



INTRODUÇÃO

1-Configurando o Visual Studio Code

Clica no ícone de extensões  e instala as seguintes extensões:

- Extension Pack for Java
- Spring Boot Extension Pack
- MySQL
- ThunderClient
- Lombok Annotations Support for VS Code

2-Criando o Projeto Spring Boot

Menu view > comand Pallet

Digite:Spring inicializr Maven Project

Configurar o Projeto:

Project: Maven Project

Language: Java

Spring Boot: 3.3.2

Group: br.com.api

Artifact: produtos

Packaging: Jar

Java: 17 (ou superior)

Dependências (funções):

Spring Web

Spring Boot DevTools

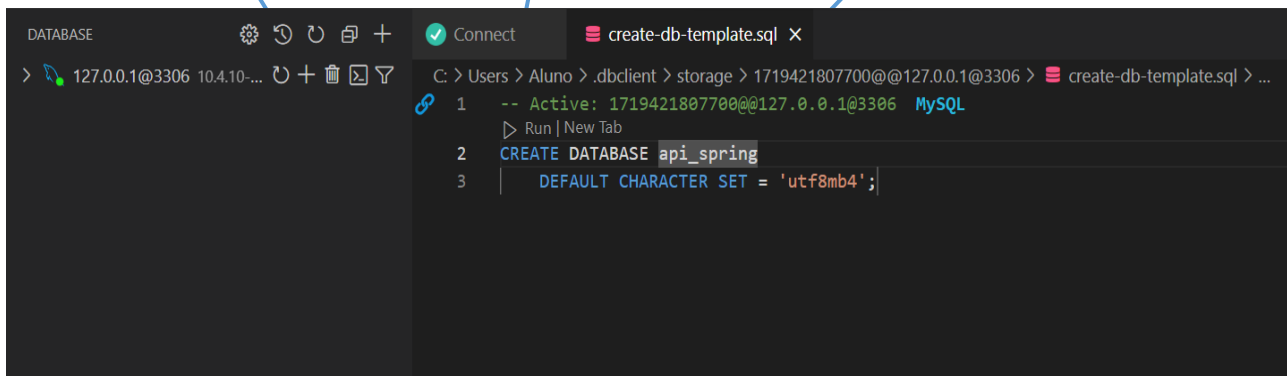
MySql(MySQL Driver SQL)

JPA(Spring Data JPA)

Lombok

3-Criando Base

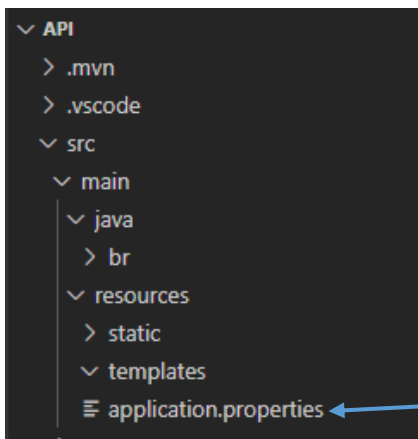
- 1-Clica em new database
- 3-Clique em Run
- 2-Digite o nome da base de dados
Spring_react



The screenshot shows a database client window with a dark theme. The left sidebar has a 'DATABASE' section with icons for settings, refresh, undo, redo, and a plus sign. The main area shows a terminal window with the following content:

```
C: > Users > Aluno > .dbclient > storage > 1719421807700@@127.0.0.1@3306 > create-db-template.sql > ...  
1 -- Active: 1719421807700@@127.0.0.1@3306 MySQL  
2 CREATE DATABASE api_spring  
3 | DEFAULT CHARACTER SET = 'utf8mb4';
```

Conexão com MySQL



Clique aqui

Altera a estrutura da tabela caso a entidade tenha mudanças.

spring.jpa.hibernate.ddl-auto=update

Acesso ao banco de dados

spring.datasource.url=jdbc:mysql://\${MYSQL_HOST:localhost}:3306/spring_react

Usuário do banco de dados

spring.datasource.username=root

Senha do banco de dados

spring.datasource.password=

2-Trabalhando com Modelos

O modelo é uma representação de dados, ele vai ter duas funcionalidades importantes que são: a manipulação de dados e a criação de tabelas.

Quando trabalhamos com uma API, geralmente temos muitos dados para manipular, e como podemos tornar fácil essa transição de dados? Através de um objeto, que será criado a partir de um modelo. Todo o dado que você queira receber ou enviar de uma API que não seja por url, deverá ter um modelo para o Spring saber como trabalhar com determinadas informações.

