

ReactJS

Instalação

Com o Node Js $\geq 14.0.0$ LTS instalado execute o comando para criar uma nova aplicação reactjs.

No diretório raiz execute o seguinte comando:

```
npx create-react-app my-app
```

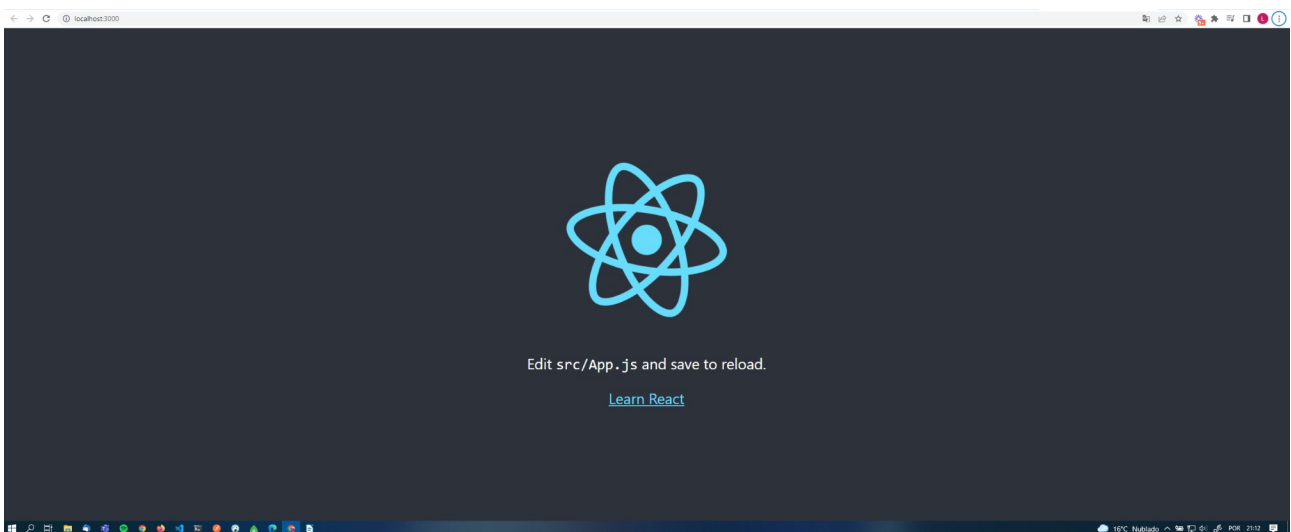
Onde a palavra **my-app** seria o nome da sua aplicação.

Executar aplicação

Depois que o processo finalizar, entre na raiz do projeto e execute o seguinte comando.

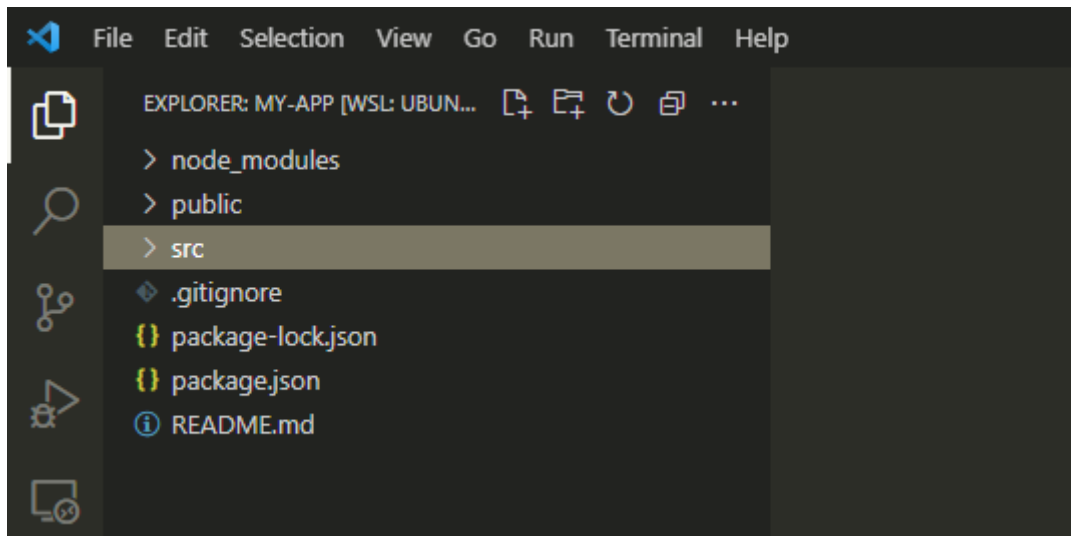
```
npm start
```

O site ficará disponível em uma página local em seu navegador, basta acessar a seguinte url <http://localhost:3000> e você verá o seguinte conteúdo.



Diretórios da aplicação

Dentro da aplicação temos diversos diretórios, mas o mais importante nesse momento é o diretório **src**, esse diretório conterá todos os arquivos de nosso projeto.



O arquivo inicial da aplicação é o arquivo App.js, começaremos nosso CRUD de usuários nele.

```
JS App.js x
1 import logo from './logo.svg';
2 import './App.css';
3
4 function App() {
5   return (
6     <div className="App">
7       <header className="App-header">
8         <img src={logo} className="App-logo" alt="logo" />
9         <p>
10           Edit <code>src/App.js</code> and save to reload.
11         </p>
12         <a
13           className="App-link"
14           href="https://reactjs.org"
15           target="_blank"
16           rel="noopener noreferrer"
17         >
18           Learn React
19         </a>
20       </header>
21     </div>
22   );
23 }
24
25 export default App;
26
```

Note que nesse arquivo temos uma função javascript e essa função retorna um html comum, ou seja, para se ter uma tela em reactjs, basicamente temos que ter uma função javascript que retorna um html.

Observação: No react, todas as paginas devem retornar uma única tag html.

Ex certo

```
1  function Exemplo()  
2  {  
3      return (  
4          <div>  
5              {/* conteudo */}  
6          </div>  
7      )  
8  }
```

Exemplo errado

```
1  function Exemplo()  
2  {  
3      return (  
4          <div>  
5              {/* conteudo */}  
6          </div>  
7          <div>  
8              {/* conteudo */}  
9          </div>  
10     )  
11 }
```

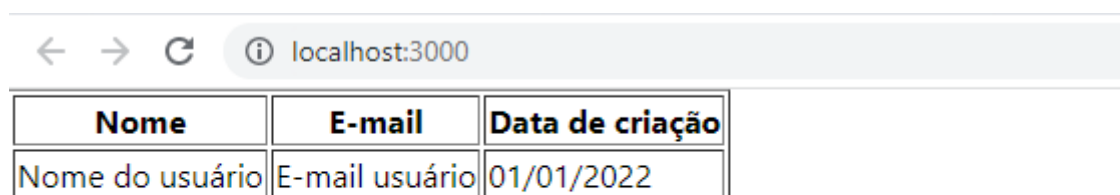
Devemos seguir essa regra básica, para que o compilador do react não se perca.

