

Desenvolvimento de Software para WEB

Projeto CRUD

Projeto CRUD

Introdução

- A ideia por trás deste projeto é criar uma aplicação simples onde será possível:
 - Criar uma entidade (CREATE)
 - Editar uma entidade (EDIT)
 - Listar as entidades criadas (LIST)
 - Apagar uma entidades criada (DELETE)
 - Mostrar uma entidade criada (RETRIEVE)

Introdução

- As tecnologias usadas nesse projeto serão:
 - React
 - Node
 - Express
 - Mongo (Futuramente o Firebase)
- MERN Stack: **M**ongo, **E**xpresse, **R**ead e **N**ode.

Parte 1

A Interface e Navegação

Criação do Projeto

- Execute o comando:
 - **create-react-app crud**
- Entre na pasta do projeto e instale as seguintes bibliotecas:
 - **cd crud**
 - **npm install bootstrap --save**
 - **npm install react-router-dom --save**

Criação do Projeto

- Crie o sistemas de pastas, dentro de src:
 - components
- Dentro de components, crie os arquivos:
 - Create.jsx
 - Edit.jsx
 - List.jsx
 - Home.jsx

Primeiros Componentes

- Create.jsx

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

export default class Create extends Component {
  render() {
    return (
      <div>
        <p>Create Component!!</p>
      </div>
    )
  }
}
```


Primeiros Componentes

- Edit.jsx

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

export default class Edit extends Component {
  render() {
    return (
      <div>
        <p>Edit Component!!</p>
      </div>
    )
  }
}
```

Primeiros Componentes

- List.jsx

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

export default class List extends Component {
  render() {
    return (
      <div>
        <p>List Component!!</p>
      </div>
    )
  }
}
```

Primeiros Componentes

- Home.jsx

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

export default class Home extends Component {
  render() {
    return (
      <div>
        <p>Home Component!!</p>
      </div>
    )
  }
}
```

Habilitando as Rotas

- index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';
```

[//https://learnwithparam.com/blog/basic-routing-in-react-router/](https://learnwithparam.com/blog/basic-routing-in-react-router/)

```
ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>,
  document.getElementById('root')
);
```

BrowserRouter é um dos roteadores mais usados.

No caso, aqui envolve toda a aplicação.

O Componente Principal

- App.js será nosso componente principal.
- A partir dele criaremos rotas para os outros componentes da aplicação, assim como a barra de navegação entre as páginas.

O Componente Principal

- Modifique o App.js (versão 0)

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import { BrowserRouter as Router } from 'react-router-dom';

export default class App extends Component {
  render() {
    return (
      <Router>
        <div className="container">

          <h2>Projeto CRUD</h2> <br />

        </div>
      </Router>
    );
  }
}
```

O Componente Principal

- Barra de navegação (versão 1)

```
import React, { Component } from 'react';
import 'bootstrap/dist/css/bootstrap.min.css';
import { BrowserRouter as Router, Link } from 'react-router-dom';

export default class App extends Component {
  render() {
    return (
      <Router>
        <div className="container">
          <nav className="navbar navbar-expand-lg navbar-light bg-light">
            <Link to="/" className="navbar-brand">CRUD</Link>

          </nav>
          <h2>Projeto CRUD</h2> <br />

        </div>
      </Router>
    );
  }
}
```

Links só podem ser criados dentro de um roteador.

O Componente Principal

- Links de navegação (versão 2)

```
...
<nav className="navbar navbar-expand-lg navbar-light bg-light">
  <Link to={"/"} className="navbar-brand">CRUD</Link>
  <div className="collapse navbar-collapse" id="navbarSupportedContent">
    <ul className="navbar-nav mr-auto">
      <li className="nav-item">
        <Link to={"/"} className="nav-link">Home</Link>
      </li>
      <li className="nav-item">
        <Link to={"/create"} className="nav-link">Create</Link>
      </li>
      <li className="nav-item">
        <Link to={"/list"} className="nav-link">List</Link>
      </li>
    </ul>
  </div>
</nav>
...
```


O Componente Principal

- Lógica das Rotas (versão 3)

```
import { BrowserRouter as Router, Switch, Route, Link } from 'react-router-dom';
```


```
import Create from './components/Create';  
import Edit from './components/Edit';  
import List from './components/List';  
import Home from './components/Home';
```

```
...
```

O Componente Principal

- Lógica das Rotas (versão 3)

```
...  
<h2>Projeto CRUD</h2> <br />  
<Switch>  
  <Route exact path="/" component={Home} />  
  <Route path="/create" component={Create} />  
  <Route path="/edit/:id" component={Edit} />  
  <Route path="/list" component={List} />  
</Switch>
```



O Switch, é um componente que recebe vários componentes Route e dado o caminho que for passado na URL um deles é renderizado.

