
 setAluno(prevAluno => ({
 ...prevAluno,
 [campo]: valor
 }));
 };

 return (
 <div>
 <h3>Perfil do Aluno</h3>
 <input
 placeholder="Nome"
 value={aluno.nome}
 onChange={(e) => handleInputChange('nome', e.target.value)}
 />
 <input
 placeholder="Email"
 value={aluno.email}
 onChange={(e) => handleInputChange('email', e.target.value)}
 />
 <p>Dados atuais: {JSON.stringify(aluno)}</p>
 </div>
 );
}
```

Estados Complexos: Objetos

Podemos usar useState com objetos para gerenciar múltiplas propriedades:

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

function PerfilAluno() {
 const [aluno, setAluno] = useState({
 nome: '',
 email: '',
 idade: '',
 curso: 'ADS'
 });

 const handleInputChange = (campo, valor) => {
 setAluno(prevAluno => ({
 ...prevAluno,
 [campo]: valor
 }));
 };

...
}
```

Estados Complexos: Objetos

Podemos usar useState com objetos para gerenciar múltiplas propriedades:

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

function PerfilAluno() {
...

 return (
 <div>
 <h3>Perfil do Aluno</h3>
 <input
 placeholder="Nome"
 value={aluno.nome}
 onChange={(e) => handleInputChange('nome', e.target.value)}
 />
 <input
 placeholder="Email"
 value={aluno.email}
 onChange={(e) => handleInputChange('email', e.target.value)}
 />
 <p>Dados atuais: {JSON.stringify(aluno)}</p>
 </div>
 );
}
```

Renderização Condicional

O que é Renderização Condicional?

A renderização condicional permite exibir diferentes conteúdos baseados no estado ou props do componente. É uma técnica fundamental para criar interfaces dinâmicas e responsivas.

Operador Ternário

`condição ? elementoSeVerdadeiro : elementoSeFalso`

Operador &&

`condição && elementoParaExibir`

If/Else Tradicional

Usando estruturas condicionais antes do return

Exemplo: Login/Logout

Vamos criar um componente que mostra diferentes interfaces baseado no estado de login:

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

function AreaUsuario() {
 const [usuarioLogado, setUsuarioLogado] = useState(false);
 const [nomeUsuario, setNomeUsuario] = useState('');

 const fazerLogin = () => {
 setNomeUsuario('João Silva');
 setUsuarioLogado(true);
 };

 const fazerLogout = () => {
 setNomeUsuario('');
 setUsuarioLogado(false);
 };
}
```

Exemplo: Login/Logout

Vamos criar um componente que mostra diferentes interfaces baseado no estado de login:

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

function AreaUsuario() {
...

 return (
 <div style={{padding: '20px', border: '1px solid #EFF3F2'}}>
 {usuarioLogado ? (
 <div>
 <h3>Bem-vindo, {nomeUsuario}!</h3>
 <p>Você está logado no sistema.</p>
 <button onClick={fazerLogout}>Logout</button>
 </div>
 ) : (
 <div>
 <h3>Faça seu login</h3>
 <p>Você precisa estar logado para acessar o sistema.</p>
 <button onClick={fazerLogin}>Login</button>
 </div>
 )}
 </div>
 );
}
```

Renderização com Operador &&

Para exibir elementos apenas quando uma condição é verdadeira, usamos o operador &&:

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

function Notificacoes() {
 const [mostrarAlerta, setMostrarAlerta] = useState(true);
 const [notificacoes, setNotificacoes] = useState(3);

 return (
 <div>
 <h3>Painel de Notificações</h3>

 {mostrarAlerta && (
 <div style={{
 backgroundColor: '#ffeb3b',
 padding: '10px',
 marginBottom: '10px'
 }}>
```

React Router

Navegação entre Páginas

Instalando React Router DOM

Para implementar navegação em nossa aplicação React, utilizamos o React Router DOM:

```
npm install react-router-dom
```

O React Router DOM permite criar aplicações **Single Page Applications (SPA)** com navegação fluida entre diferentes "páginas" sem recarregar o navegador.