Grid de usuários

Em nossa primeira página iremos criar uma tabela simples que terá nossa listagem de usuários, para isso remova todo o html que temos em nossa página inicial e coloque uma tabela comum, para exibir uma lista de usuarios com as seguintes colunas (nome, e-mail, data de criação)

```
JS App.js M X
1 function App() {
2   return (
3     <div>
4       <table border="1">
5         <thead>
6           <tr>
7             <th>Nome</th>
8             <th>E-mail</th>
9             <th>Data de criação</th>
10          </tr>
11        </thead>
12        <tbody>
13          <tr>
14            <td>Nome do usuário</td>
15            <td>E-mail usuário</td>
16            <td>01/01/2022</td>
17          </tr>
18        </tbody>
19      </table>
20    </div>
21  );
22 }
23
24 export default App;
25
```

Feito isso você deve ter esse resultado.



The screenshot shows a web browser window with the address bar displaying 'localhost:3000'. Below the address bar, there is a table with three columns: 'Nome', 'E-mail', and 'Data de criação'. The first row of data contains the values 'Nome do usuário', 'E-mail usuário', and '01/01/2022'.

Nome	E-mail	Data de criação
Nome do usuário	E-mail usuário	01/01/2022

Como teremos uma lista dinamica de usuarios vamos deixar ele preparado para receber essa lista dinamica, que a principio será um array simples.

Antes de retornar o seu html, inclua o seguinte array javascript.

```
function App() {  
  const lista_de_usuarios = [  
    { name: "Usuario 1", email: "usuario1@email.com", created_at: "2022-01-01 00:00:00" },  
    { name: "Usuario 2", email: "usuario2@email.com", created_at: "2022-01-01 00:00:00" },  
    { name: "Usuario 3", email: "usuario3@email.com", created_at: "2022-01-01 00:00:00" },  
  ]  
}
```

Note que esse array é um javascript comum, ou seja, o react permite qualquer código javascript que seja necessário utilizar.


Dito isso utilizaremos uma função javascript comum a função **map()**.

Para mais detalhes da função acesse o link abaixo.

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Para utilizarmos javascript dentro do html do react, ele deve estar entre chaves {}

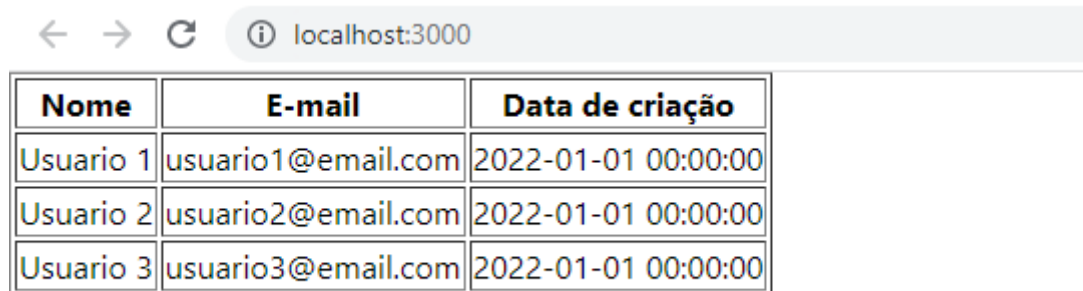
```
JS App.js M X  
1 function App() {  
2  
3   const lista_de_usuarios = [  
4     { name: "Usuario 1", email: "usuario1@email.com", created_at: "2022-01-01 00:00:00" },  
5     { name: "Usuario 2", email: "usuario2@email.com", created_at: "2022-01-01 00:00:00" },  
6     { name: "Usuario 3", email: "usuario3@email.com", created_at: "2022-01-01 00:00:00" },  
7   ]  
8  
9   return (  
10     <div>  
11       <table border="1">  
12         <thead>  
13           <tr>  
14             <th>Nome</th>  
15             <th>E-mail</th>  
16             <th>Data de criação</th>  
17           </tr>  
18         </thead>  
19         <tbody>  
20           { lista_de_usuarios.map( usuario => {  
21             return <tr>  
22               <td>{ usuario.name }</td>  
23               <td>{ usuario.email }</td>  
24               <td>{ usuario.created_at }</td>  
25             </tr>  
26           } ) }  
27         </tbody>  
28       </table>  
29     </div>  
30   );  
31 }  
32  
33 export default App;  
34
```



Note que dentro da função map utilizamos o return, isso deve ser feito, pois a cada loop do map, devemos retornar um conteúdo html, caso contrário não conseguiremos visualizar a lista de usuários. Outro detalhe importante é que para exibirmos o conteúdo do objeto javascript estamos utilizando uma interpolação em chaves { **variável** }

No react utilizamos uma interpolação simples toda vez que desejamos printar um conteúdo de uma variável no navegador.

Ao atualizar o conteúdo devemos ter esse resultado.



A screenshot of a web browser window. The address bar shows 'localhost:3000'. Below the address bar is a table with three columns: 'Nome', 'E-mail', and 'Data de criação'. The table contains three rows of user data.

Nome	E-mail	Data de criação
Usuario 1	usuario1@email.com	2022-01-01 00:00:00
Usuario 2	usuario2@email.com	2022-01-01 00:00:00
Usuario 3	usuario3@email.com	2022-01-01 00:00:00

Consultando uma lista de usuários

Normalmente o react trabalha sempre na camada visual do sistema (frontend), quando estamos trabalhando com dados dinamicos, normalmente precisaremos de uma API que nos permita consultar/modificar determinada informação.

No nosso caso utilizaremos uma API simples previamente construida, onde nos permitirá cadastrar, editar, consultar e deletar usuários.

Para se comunicar com essa api, utilizaremos uma lib especializada em processos http, a lib no caso é a **axios**.

Para mais detalhes acesse o link abaixo

<https://axios-http.com/>

Instalando uma lib no react

Para instalar uma lib é simples, dentro do seu diretorio raiz do projeto execute o seguinte comando

```
npm install --save nome-da-lib
```

No caso iremos instala a lib axios, então execute o seguinte comando

```
npm install --save axios
```

Após o processo de instalação deve-se reiniciar o projeto react. Para fazer isso acesse o terminal aonde foi executado o comando inicial **npm start** e execute o atalho **ctrl + c** após isso execute novamente o comando **npm start**.

Executando comando ao carregar a página

Ao carregarmos nossa página, precisaremos consultar nossa api e alimentar nossa variavel de usuarios.

Para isso utilizaremos os hooks (são funções nativas do react), no caso utilizaremos 2 hooks

- `useEffect`

- `useState`

useEffect

Esse hook será executado toda vez que a tela for carregada, ou seja, utilizaremos ele para carregar nossa listagem inicial de usuarios.

useState

Como teremos uma variavel que será dinamica, devemos utilizar esse hook, pois ele é responsável por cuidar da atualização em tempo real dessa variavel.