Resultado

CRUD Home Create List

Projeto CRUD

Home Component!!

Parte 2

Formulário Criar Estudante

```

export default class Create extends Component {
  render() {
    return (
      <div style={{marginTop: 10}}>
        <h3>Criar Estudante</h3>
        <form>
          <div className="form-group">
            <label>Nome: </label>
            <input type="text" className="form-control"/>
          </div>
          <div className="form-group">
            <label>Curso: </label>
            <input type="text" className="form-control"/>
          </div>
          <div className="form-group">
            <label>IRA: </label>
            <input type="text" className="form-control"/>
          </div>
          <div className="form-group">
            <input type="submit" value="Criar" className="btn btn-primary"/>
          </div>
        </form>
      </div>
    )
  }
}

```

Create.jsx

Submetendo o Form

- O próximo passo é submeter o Formulário para o Back-End (ou seja, o componente `Create.jsx`).
- Para cada campo, teremos uma função que captura seu valor e modifica o estado do componente `Create.jsx`.
- Vamos ver como ficaria o código para submeter apenas o nome, para os outros campos, a lógica é a mesma.

Submetendo o Form

- Crie o construtor

```
constructor(props){  
  super(props)  
  this.state = {nome:""}  
  
  this.setNome = this.setNome.bind(this)  
  this.onSubmit = this.onSubmit.bind(this)  
}
```

Submetendo o Form

- Os método de ajuste do 'nome' e submissão do formulário:

```
setNome(e){  
  this.setState({nome:e.target.value})  
}
```

```
onSubmit(e){  
  //https://www.robinwieruch.de/react-preventdefault  
  e.preventDefault() //impede que o browser faça o reload, perdendo assim a informação  
  console.log('Nome: ' + this.state.nome)  
  this.setState({nome:''})  
}
```


Submetendo o Form

- Modificando o Formulário

*Métodos sendo chamados
no formulário.*

...

```
<form onSubmit={this.onSubmit}>  
  <div className="form-group">  
    <label>Nome: </label>  
    <input type="text" className="form-control"  
      value={this.state.nome} onChange={this.setNome}/>  
  </div>
```

Submetendo o Form

- Exercício
 - Agora implemente para o resto dos campos, a mesma lógica do campo nome. Mostre os valores no método “onSubmit”

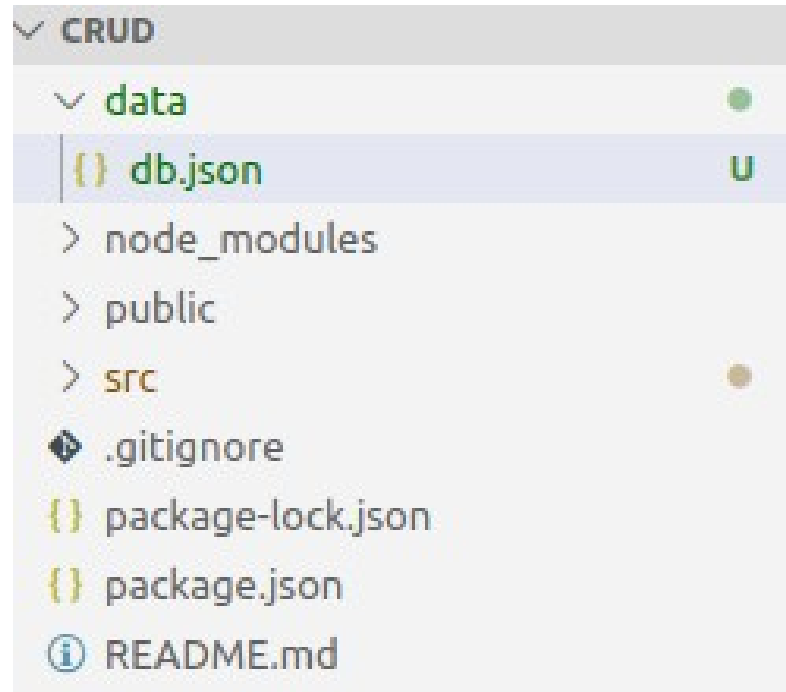
Usando uma Base de Dados

- Devemos registrar nossos dados em alguma base.
- Inicialmente vamos usar o json-server, um programa Node.js que implementar uma API Rest falsa.
- Depois, passaremos para o Express+MongoDB.