1. ### Instalação da biblioteca
   Comando npm install para adicionar a dependência

2. ### Configuração das rotas
   Definição dos caminhos e componentes correspondentes

3. ### Implementação da navegação
   Criação de links e menus para transição entre páginas

Estrutura Básica de Rotas

Vamos configurar o sistema de rotas no componente principal da aplicação:

```
// src/App.js
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Navbar from './components/Navbar';
import Home from './pages/Home';
import Alunos from './pages/Alunos';
import Cursos from './pages/Cursos';
import Sobre from './pages/Sobre';

function App() {
 return (
 <Router>
 <div className="App">
 <Navbar />
 <main style={{padding: '20px'}}>
 <Routes>
 <Route path="/" element={<Home />} />
 <Route path="/alunos" element={<Alunos />} />
 <Route path="/cursos" element={<Cursos />} />
 <Route path="/sobre" element={<Sobre />} />
 </Routes>
 </main>
 </div>
 </Router>
 );
}

export default App;
```

Criando Componente de Navegação

Vamos criar um componente Navbar com links de navegação:

```
// src/components/Navbar.js
import React from 'react';
import { Link, useLocation } from 'react-router-dom';

function Navbar() {
 const location = useLocation();

 const linkStyle = (path) => ({
 padding: '10px 15px',
 textDecoration: 'none',
 color: location.pathname === path ? '#fff' : '#333',
 backgroundColor: location.pathname === path ? '#EFF3F2' : 'transparent',
 borderRadius: '4px',
 margin: '0 5px'
 });

 return (
 <nav s>
 <div style={{display: 'flex', alignItems: 'center'}}>
 <h2 style={{margin: '0 20px 0 0'}}>Sistema Acadêmico</h2>
 <Link to="/" style={linkStyle('/')}>Home</Link>
 <Link to="/alunos" style={linkStyle('/alunos')}>Alunos</Link>
 <Link to="/cursos" style={linkStyle('/cursos')}>Cursos</Link>
 <Link to="/sobre" style={linkStyle('/sobre')}>Sobre</Link>
 </div>
 </nav>
 );
}

export default Navbar;
```

Criando as Páginas

Agora vamos criar as páginas que serão exibidas para cada rota:

Home.js

```
// src/pages/Home.js
import React from 'react';

function Home() {
  return (
    <div>
      <h1>Bem-vindos ao Sistema Acadêmico</h1>
      <p>Este é um sistema completo para
      gerenciar alunos e cursos.</p>
      <div style={{
        backgroundColor: '#EFF3F2',
        padding: '20px',
        borderRadius: '8px',
        margin: '20px 0'
      }}>
        <h3>Funcionalidades:</h3>
        <ul>
          <li>Cadastro de alunos</li>
          <li>Gerenciamento de cursos</li>
          <li>Relatórios acadêmicos</li>
        </ul>
      </div>
    </div>
  );
}

export default Home;
```

Alunos.js

```
// src/pages/Alunos.js
import React from 'react';
import ListaAlunosAvancada from '../components/ListaAlunosAvancada';

function Alunos() {
 return (
 <div>
 <h1>Gerenciamento de Alunos</h1>
 <p>Visualize e gerencie todos os
 alunos cadastrados no sistema.</p>
 <ListaAlunosAvancada />
 </div>
 );
}

export default Alunos;
```

Navegação Programática

Além dos links, podemos navegar programaticamente usando o hook `useNavigate`:

```
// src/components/BotaoNavegacao.js
import React from 'react';
import { useNavigate } from 'react-router-dom';

function BotaoNavegacao() {
 const navigate = useNavigate();

 const irParaAlunos = () => {
 // Lógica adicional aqui
 console.log('Redirecionando para página de alunos...');
 navigate('/alunos');
 };

 const voltarPagina = () => {
 navigate(-1); // Volta uma página no histórico
 };

...

export default BotaoNavegacao;
```

Navegação Programática

Além dos links, podemos navegar programaticamente usando o hook `useNavigate`:

```
// src/components/BotaoNavegacao.js
import React from 'react';
import { useNavigate } from 'react-router-dom';

function BotaoNavegacao() {
...
 return (
 <div style={{margin: '20px 0'}}>
 <button
 onClick={irParaAlunos}
 >
 Ver Alunos
 </button>

 <button
 onClick={voltarPagina}
 >
 Voltar
 </button>
 </div>
 );
}

export default BotaoNavegacao;
```