Atualizando nosso sistema

Aplique esse código onde antes tínhamos nosso array simples



```
JS App.js M X
1  import axios from "axios";
2  import { useEffect, useState } from "react";
3
4  function App() {
5
6      const [usuarios, setUsuarios] = useState([]);
7
8      useEffect(() => {
9          axios.get('http://localhost:8000/users').then(response => {
10              setUsuarios(response.data.users);
11          }).catch(error => console.log(error));
12      }, []);
13  }
```

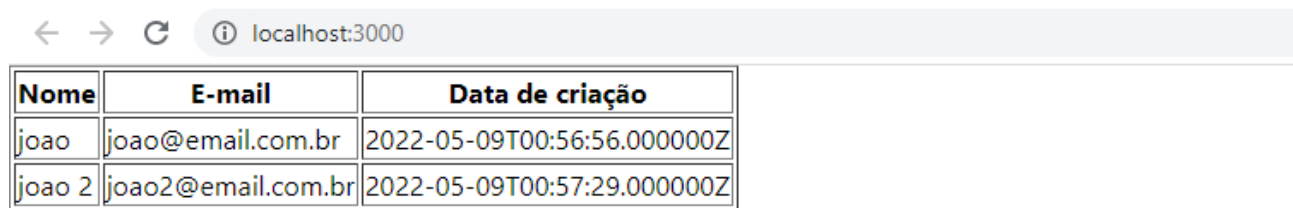
Note que no topo do arquivo temos 2 imports, fazemos isso pois toda lib que desejamos utilizar devemos previamente importar no topo de cara arquivo react.

No caso importamos a lib do **axios** e importamos duas funções da lib do react (no caso os 2 hooks que vamos utilizar)

Na linha 6 do arquivo, utilizamos nosso 1º hook, no caso retornamos em uma variavel e uma função, aproveitamos um comportamento de desestruturação do javascript, devido a isso teremos acesso a variavel **usuarios** e o metodo **setUsuarios** que será responsável por atribuir valores dinamicamente em nossa variavel.

Já na linha 8 utilizamos nosso outro hook, esse hook é responsável por executar nosso metodo de consulta de usuarios com a lib axios, ou seja, toda vez que acessarmos nossa página, o hook **useEffect**, chamara nossa consulta, que ao terminar de consultar, irá atribuir a lista de usuarios utilizando o metodo **setUsuarios**.

Feito isso, teremos nosso html integrado a uma api de consulta.



A screenshot of a web browser window with the address bar showing 'localhost:3000'. Below the address bar is a table with three columns: 'Nome', 'E-mail', and 'Data de criação'. The table contains two rows of data.

Nome	E-mail	Data de criação
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z

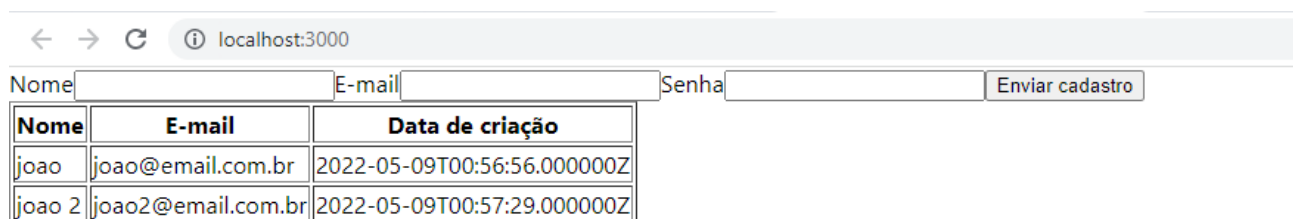
Criando um usuário

Para criar seguiremos passos parecidos, utilizaremos um html de formulario e chamaremos uma função javascript que utilizará nossa api de consulta.

Atualize seu codigo com esse html antes da sua tabela de dados

```
<form>
  <label>Nome</label>
  <input name="nome" />
  <label>E-mail</label>
  <input type="email" name="email" />
  <label>Senha</label>
  <input type="password" name="senha" />
  <button type="submit">Enviar cadastro</button>
</form>
```

Note que teremos um formulario de input de dados



A screenshot of a web browser window with the address bar showing 'localhost:3000'. Below the address bar is a registration form with three input fields labeled 'Nome', 'E-mail', and 'Senha', followed by a 'Enviar cadastro' button. Below the form is a table with three columns: 'Nome', 'E-mail', and 'Data de criação'. The table contains two rows of data.

Nome	E-mail	Data de criação
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z

Em um sistema html comum, colocaríamos um atributo **action** na tag **<form>** e essa url no action seria o lugar aonde pegaríamos os dados digitados no formulario e salvaríamos no banco de dados.

Esse processo normalmente envolve o redirecionamento para a tela que salva os dados e após o save dos dados, novamente seria redirecionados para a tela de consulta.

No React podemos fazer isso sem problemas, porém faremos diferente, aproveitaremos o comportamento do nosso hook **useState** pois ele é capaz de atualizar variaveis dinamicamente.

Para isso crie utilizando o **useState** 3 variaveis (nome, email e senha)

O início do seu código deve ficar assim

```
JS App.js M X
1 import axios from "axios";
2 import { useEffect, useState } from "react";
3
4 function App() {
5
6     const [usuarios, setUsuarios] = useState([]);
7
8     const [nome, setNome] = useState("");
9     const [email, setEmail] = useState("");
10    const [senha, setSenha] = useState("");
11
```

Note que no **useState** colocamos uma string vazia, fazemos isso pois precisamos sempre inicializar a variável com um valor default.

Para setarmos os valores ao digitar nos inputs, podemos utilizar os eventos padrões que algumas tags html tem ao utilizar o react, no caso utilizaremos o evento **onChange**.

Atualize seu código para

```
<form>
  <label>Nome</label>
  <input name="nome" onChange={ e => setNome(e.target.value) } />
  <label>E-mail</label>
  <input type="email" name="email" onChange={ e => setEmail(e.target.value) } />
  <label>Senha</label>
  <input type="password" name="senha" onChange={ e => setSenha(e.target.value) } />
  <button type="submit">Enviar cadastro</button>
</form>
```

Note que estamos utilizando arrowFunctions, estamos utilizando apenas para agilizar, pois poderíamos criar uma function e passar ela de callback, o importante é que ao digitar, sempre estamos chamando nossa função que faz o set do nome, e-mail e senha isso irá preencher os dados conforme digitamos.

Para exemplificar faça a seguinte experiencia

Atualize seu código temporariamente para

```

<form>
  <label>Nome</label>
  <input name="nome" onChange={ e => setNome(e.target.value) } />
  <label>E-mail</label>
  <input type="email" name="email" onChange={ e => setEmail(e.target.value) } />
  <label>Senha</label>
  <input type="password" name="senha" onChange={ e => setSenha(e.target.value) } />
  <button type="submit">Enviar cadastro</button>
</form>

<p>{ nome }</p>
<p>{ email }</p>
<p>{ senha }</p>

```

Salve, acesse o navegador e comece a digitar nos campos do formulário.