Json-Server

- <https://github.com/typicode/json-server>
- Instalando:
 - `sudo npm install -g json-server`
- Crie uma pasta chamada “data”, no mesmo nível de src, onde serão salvos os arquivos relativos as entidades do projeto CRUD.

Json-Server



Json-Server

- O arquivo db.json

```
{  
  "estudantes": [  
    {"id": 1, "nome": "Jefferson", "curso": "SI", "IRA": 9.0},  
    {"id": 2, "nome": "Thomas", "curso": "SI", "IRA": 7.5}  
  ]  
}
```

Json-Server

- Iniciando nossa API:
 - **json-server --watch data/db.json --port 3001**
- Agora acesse <http://localhost:3001/estudantes>
- Dica:
 - Instale um cliente REST no seu navegador (extensão)
 - Sugestões: Postman ou ARC

Json-Server

The screenshot displays the ARC REST client interface. On the left sidebar, the 'History' section is expanded, showing a list of requests. The selected request is a GET request to 'http://localhost:3001/estudantes'. The main panel shows the details of this request. The 'Method' is 'GET' and the 'Request URL' is 'http://localhost:3001/estudantes'. The 'Parameters' section is collapsed. The 'Headers' section is also collapsed, showing a message 'Headers are valid' and 'Headers size: 0 bytes'. The 'Variables' section is collapsed. The 'Status' bar shows '200 OK' and '21.00 ms'. The response body is displayed in a code editor, showing a JSON array with two objects. The first object has the following properties: 'id': 1, 'nome': 'Jefferson', 'curso': 'SI', and 'IRA': 9.

ARC Request

HTTP request

Socket

History

Today

GET http://localhost:3001/estudantes

Method GET Request URL http://localhost:3001/estudantes

Parameters

Headers Variables

Toggle source mode Insert headers set

Header name Header value

ADD HEADER

Headers are valid Headers size: 0 bytes

200 OK 21.00 ms DETAILS

```
[Array[2]
  -0: {
    "id": 1,
    "nome": "Jefferson",
    "curso": "SI",
    "IRA": 9
  },
  -1: {
```


React → Json-Server

- O problema agora é fazer com que nossa aplicação se comunique com o Json-Server (e futuramente com o Mongo, via Express).
- Para ajudar nessa tarefa, iremos instalar o Axios.
 - **npm install axios --save**
- O Axios facilita a chamada dos métodos HTTP (PUT, DELETE, POST, GET...)
- No caso da criação de um novo estudante, usaremos o método **HTTP POST**.

React → Json-Server

- Create.jsx

```
...
import axios from 'axios'

export default class Create extends Component {

  constructor(props){
    super(props)
    this.state = {nome:"",curso:"",IRA:""}

    this.setNome = this.setNome.bind(this)
    this.setCurso = this.setCurso.bind(this)
    this.setIRA = this.setIRA.bind(this)
    this.onSubmit = this.onSubmit.bind(this)
  }

  setNome(e){
    this.setState({nome:e.target.value})
  }

  ...
```

React → Json-Server

- Create.jsx

```
onSubmit(e){  
  //https://www.robinwieruch.de/react-preventdefault  
  e.preventDefault() //impede que o browser faça o reload, perdendo assim a informação  
  
  const novoEstudante = {nome:this.state.nome,  
                           curso:this.state.curso,  
                           IRA:this.state.IRA}  
  
  axios.post('http://localhost:3001/estudantes',novoEstudante)  
    .then(  
      (res)=>{  
        console.log(res.data.id)  
      }  
    )  
    .catch(  
      (error)=>{  
        console.log(error)  
      }  
    )  
  
  this.setState({nome:"",curso:"",IRA:""})  
}
```

Parte 3

Listar Estudantes Cadastrados

Introdução

- O objetivo agora, é listar os estudantes criados pela Create.jsx no componente List.jsx
- Nós já temos o Axios instalado, então não será problema.
- Para listar os estudantes cadastrados, devemos usar o método **HTTP GET**.

GET Estudantes

- Para “pegar” a lista de estudantes cadastrados, devemos criar no construtor um objeto que receba essa lista do json-server.
- Esse objeto será inicializado no método **componentDidMount**, chamado toda vez que o componente é montado. Geralmente, comunicações com bases de dados são feitas nesse método.