Em produtos crie uma nova pasta chamada modelo

Em modelo crie uma classe ProdutoModelo.java

Em modelo crie um arquivo RespostaModelo.java

```
package br.com.api.produtos.modelo;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.Getter;
import lombok.Setter;

@Entity
@Table(name= "produtos")
@Getter
@Setter
public class ProdutoModelo {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long codigo;
    private String nome;
    private String marca;
}
```

Estruturando projeto

“Pare o processo”

Clica em cima de produtos -> clica com o botão direito -> nova pasta

Pasta chamada **controle**

Dentro dessa pasta cria um arquivo chamado **ProdutoControle.java**

Clica em cima de produtos -> clica com o botão direito -> nova pasta

Pasta chamada **servico**

Dentro dessa pasta cria um arquivo chamado **ProdutoServico.java**

Clica em cima de produtos -> clica com o botão direito -> nova pasta

Pasta chamada **repositorio**

Dentro dessa pasta cria um arquivo chamado **ProdutoRepositorio.java**

```
public interface ProdutoRepositorio {  
  
  
  
  
  
  
  
  
  
}
```

Configurando repositório

```
1 package br.com.api.produtos.repositorio;  
2  
3 import org.springframework.data.repository.CrudRepository;  
4 import org.springframework.stereotype.Repository;  
5  
6 import br.com.api.produtos.modelo.ProdutoModelo;  
7  
8 @Repository  
9 public interface ProdutoRepositorio extends CrudRepository<ProdutoModelo, Long> {  
10  
11  
12  
13  
14 }  
15 |
```

Configurando controle

Clica em ProdutoControle

```
package br.com.api.produtos.controle;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ProdutoControle {

    @GetMapping("")
    public String rota(){
        return "api de produtos funcionando!";
    }
}
```

FASE1

Modelo para Validar Requisições

Quando especificamos que uma classe é um Componente, o Spring entenderá que podemos criar objetos e realizar diversas ações, porém o grande segredo em utilizar essa annotation é usufruir da injeção de dependência, **lembrando que a injeção de dependência é uma das características do Spring, fazendo com que objetos não precisem ser instanciados pelos desenvolvedores.**

Clica RespostaModelo.java

```
package br.com.api.produtos.modelo;

import org.springframework.stereotype.Component;

import lombok.Getter;
import lombok.Setter;

@Component
@Getter
@Setter
public class RespostaModelo {
    private String mensagem;
}
```

Serviço para listar produtos

“Pare o processo”

Utilizar a injeção de dependências para criarmos um objeto do tipo **ProdutoRepositorio**, em seguida iremos implementar um método que irá listar todos os nossos produtos, esse método irá utilizar uma função do nosso repositório chamado **findAll()**. O comando **findAll()** retorna uma informação do tipo **Iterable**, que nada mais é do que uma interface que determina uma coleção de dados.

Clica em ProdutoServico.java

```
package br.com.api.produtos.servico;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import br.com.api.produtos.modelo.ProdutoModelo;
import br.com.api.produtos.repositorio.ProdutoRepositorio;

@Service
public class ProdutoServico {

    @Autowired
    private ProdutoRepositorio pr;

    //Método para listar os produtos

    public Iterable<ProdutoModelo> listar(){
        return pr.findAll();
    }
}
```

Rota para listar produtos

Clica em ProdutoControle

```

package br.com.api.produtos.controle;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import br.com.api.produtos.modelo.ProdutoModelo;
import br.com.api.produtos.servico.ProdutoServico;

@RestController
public class ProdutoContole {
    @Autowired
    private ProdutoServico ps;

    @GetMapping("/listar")
    public Iterable< ProdutoModelo> listar(){
        return ps.listar();
    }

    @GetMapping("")
    public String rota(){
        return "api de produtos funcionando!";
    }
}

```

“Execute o processo”

Acesse: <http://localhost:8080/listar>

Estilos de formatação ☐

[]

“Pare o processo”

Serviço para cadastrar produtos

Utilizar a injeção de dependências para criarmos um objeto do tipo **ProdutoRepositorio**, em seguida iremos implementar um método que irá cadastrar os produtos, esse método irá utilizar uma função do nosso repositório chamado **save()**, que efetua uma inserção em nossa tabela, esse comando seria o mesmo que: **INSERT INTO produtos VALUES()**.