The screenshot shows a web browser at localhost:3000. The registration form has three input fields: 'Nome' with the value 'meu nome digitado', 'E-mail' with the value 'email@email.com', and 'Senha' with masked characters '.....'. A red cursor is visible in the 'Nome' field. Below the form, the values are displayed: 'meu nome digitado', 'email@email.com', and '123456'. At the bottom, there is a table with the following data:

Nome	E-mail	Data de criação
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z

Você verá que ao digitar o texto automaticamente irá para as tag <p>, isso acontece pois estamos atualizando nossas variaveis dinamicamente com o hook do react.

Registrando um usuário

Com nossas variaveis devidamente atualizadas, vamos enviar essas informações para a api de cadastro de usuarios.

Atualize seu código para


```

JS App.js M X
1  import axios from "axios";
2  import { useEffect, useState } from "react";
3
4  function App() {
5
6      const [usuarios, setUsuarios] = useState([]);
7
8      const [nome, setNome] = useState("");
9      const [email, setEmail] = useState("");
10     const [senha, setSenha] = useState("");
11
12     useEffect(() => {
13         axios.get('http://localhost:8000/users').then(response => {
14             setUsuarios(response.data.users);
15         }).catch(error => console.log(error));
16     }, []);
17
18     function salvarUsuarios()
19     {
20
21         let body = {
22             name: nome,
23             email: email,
24             password: senha
25         }
26
27         axios.post("http://localhost:8000/users", body).then(response => {
28             if(response.status === 201) {
29                 alert("Usuario criado com sucesso!");
30             }
31         }).catch(error => console.log(error));
32     }
33
34     return (
35         <div>
36             <form onSubmit={event => {
37                 event.preventDefault();
38                 salvarUsuarios();
39             }}>
40                 <label>Nome</label>
41                 <input name="nome" onChange={e => setNome(e.target.value)} />
42                 <label>E-mail</label>
43                 <input type="email" name="email" onChange={e => setEmail(e.target.value)} />
44                 <label>Senha</label>
45                 <input type="password" name="senha" onChange={e => setSenha(e.target.value)} />
46                 <button type="submit">Enviar cadastro</button>
47             </form>

```

Criamos a function **salvarUsuarios**, essa função é responsável por pegar nossas variaveis e enviar para a api de cadastro.

Note que a function não tem parametros, mas mesmo assim conseguimos pegar nossas variaveis (nome, email e senha) isso é possível pois criamos no começo da página nossas variaveis em scope global. Com isso, podemos acessa-las em todas as partes dessa pagina, tando para exibi-las (como fizemos na experiencia quando para envia-las para uma api.

Na parte destacada da imagem utilizamos um novo evento, no caso o evento **onSubmit**. Esse evento é proveniente da tag **<form>**, toda vez que esse formulario for submtido ele irá executar o código dentro do evento.

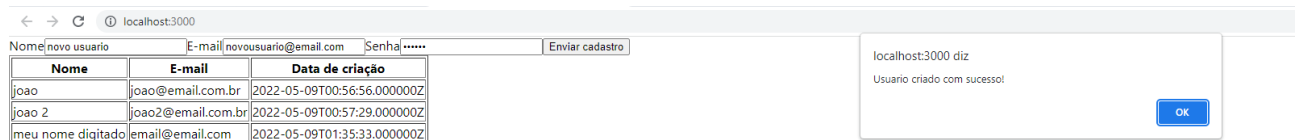
No caso colocamos duas instruções

1. `event.preventDefault()`

2. salvarUsuarios()

No caso a 1º serve para parar o processo que causaria um refresh na pagina, pois esse é um comportamento padrão de formularios que são submetidos, mas no caso é interessante para nosso projeto que isso não ocorra, então utilizamos o comando **preventDefault()**.

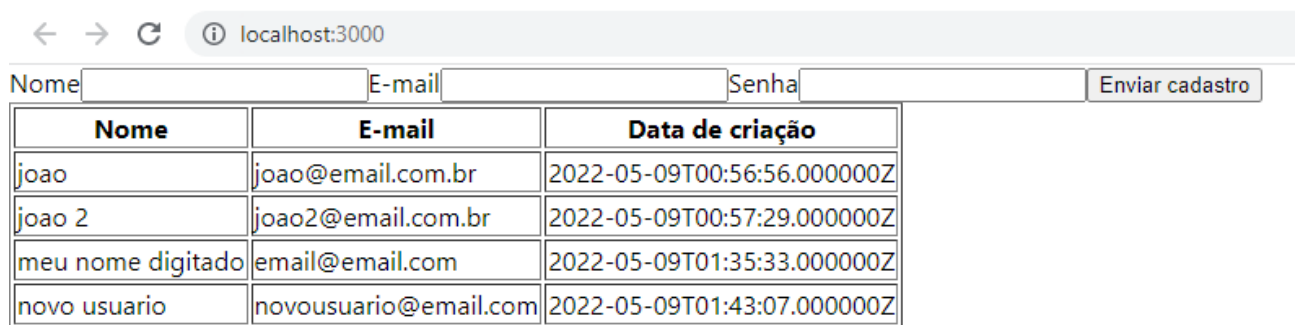
E a segunda instrução é a chamada da function que criamos, e ela por sua vez, envia os dados para a API que responde com o status code 201 informando que o dado foi criado e damos a mensagem de alerta.



The screenshot shows a web browser at localhost:3000. The registration form has three input fields: 'Nome' (containing 'novo usuario'), 'E-mail' (containing 'novousuario@email.com'), and 'Senha' (containing '.....'). There is an 'Enviar cadastro' button. Below the form is a table with three columns: 'Nome', 'E-mail', and 'Data de criação'. The table contains three rows of data. To the right of the form, a modal message box says 'localhost:3000 diz: Usuario criado com sucesso!' with an 'OK' button.

Nome	E-mail	Data de criação
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z
meu nome digitado	email@email.com	2022-05-09T01:35:33.000000Z

Feito isso, podemos dar um refresh na tela (**ctrl + F5**) e verificar o novo usuario criado.



The screenshot shows the same web browser after a refresh. The 'Nome' input field now contains 'novo usuario'. The table now has four rows, including the new user.

Nome	E-mail	Data de criação
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z
meu nome digitado	email@email.com	2022-05-09T01:35:33.000000Z
novo usuario	novousuario@email.com	2022-05-09T01:43:07.000000Z

Removendo um usuário

Para remover um usuário é simples, basicamente pegaremos um evento de click em um botão de remover, e chamaremos nossa API que previamente terá o metodo de remover usuarios.