GET Estudiantes

```
...
constructor(props) {
  super(props)
  this.state = { estudiantes: [] }
}

componentDidMount() {
  axios.get('http://localhost:3001/estudiantes')
    .then(
      (res) => {
        this.setState({ estudiantes: res.data })
      }
    )
    .catch(
      (error) => {
        console.log(error)
      }
    )
}
...
```

- List.jsx

GET Estudantes

- Devemos montar esses “estudantes” de uma forma que fique fácil sua renderização JSX.
- Para isso, vamos criar um componente auxiliar que recebe um estudante “cru” (apenas os dados), e retorna uma linha de uma tabela em JSX.
- Crie o arquivo **TableRow.jsx**

GET Estudantes

- TableRow.jsx

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

export default class TableRow extends Component{
  render(){
    return(
      <tr>
        <td>
          {this.props.estudante.id}
        </td>
        <td>
          {this.props.estudante.nome}
        </td>
        <td>
          {this.props.estudante.curso}
        </td>
        <td>
          {this.props.estudante.IRA}
        </td>
      </tr>
    )
  }
}
```

GET Estudantes

- Feito o TableRow, o próximo passo é montar uma lista de objetos/componentes TableRow e renderizar essa lista dentro do render de List.jsx.

GET Estudantes

- List.jsx

```
montarTabela() {  
  if(!this.state.estudantes) return  
  return this.state.estudantes.map(  
    (estudante, i) => {  
      return <TableRow estudante={estudante} key={i} />;  
    }  
  )  
}
```

GET Estudantes

```
render() {  
  return (  
    <div>  
      <p>List Estudante</p>  
      <table className="table table-striped" style={{ marginTop: 20 }}>  
        <thead>  
          <tr>  
            <th>ID</th>  
            <th>Nome</th>  
            <th>Curso</th>  
            <th>IRA</th>  
          </tr>  
        </thead>  
        <tbody>  
          {this.montarTabela()}  
        </tbody>  
      </table>  
    </div>  
  )  
}
```

- List.jsx

Botões Editar e Apagar

- Voltando a TableRow.jsx

...

```
<td>
  {this.props.estudante.IRA}
</td>
<td style={{textAlign:"center"}}>
  <button className="btn btn-primary">Editar</button>
</td>
<td style={{textAlign:"center"}}>
  <button className="btn btn-danger">Apagar</button>
</td>
</tr>
```

...

Botões Editar e Apagar

- E em List.jsx

```
<table className="table table-striped" style={{ marginTop: 20 }}>
  <thead>
    <tr>
      <th>ID</th>
      <th>Nome</th>
      <th>Curso</th>
      <th>IRA</th>
      <th colspan="2" style={{ textAlign: "center" }}>Ação</th>
    </tr>
  </thead>
  <tbody>
    {this.montarTabela()}
  </tbody>
</table>
```

Parte 4

Editar e Apagar

Introdução

- Editar
 - O usuário irá clicar no botão editar, na listagem de estudantes.
 - O botão terá um `<Link>` para o componente `Editar.jsx` (a gente já fez esse link no `App.jsx`).
 - Pelo `<Link>`, também será enviado o id do estudante a ser editado.
 - Em `Editar.jsx`, de posse do id, iremos fazer um **GET** no json-server de um Estudante específico e só daí preencher os campos de `Editar.jsx` com os dados.
 - Uma vez os campos de `Editar.jsx` preenchidos, mudamos os campos que queremos e submetemos o formulário, usando o `Axios`.
 - Nesse caso, usaremos o método **HTTP PUT**.


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

```
export default class Edit extends Component {
```

```
  constructor(props){
    super(props)
    this.state = {nome:"",curso:"",IRA:""}

    this.setNome = this.setNome.bind(this)
    this.setCurso = this.setCurso.bind(this)
    this.setIRA = this.setIRA.bind(this)
    this.onSubmit = this.onSubmit.bind(this)
  }
```

```
  setNome(e){
    this.setState({nome:e.target.value})
  }
```

```
  setCurso(e){
    this.setState({curso:e.target.value})
  }
```

```
  setIRA(e){
    this.setState({IRA:e.target.value})
  }
```

```
...
```

Edit.jsx (mesmo código do Create)

```
...
```

```
  onSubmit(e){
    e.preventDefault()
  }
```

```
  render() {
    return (
      <div style={{marginTop: 10}}>
        <h3>Editar Estudante</h3>
        <form onSubmit={this.onSubmit}>
          <div className="form-group">
            <label>Nome: </label>
            <input type="text" className="form-control"
              value={this.state.nome} onChange={this.setNome}/>
          </div>
          <div className="form-group">
            <label>Curso: </label>
            <input type="text" className="form-control"
              value={this.state.curso} onChange={this.setCurso}/>
          </div>
        </form>
      </div>
    )
  }
```