Renderização de Listas

Renderizando Lista Básica

Vamos criar um componente que exibe a lista de alunos usando o método `map()`:

```
import React from 'react';
import { alunos } from '../models/dadosAlunos';

function ListaAlunos() {
  const [alunos, setAlunos] = useState([]); // estado para lista
  const navigate = useNavigate();

  // carregar alunos ao montar o componente
  useEffect(() => {
    fetch("http://localhost:3000/alunos")
      .then((res) => res.json())
      .then((data) => setAlunos(data))
      .catch((err) => console.error("Erro ao carregar alunos:", err));
  }, []);

// função para deletar aluno
  function deletarAluno(id) {
    fetch(`http://localhost:3000/alunos/${id}`, {
      method: "DELETE"
    })
      .then((res) => res.json())
      .then(() => {
        // atualiza estado removendo o aluno sem precisar recarregar
        setAlunos(alunos.filter((a) => a.id !== id));
      })
      .catch((err) => console.error("Erro ao deletar aluno:", err));
  }
...

export default ListaAlunos;
```

Renderizando Lista Básica

Vamos criar um componente que exibe a lista de alunos usando o método `map()`:

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";

function ListaAlunos() {
...
 return (
   <div>
    <h2>Lista de Alunos</h2>

    <button onClick={() => navigate("/aluno/novo")}>
      Cadastrar Novo
    </button>

    <ul>
      {alunos.map((a) => (
        <li key={a.id}>
          {a.nome}
          {" "}
          <button onClick={() => navigate(`/aluno/${a.id}`)}>
            Ver/Editar
          </button>
          <button onClick={() => deletarAluno(a.id)}>
            Deletar
          </button>
        </li>
      ))}
    </ul>
   </div>
  );
}
export default ListaAlunos;
```

A Importância da Prop Key

Identificação Única

A prop `key` ajuda o React a identificar quais itens mudaram, foram adicionados ou removidos da lista.

Performance

Com keys adequadas, o React pode otimizar a re-renderização, atualizando apenas os elementos que realmente mudaram.

Boas Práticas

Use IDs únicos e estáveis. Evite usar índices do array como key quando a ordem pode mudar.

Importante: Nunca use o índice do map como key em listas que podem ser reordenadas ou ter itens removidos/adicionados.

Lista com Componente Personalizado

Vamos refatorar para usar um componente dedicado para cada item da lista:

```
// src/components/CartaoAluno.js
import React from 'react';

function CartaoAluno({ aluno }) {
 return (
 <div>
 <div style={{display: 'flex', justifyContent: 'space-between'}}>
 <h4>{aluno.nome}</h4>
 <span style={{
 color: aluno.ativo ? 'green' : 'red',
 fontWeight: 'bold'
 }}>
 {aluno.ativo ? '
```

Lista Filtrada e Interativa

Vamos criar uma lista com funcionalidade de busca e filtro:

```
import React, { useState } from 'react';
import { alunos } from '../models/dadosAlunos';
import CartaoAluno from './CartaoAluno';

function ListaAlunosAvancada() {
 const [filtro, setFiltro] = useState('');
 const [mostrarApenasAtivos, setMostrarApenasAtivos] = useState(false);

 const alunosFiltrados = alunos.filter(aluno => {
 const passaNomeFiltro = aluno.nome.toLowerCase()
 .includes(filtro.toLowerCase());
 const passaFiltroAtivo = mostrarApenasAtivos ? aluno.ativo : true;

 return passaNomeFiltro && passaFiltroAtivo;
 });

 return (
 <div style={{padding: '20px'}}>
 <h2>Lista de Alunos - Avançada</h2>

 <div style={{marginBottom: '20px'}}>
 <input
 type="text"
 placeholder="Buscar por nome..."
 value={filtro}
 onChange={(e) => setFiltro(e.target.value)}
 style={{padding: '8px', marginRight: '10px'}}
 />

 <label>
 <input
 type="checkbox"
 checked={mostrarApenasAtivos}
 onChange={(e) => setMostrarApenasAtivos(e.target.checked)}
 />
 Mostrar apenas ativos
 </label>
 </div>

 <p>Exibindo {alunosFiltrados.length} de {alunos.length} alunos</p>

 {alunosFiltrados.map(aluno => (
 <CartaoAluno key={aluno.id} aluno={aluno} />
 ))}
 </div>
 );
}
```