Atualize seu código para

```

function removerUsuario(id)
{
}

return (
  <div>
    <form onSubmit={ event => {
      event.preventDefault();
      salvarUsuarios();
    } }>
      <label>Nome</label>
      <input name="nome" onChange={ e => setNome(e.target.value) } />
      <label>E-mail</label>
      <input type="email" name="email" onChange={ e => setEmail(e.target.value) } />
      <label>Senha</label>
      <input type="password" name="senha" onChange={ e => setSenha(e.target.value) } />
      <button type="submit">Enviar cadastro</button>
    </form>

    <table border="1">
      <thead>
        <tr>
          <th>Nome</th>
          <th>E-mail</th>
          <th>Data de criação</th>
          <th>Ações</th>
        </tr>
      </thead>
      <tbody>
        { usuarios.map( usuario => [
          return <tr>
            <td>{ usuario.name }</td>
            <td>{ usuario.email }</td>
            <td>{ usuario.created_at }</td>
            <td>
              <button onClick={ event => removerUsuario(usuario.id) }>remover</button>
            </td>
          </tr>
        ) ] ) }
      </tbody>
    </table>
  </div>
)

```

Fizemos duas coisas importantes nessa primeira modificação. Dentro de nosso loop de usuarios, adicionamos um botão **remover** e esse botão chama uma function **removerUsuario** e nela passamos o ID do nosso usuário (temos essa informação da API)

A function por sua vez espera como parametro o ID do usuario, note que dessa vez estamos utilizando parametros na function, estamos fazendo isso, pois facilita receber os Ids dos usuarios que devemos remover, pois pegaremos essa informação ao clicar no botão remover.

Agora vamos implementar a function de remover usuarios.

```

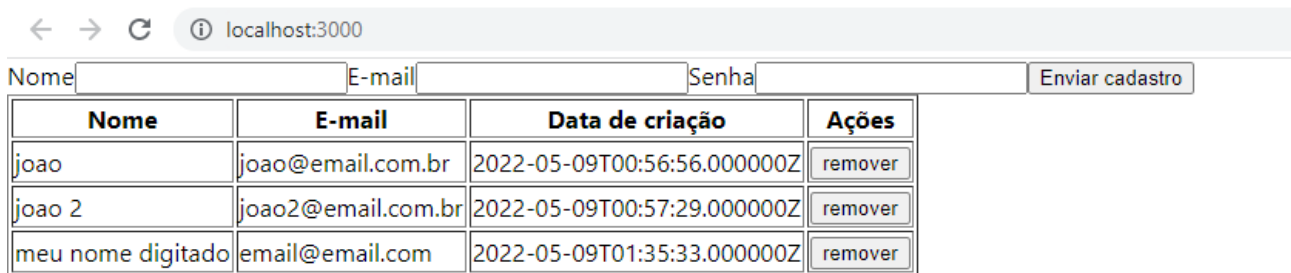
function removerUsuario(id)
{
  axios.delete("http://localhost:8000/users/" + id).then(response => {
    alert("Usuario removido com sucesso!");
  }).catch(error => console.log(error));
}

```

O metodo de remover usuarios é simples, basicamente ele recebe o id do usuario, envia para API e ao finalizar o processo, ele da um alerta informando que tudo ocorreu com sucesso.



Ao visualizar a mensagem, dê um refresh em sua tela e verifique que o usuario foi removido com sucesso.



Atualizando um usuario

Para atualizar um usuário vamos aproveitar o formulario de criação de usuarios, mas antes precisamos de uma nova variavel dinamica, pois para atualizar um usuario, alem do nome, e-mail e senha, precisaremos do ID, então primeiro vamos criar a nova variavel ID utilizando o hook `useState`.

```
function App() {  
  const [usuarios, setUsuarios] = useState([]);  
  const [id, setId] = useState("");  
  const [nome, setNome] = useState("");  
  const [email, setEmail] = useState("");  
  const [senha, setSenha] = useState("");  
}
```

Depois de criada, a ideia é que, ao clicarmos em um botão **editar** na listagem de usuarios, nos preencheremos nossas variaveis (id, nome, email e senha) com os dados do usuario da listagem.

Após isso poderemos atualizar os dados que gostaríamos e ao final, enviaremos esses dados para a API de atualização.

Vamos 1º criar um botão editar que atualize nossas variaveis

```

42 function dadosDeEdicao(usuario)
43 {
44   |
45 }
46
47 return (
48   <div>
49     <form onSubmit={ event => {
50       event.preventDefault();
51       salvarUsuarios();
52     } }>
53       <label>Nome</label>
54       <input name="nome" onChange={ e => setNome(e.target.value) } />
55       <label>E-mail</label>
56       <input type="email" name="email" onChange={ e => setEmail(e.target.value) } />
57       <label>Senha</label>
58       <input type="password" name="senha" onChange={ e => setSenha(e.target.value) } />
59       <button type="submit">Enviar cadastro</button>
60     </form>
61
62     <table border="1">
63       <thead>
64         <tr>
65           <th>Nome</th>
66           <th>E-mail</th>
67           <th>Data de criação</th>
68           <th>Ações</th>
69         </tr>
70       </thead>
71       <tbody>
72         { usuarios.map( usuario => {
73           return <tr>
74             <td>{ usuario.name }</td>
75             <td>{ usuario.email }</td>
76             <td>{ usuario.created_at }</td>
77             <td>
78               <button onClick={ event => dadosDeEdicao(usuario) }>editar</button>
79               <button onClick={ event => removerUsuario(usuario.id) }>remover</button>
80             </td>
81           </tr>
82         } ) }
83       </tbody>
84     </table>

```

Esse botão, como vocês podem ver acima, chama um metodo dadosDeEdicao e passa como parametro o usuario da lista.

Previamente criamos esse metodo dadosDeEdicao na parte superior do codigo, agora vamos implemenar a funcionalidade, que no caso será (setar todos os valores em nossas variaveis)

```

function dadosDeEdicao(usuario)
{
  setId(usuario.id);
  setNome(usuario.name);
  setEmail(usuario.email);
  setSenha(usuario.password);
}

```

Se nesse momento, você clicar no botão editar, não irá acontecer nada com nosso formulario, isso acontece por que não linkamos nossas variaveis em nossos campos <input> precisamos controlar nossos inputs. Anteriormente eles apenas conseguiam setar novos valores em nossas variaveis, agora eles receberao valores também.

Para isso é simples, basta passar nossas variaveis no atributo value da tag input


```

<form onSubmit={ event => {
  event.preventDefault();
  salvarUsuarios();
} }>
  <label>Nome</label>
  <input name="nome" value={ nome } onChange={ e => setNome(e.target.value) } />
  <label>E-mail</label>
  <input type="email" name="email" value={ email } onChange={ e => setEmail(e.target.value) } />
  <label>Senha</label>
  <input type="password" name="senha" value={ senha } onChange={ e => setSenha(e.target.value) } />
  <button type="submit">Enviar cadastro</button>
</form>

```

Note que as variaveis estão entre chaves, isso acontece pois nossas variaveis no fim, são javascript e todo javascript deve estar entre chaves, feito isso salve e clique novamente no botão editar.

The screenshot shows a web browser at localhost:3000. At the top, there is a form with three input fields: 'Nome' (containing 'joao 2'), 'E-mail' (containing 'joao2@email.com.br'), and 'Senha' (masked with dots). To the right of the inputs is a button labeled 'Enviar cadastro'. Below the form is a table with four columns: 'Nome', 'E-mail', 'Data de criação', and 'Ações'.