```
...
```

Edit.jsx (mesmo código do Create)

```
...  
<div className="form-group">  
  <label>IRA: </label>  
  <input type="text" className="form-control"  
    value={this.state.IRA} onChange={this.setIRA}/>  
</div>  
<div className="form-group">  
  <input type="submit" value="Editar" className="btn btn-primary"/>  
</div>  
</form>  
</div>  
)  
}  
}
```

ComponentDidMount

- *Inicialmente, pegamos o id passado como parâmetro pro Edit.jsx e fazemos uma busca, por id, no json-server. Essa busca é feita dentro de componentDidMount.*

```
componentDidMount(){  
  //console.log("id: " + this.props.match.params.id)  
  axios.get('http://localhost:3001/estudantes/'+this.props.match.params.id)  
  .then(  
    (res)=>{  
      this.setState(  
        {  
          nome:res.data.nome,  
          curso:res.data.curso,  
          IRA:res.data.IRA  
        }  
      )  
    }  
  )  
  .catch(  
    (error)=>{  
      console.log(error)  
    }  
  )  
}
```

onSubmit

```
onSubmit(e){  
  e.preventDefault()  
  const estudanteAtualizado =  
    {nome:this.state.nome,  
     curso:this.state.curso,  
     IRA:this.state.IRA}  
  
  axios.put('http://localhost:3001/estudantes/'+this.props.match.params.id,estudanteAtualizado)  
    .then(  
      res=>{  
        //console.log(res.data)  
        this.props.history.push('/list');  
      }  
    )  
    .catch(error=>console.log(error))  
}
```

Apagar um Registro

- em TableRow.jsx

```
...
constructor(props){
  super(props)
  this.apagar = this.apagar.bind(this)
}

apagar(){
  axios.delete('http://localhost:3001/estudantes/'+this.props.estudante.id)
    .then(
      (res)=>{
        console.log('Registro apagado')
      })
    .catch((error)=>console.log(error))
}
...
```

Apagar um Registro

- em TableRow.jsx

```
...  
<td style={{ textAlign: "center" }}>  
  <button onClick={this.apagar} className="btn btn-danger">Apagar</button>  
</td>  
...
```

Atualizar a Página

- Existem duas abordagens:
 - Recarregar a página, fazendo uma nova conexão com o json-server, atualizando assim a lista de estudantes em List.jsx (NÃO RECOMENDADO)
 - Remover localmente no this.state de List.jsx, evitando uma chamada remota desnecessária. (MUITO RÁPIDO MAS EXIGE MAIS CÓDIGO)

Atualizar a Página

- Em List.jsx, iremos criar um método que irá apagar um elemento do array “estudantes” de this.state. Vejamos:

```
apagarElementoPorId(id){  
  let estudantesTemp = this.state.estudantes  
  for(let i=0;i<estudantesTemp.length;i++){  
    if(estudantesTemp[i].id===id){  
      estudantesTemp.splice(i,1)  
    }  
  }  
  this.setState({estudantes:estudantesTemp})  
}
```

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

Atualizar a Página

- A solução é repassar esse método para o componente TableRow.jsx (se lembram da comunicação filho → pai?). Mas antes, como o contexto irá mudar (vou passar pra TableRow.jsx), devo fazer um bind local. Vejamos ainda em List.jsx:

```
constructor(props) {  
  super(props)  
  this.state = { estudantes: [] }  
  
  this.apagarElementoPorId = this.apagarElementoPorId.bind(this)  
}
```

Atualizar a Página

- ...E repassar o método `apagarElementoPorId` para `TableRow.jsx`, via props:

```
montarTabela() {  
  if(!this.state.estudantes) return  
  return this.state.estudantes.map(  
    (estudante, i) => {  
      return <TableRow estudante={estudante} key={i} apagarElementoPorId={this.apagarElementoPorId}/>  
    }  
  )  
}
```

Atualizar a Página

- Em TableRow.jsx, modificamos o método apagar para que chame **apagarElementoPorId** CASO a chamada DELETE do axios tenha dado certo :

```
apagar(){
  axios.delete('http://localhost:3001/estudantes/'+this.props.estudante.id)
    .then(
      (res)=>{
        console.log('Registro apagado')
        this.props.apagarElementoPorId(this.props.estudante.id)
      })
    .catch((error)=>console.log(error))
}
```