Nome	E-mail	Data de criação	Ações
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z	editar remover
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z	editar remover
meu nome digitado	email@email.com	2022-05-09T01:35:33.000000Z	editar remover

Você notará que ao clicar o formulario será preenchido com os valores do usuario da lista, agora precisamos enviar esses dados para a api de atualização.

Mas nosso formulario, ao clicar envia os dados para a api de criação e não para a api de edição

```

<form onSubmit={ event => {
  event.preventDefault();
  salvarUsuarios();
} }>
  <label>Nome</label>

```

Vamos aproveitar que temos nosso ID e vamos fazer uma condicional, onde se tivermos o ID preenchido (que acontece apenas ao clicar na listagem) enviaremos para a atualização, mas caso não tenha o ID, enviaremos para a criação de usuarios.

Para isso previamente vamos criar o metodo de atualização

```
function atualizarUsuarios()
{
    let body = {
        name: nome,
        email: email,
        password: senha
    }

    axios.put("http://localhost:8000/users/" + id, body).then(response => {
        alert("Usuario atualizado com sucesso!");
    }).catch(error => console.log(error));
}
```

Esse metodo é parecido com o metodo de criação, mas ele utiliza o ID para mandar para a rota correta.

Nossa condicional

```
return (
    <div>
        <form onSubmit={ event => {
            event.preventDefault();

            if(id) {
                atualizarUsuarios();
            } else {
                salvarUsuarios();
            }
        } }>
```

Se tiver o ID utiliza o metodo atualizar, caso contrario ele continua salvando o usuario.

The screenshot shows a web application running on localhost:3000. At the top, there is a form with fields for 'Nome' (João 3 editado), 'E-mail' (email@email.com), and 'Senha' (masked with dots), followed by an 'Enviar cadastro' button. Below the form is a table with the following data:

Nome	E-mail	Data de criação	Ações
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z	editar remover
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z	editar remover
João 3	email@email.com	2022-05-09T01:35:33.000000Z	editar remover

On the right side, a modal dialog is open with the title 'localhost:3000 diz' and the message 'Usuario atualizado com sucesso!'. It has an 'OK' button.

Clicando no botão editar, e na sequencia mudar as informações do formulario e enviar o cadastro, o mesmo será atualizado.

Basta atualizar a página e você verá o resultado atualizado.

← → ↻ ⓘ localhost:3000

Nome E-mail Senha

Nome	E-mail	Data de criação	Ações
joao	joao@email.com.br	2022-05-09T00:56:56.000000Z	<input type="button" value="editar"/> <input type="button" value="remover"/>
joao 2	joao2@email.com.br	2022-05-09T00:57:29.000000Z	<input type="button" value="editar"/> <input type="button" value="remover"/>
João 3 editado	email@email.com	2022-05-09T01:35:33.000000Z	<input type="button" value="editar"/> <input type="button" value="remover"/>

Basicamente temos um crud finalizado, porém nosso arquivo está um pouco grande e ele tem muitas responsabilidades, para melhorar isso vamos utilizar o Reactjs com o proposito de uso dele, que não é apenas utilizar dados dinamicos, mas sim componentizar sua aplicação, e poder reaproveitar esses componentes em outras partes do sistemas.

Para isso iremos separar nosso sistema em componentes de listagem (nossa tabela) e criação/edição (nosso formulario)

Componentizando nossa listagem

Crie um arquivo chamado lista.js

```
JS App.js  M    JS lista.js  U ●
1  function Lista()
2  {
3
4  }
5
6  export default Lista;
```

Como o arquivo App.js esse componente deve retornar um html, então vamos utilizar o html da nossa tabela de listagem de usuarios para esse componente.

```

1 function Lista()
2 {
3   return [
4     <table border="1">
5       <thead>
6         <tr>
7           <th>Nome</th>
8           <th>E-mail</th>
9           <th>Data de criação</th>
10          <th>Ações</th>
11        </tr>
12      </thead>
13      <tbody>
14        { usuarios.map( usuario => {
15          return <tr>
16            <td>{ usuario.name }</td>
17            <td>{ usuario.email }</td>
18            <td>{ usuario.created_at }</td>
19            <td>
20              <button onClick={ event => dadosDeEdicao(usuario) }>editar</button>
21              <button onClick={ event => removerUsuario(usuario.id) }>remover</button>
22            </td>
23          </tr>
24        } ) }
25      </tbody>
26    </table>
27  ]
28 }
29
30 export default Lista;

```

Porém temos uma questão aqui, agora nosso componente está em outro escopo, e com isso ele perde o acesso da variavel usuarios e tambem aos metodos dadosDeEdicao e removerUsuario.

Para o 1º problema de variaveis vamos utilizar uma caracteristica dos componentes que são as **props**, para utilizar eles precisamos no inicio da function passar o parametro props.

```

1 function Lista(props)
2 {
3   return [

```

Feito isso, podemos atualizar nosso loop de usuarios para isto

```

<tbody>
  { props.usuarios.map( usuario => {
    return <tr>
      <td>{ usuario.name }</td>
      <td>{ usuario.email }</td>
      <td>{ usuario.created_at }</td>
      <td>
        /* <button onClick={ event => dadosDeEdicao(usuario) }>editar</button>
        <button onClick={ event => removerUsuario(usuario.id) }>remover</button> */
      </td>
    </tr>
  } ) }
</tbody>

```

Pois agora pegaremos esse valor via props.

Note que deixei nossos botões comentados, fiz isso para temporariamente evitar erros de compilação, pois atualmente não temos acesso a esses metodos dos botoes.

No momento pode salvar seu novo componente.

No nosso arquivo App.js vamos utilizar nosso novo componente.

Primeiramente é necessário importar seu novo componente (como se fosse uma lib)

```
import axios from "axios";
import { useEffect, useState } from "react";
import Lista from "../lista";

function App() {

  const [usuarios, setUsuarios] = useState([]);
```

Feito isso é possível utilizar seu componente como se ele fosse uma tag html

```
<label>Senha</label>
<input type="password" name="senha" />
<button type="submit">Enviar cada</button>
</form>

<Lista usuarios={ usuarios } />
</div>
```

Note que passei como se fosse um atributo a variavel usuarios, isso é possível pois, dentro do componente estamos utilizando as props.

Salve seu arquivo e verifique, tudo esta funcionando corretamente

Agora vamos resolver o problema dos botoes.

1º botão de remover, esse é mais simples, basicamente copiaremos nosso metodo **removeUsuario** e importaremos nossa lib **axios**, pois agora precisamos utilizar ela em outro arquivo.

```

1 import axios from "axios";
2
3 function Lista(props)
4 {
5
6     function removerUsuario(id)
7     {
8         axios.delete("http://localhost:8000/users/" + id).then(response => {
9             alert("Usuario removido com sucesso!");
10         }).catch(error => console.log(error));
11     }
12
13     return (
14         <table border="1">
15             <thead>
16                 <tr>
17                     <th>Nome</th>
18                     <th>E-mail</th>
19                     <th>Data de criação</th>
20                     <th>Ações</th>
21                 </tr>
22             </thead>
23             <tbody>
24                 { props.usuarios.map( usuario => {
25                     return <tr>
26                         <td>{ usuario.name }</td>
27                         <td>{ usuario.email }</td>
28                         <td>{ usuario.created_at }</td>
29                         <td>
30                             /* <button onClick={ event => dadosDeEdicao(usuario) }>editar</button> */
31                             <button onClick={ event => removerUsuario(usuario.id) }>remover</button>
32                         </td>
33                     </tr>
34                 } ) }
35             </tbody>
36         </table>
37     )
38 }
39

```

Feito isso salve o arquivo e pode testar, nossa função de deletar esta funcionando corretamente.

2º editar o usuario, nesse ponto precisamos de um gatilho, para que ao clicar no botão editar, o metodo dadosDeEdicao seja chamado, não podemos simplesmente copiar o metodo para dentro desse nosso componente, pois esse metodo utiliza as variaveis nome, email e senha que são compartilhadas no formulario de cadastro.

Para resolver isso, o react disponibiliza para nos a possibilidade de criar eventos, que basicamente são props criadas dentro do próprio componente, ou seja, se criarmos uma prop dentro de um componente, essa prop automaticamente vira um evento do componente.

Então no nosso botão editar, ao clicar nele vamos criar o evento **onEditar** via props.

```

<td>{ usuario.created_at }</td>
<td>
    <button onClick={ event => props.onEditar([usuario]) }>editar</button>
    <button onClick={ event => removerUsuario(usuario.id) }>remover</button>
</td>
</tr>

```

É bem simples, basta passar a variavel props mais o nome do evento a ser criado, note que eu continuei passando a variavel usuario como parametro, faço isso pois esse evento por padrão terá os dados do usuario que clicamos, esse arquivo já pode ser salvo e agora vamos recuperar esse valor no nosso arquivo App.js

Em nosso arquivo App.js atualize nossa chamada do componente <Lista /> para

```

    <button type="submit">Enviar cadastro</button>
  </form>

  <Lista usuarios={ usuarios } onEditar={ usuario => dadosDeEdicao(usuario) } />
</div>

```

Pegamos nosso evento como qualquer outro, basta passar o nome que designamos dentro do componente e a partir daí, podemos fazer o que quisermos, que no caso é chamar nosso método **dadosDeEdicao**, salve e teste o processo de edição novamente, tudo deve ocorrer normalmente.

A ideia é que sempre que possível crie componentes separados, cada um com sua responsabilidade, isso facilita na manutenção e na separação de responsabilidades do sistema.

Extra

Em todo o processo, a cada ação tivemos sempre que atualizar nossa página, mas o Reactjs é utilizado para dar dinamismo nas páginas, então vamos resolver isso com o Reactjs + javascript.

Ao cadastrar um usuario vamos atualizar nossa listagem

Crie o método atualizar listagem

```

function atualizarListagem()
{
  axios.get('http://localhost:8000/users').then(response => {
    setUsuarios(response.data.users);
  }).catch(error => console.log(error));
}

```

Esse método basicamente busca os usuarios na api e atualiza a variável **usuarios**.

Feito isso vamos atualizar nosso App.js em pontos estratégicos.

Ao criar um novo usuario depois do alert

```

function salvarUsuarios()
{
    let body = {
        name: nome,
        email: email,
        password: senha
    }

    axios.post("http://localhost:8000/users", body).then(response => {
        if(response.status === 201) {
            alert("Usuario criado com sucesso!");
            atualizarListagem();
        }
    }).catch(error => console.log(error));
}

```

Ao atualizar um usuario depois do alert tambem

```

function atualizarUsuarios()
{
    let body = {
        name: nome,
        email: email,
        password: senha
    }

    axios.put("http://localhost:8000/users/" + id, body).then(response => {
        alert("Usuario atualizado com sucesso!");
        atualizarListagem();
    }).catch(error => console.log(error));
}

```

E por ultimo, como não temos acesso ao metodo **setUsuarios** dentro do componente <Lista />, vamos novamente criar um novo evento, que ao ser chamado irá atualizar nossa listagem.

Dentro do componente <Lista /> depois do alert ao remover o usuario crie o evento **onDelete**


```

1  import axios from "axios";
2
3  function Lista(props)
4  {
5
6      function removerUsuario(id)
7      {
8          axios.delete("http://localhost:8000/users/" + id).then(response => {
9              alert("Usuario removido com sucesso!");
10             props.onDelete();
11         }).catch(error => console.log(error));
12     }
13

```

Feito isso, retorne ao arquivo App.js e capture esse evento e chame o metodo de atualizarListagem

```

<label>Senha</label>
<input type="password" name="senha" value={ senha } onChange={ e => setSenha(e.target.value) } />
<button type="submit">Enviar cadastro</button>
</form>

<Lista usuarios={ usuarios } onEditar={ usuario => dadosDeEdicao(usuario) } onDelete={ atualizarListagem } />
</div>
);

```

Note que dessa vez eu não utilizei uma arrowFunction, fiz isso pois como não precisamos seguir parametros nesse evento, eu chamo o metodo direto, pois ele não precisa de nenhum parametro para existir, agora salve e teste seu projeto, não precisamos mais dar refresh na tela a cada ação.

Observação: note que ao criar um usuario ou atualizar o mesmo, o formulario ainda fica com dados, isso pode gerar um bug, pois os dados ficam misturados (dados de criação e edição), para evitar problemas podemos simplesmente a cada ação, limpar nossas variaveis, implemente o seguinte metodo.

```

function limparDados()
{
    setId("");
    setNome("");
    setEmail("");
    setSenha("");
}

```

E no metodo atualizarListagem, chame esse metodo logo abaixo do metodo que seta a variavel de usuarios.

```
function atualizarLista() {
  axios.get('http://localhost:8000/users').then(response =>
    setUsuarios(response.data.users);
    limparDados();
  ).catch(error => console.log(error));
}
```

Follow link (ctrl + click)

Feito isso salve e teste novamente, todo o processo agora esta correto, a cada ação o formulario será limpo.

Extra 2

Em todo o projeto, trabalhamos com o visual padrão dos navegadores, para aplicarmos estilos css é simples, podemos importar bibliotecas de temas completas ou apenas importar o css como um site comum.

Dentro do arquivo App.js, vamos importar o bootstrap, mas antes vamos instalar via npm a lib do bootstrap com o comando na raiz do projeto

npm install --save bootstrap

Feito isso basta importar no topo do arquivo o css

```
1 import axios from "axios";
2 import { useEffect, useState } from "react";
3 import Lista from "../lista";
4 import 'bootstrap/dist/css/bootstrap.css';
5
6 function App() {
```

A partir dai basta utilizar as classes do bootstrap

```

    }
    return (
      <div className="container">
        <form onSubmit={ event => {
          event.preventDefault();

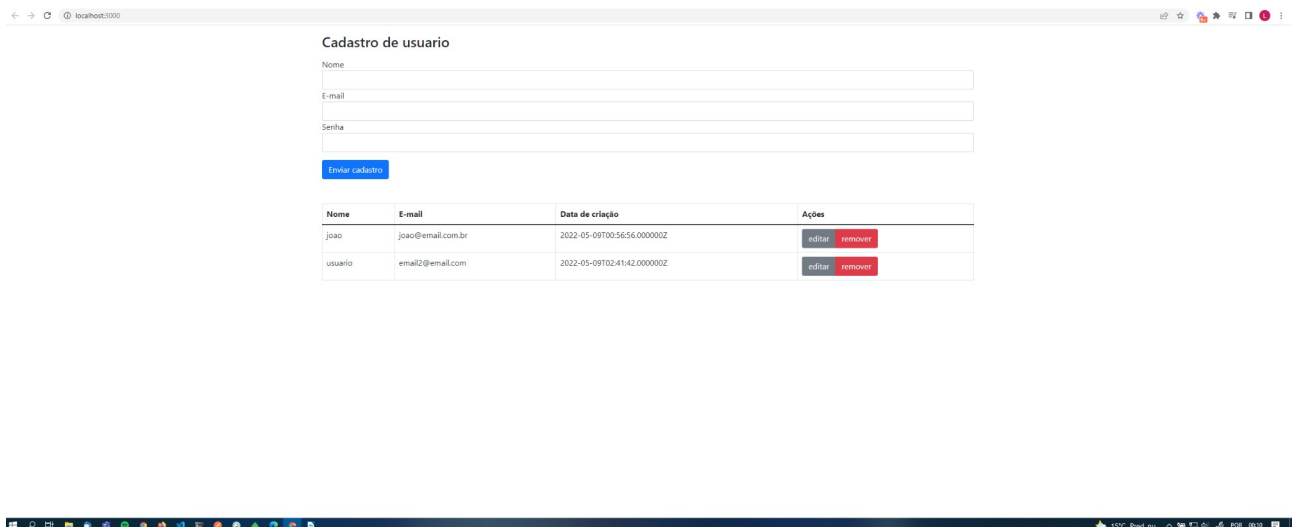
          if(id) {
            atualizarUsuarios();
          } else {
            salvarUsuarios();
          }
        }}>
          <div className="form-group">
            <label>Nome</label>
            <input className="form-control" name="nome" value={ nome } onChange={ e => setNome(e.target.value) } />
          </div>
          <div className="form-group">
            <label>E-mail</label>
            <input className="form-control" type="email" name="email" value={ email } onChange={ e => setEmail(e.target.value) } />
          </div>
          <div className="form-group">
            <label>Senha</label>
            <input className="form-control" type="password" name="senha" value={ senha } onChange={ e => setSenha(e.target.value) } />
          </div>
          <button className="btn btn-primary" type="submit">Enviar cadastro</button>
        </form>

        <Lista usuarios={ usuarios } onEditar={ usuario => dadosDeEdicao(usuario) } onDelete={ atualizarListagem } />
      </div>
    );
  };
}

```

Note que estamos utilizando o atributo **className** em vez de apenas **class**, fazemos dessa maneira para pois o Reactjs utiliza essa palavra reservada **class** para outros processos.

Feito as modificações teremos esse resultado visual facilmente.



Extra 3

Falando ainda sobre componentes, criamos um componente `<Lista />` e esse componente era complexo, mas tinha o proposito de ser utilizado apenas na nossa pagina de usuarios.

No Reactjs, podemos criar componentes e reutilizar ele em qualquer lugar do sistema, então vamos criar um componente chamado Card e esse componente deverá ser um container semelhante visualmente a um cartão, esse cartão irá conter nosso formulario de cadastro, ou seja, ele só sera responsavel por conter esse estilo de cartão.

Para isso crie um arquivo chamado Cartao.js

```
JS App.js M JS Cartao.js U X
1 function Cartao(props)
2 {
3
4 }
5
6 export default Cartao;
```

Essa function deverá retornar o conteúdo html do nosso cartão + as classes bootstrap que importamos antes.

```
JS App.js M JS Cartao.js U X
1 function Cartao(props)
2 {
3   return (
4     <div class="card my-3">
5       <div class="card-body">
6         <h5 class="card-title">{ props.titulo }</h5>
7         { props.children }
8       </div>
9     </div>
10  )
11 }
12
13 export default Cartao;
```

Note que estou esperando duas props (titulo e children), a prop titulo é auto explicativa, ele espera o titulo do cartão.

Já a props.children é uma prop padrão de componentes do Reactjs, anteriormente chamamos nosso componente <Lista /> como uma short tag, onde abrimos e fechamos ela na mesma tag.

Para termos acesso a prop children, devemos abrir a tag do nosso componente e fecha-lo normalmente e todo o conteúdo que ficar no meio será nossa props.children.

No nosso arquivo App.js veremos isso sendo utilizado.

```

64     axios.put("http://localhost:8000/users/" + id, body).then(response => {
65         alert("Usuário atualizado com sucesso!");
66         atualizarListagem();
67     }).catch(error => console.log(error));
68     }
69
70     function limparDados()
71     {
72         setId("");
73         setNome("");
74         setEmail("");
75         setSenha("");
76     }
77
78     return (
79         <div className="container">
80             <Cartao titulo="Cadastro de usuário">
81                 <form onSubmit={ event => {
82                     event.preventDefault();
83
84                     if(id) {
85                         atualizarUsuarios();
86                     } else {
87                         salvarUsuarios();
88                     }
89                 }>
90
91                 <div className="form-group">
92                     <label>Nome</label>
93                     <input className="form-control" name="nome" value={ nome } onChange={ e => setNome(e.target.value) } />
94                 </div>
95                 <div className="form-group">
96                     <label>E-mail</label>
97                     <input className="form-control" type="email" name="email" value={ email } onChange={ e => setEmail(e.target.value) } />
98                 </div>
99                 <div className="form-group">
100                     <label>Senha</label>
101                     <input className="form-control" type="password" name="senha" value={ senha } onChange={ e => setSenha(e.target.value) } />
102                 </div>
103                 <button className="btn btn-primary mt-3" type="submit">Enviar cadastro</button>
104             </form>
105         </Cartao>
106         <Cartao titulo="Lista de usuários">
107             <lista usuarios={ usuarios } onEditar={ usuario => dadosDeEdicao(usuario) } onDelete={ atualizarListagem } />
108         </Cartao>
109     </div>
110 );
111 }
112 }
113 }

```

Na mesma pagina eu estou utilizando o componente cartao tanto para envolver nosso formulario, quando para envolver nossa listagem de usuarios, com isso teremos esse resultado.

Cadastro de usuario

Nome

E-mail

Senha

Enviar cadastro

Lista de usuarios

Nome	E-mail	Data de criação	Ações
joao editado	joao@email.com.br	2022-05-09T00:56:56.000000Z	<div> editar remover </div>
novo usuario	admin@admin.com	2022-05-09T03:34:27.000000Z	<div> editar remover </div>

Bem mais organizado visualmente, e esse componente poderá ser utilizado em todo o sistema, basta importalo e utilizar.