Clica em ProdutoServico


```

@Service
public class ProdutoServico {

    @Autowired
    private ProdutoRepositorio pr;

    @Autowired
    private RespostaModelo rm;

    //Método para listar todos os produtos
    public Iterable<ProdutoModelo> listar(){
        return pr.findAll();
    }

    //Método para cadastrar os produtos
    public ResponseEntity<?> cadastrar(ProdutoModelo pm){
        if(pm.getNome().equals("")){
            rm.setMensagem(mensagem:"O nome do produto é obrigatório");
            return new ResponseEntity<RespostaModelo>(rm,HttpStatus.BAD_REQUEST);
        }else if(pm.getMarca().equals("")){
            rm.setMensagem(mensagem:"A nome da marca é obrigatório");
            return new ResponseEntity<RespostaModelo>(rm, HttpStatus.BAD_REQUEST);
        }else{
            return new ResponseEntity<ProdutoModelo>(pr.save(pm),HttpStatus.CREATED);
        }
    }
}

```

Rota para cadastrar produtos

Acessa ProdutoControle

```

@RestController
public class ProdutoControle {

    @Autowired
    private ProdutoServico ps;

    @PostMapping("/cadastrar")
    public ResponseEntity<?> cadastrar(@RequestBody ProdutoModelo pm){
        return ps.cadastrar(pm);
    }

    @GetMapping("/listar")
    public Iterable<ProdutoModelo> listar(){
        return ps.listar();
    }

    @GetMapping("")
    public String rota(){
        return "api de produtos funcionando!";
    }
}

```

“Execute o processo”

Testando a rota cadastrar

POST localhost:8080/cadastrar Send

Status: 400 Bad Request Size: 47 Bytes Time: 160 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "nome": "",
3   "marca": ""
4 }
```

Response Headers 4 Cookies Results Docs

```
1 {
2   "mensagem": "O nome do produto é obrigatorio"
3 }
```

POST localhost:8080/cadastrar Send

Status: 201 Created Size: 55 Bytes Time: 452 ms

Query Headers 2 Auth Body 1 Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

JSON Content Format

```
1 {
2   "nome": "Motorola g600",
3   "marca": "motorola"
4 }
```

Response Headers 4 Cookies Results Docs

```
1 {
2   "codigo": 1,
3   "nome": "Motorola g600",
4   "marca": "motorola"
5 }
```

“Pare o processo”

Serviço para alterar produtos

Acessa ProdutoServico

```
//Método para cadastrar ou alterar os produtos
public ResponseEntity<?> cadastrarAlterar(ProdutoModelo pm, String acao){
    if(pm.getNome().equals("")){
        rm.setMensagem(mensagem:"O nome do produto é obrigatório");
        return new ResponseEntity<RespostaModelo>(rm,HttpStatus.BAD_REQUEST);
    }else if(pm.getMarca().equals("")){
        rm.setMensagem(mensagem:"A nome da marca é obrigatório");
        return new ResponseEntity<RespostaModelo>(rm, HttpStatus.BAD_REQUEST);

    }else{
        if(acao.equals("cadastrar")){
            return new ResponseEntity<ProdutoModelo>(pr.save(pm), HttpStatus.CREATED);
        }else{
            return new ResponseEntity<ProdutoModelo>(pr.save(pm),HttpStatus.OK);
        }
    }
}
}
```

Rota para alterar produtos

Acessa ProdutoControle

```
@PutMapping("/alterar")
public ResponseEntity<?> alterar(@RequestBody ProdutoModelo pm){
    return ps.cadastrarAlterar(pm, acao:"alterar");
}
```

“Execute o processo”

Acessa: localhost:8080/listar

Vamos testar

PUT	localhost:8080/alterar	Send	Status: 200 OK	Size: 54 Bytes	Time: 89 ms
Query	Headers 2	Auth	Body 1	Tests	Pre Run
JSON	XML	Text	Form	Form-encode	GraphQL
JSON Content			Format		
1 {					
2					
3 "codigo":2,					
4 "nome": "motorola g650",					
5 "marca": "motorola"					
6 }					
			Response	Headers 4	Cookies
			1 {		
			2 "codigo": 2,		
			3 "nome": "motorola g650",		
			4 "marca": "motorola"		
			5 }		

“Pare o processo”

Serviço para remover produtos

acessa produto serviço

```
//Método para remover produtos
public ResponseEntity<RespostaModelo> remover(long codigo){
    pr.deleteById(codigo);

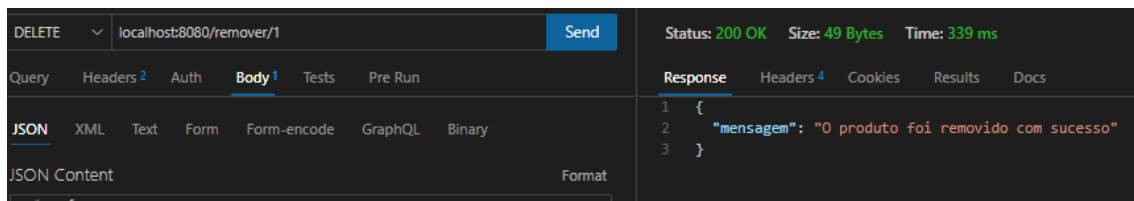
    rm.setMensagem(mensagem:"O produto foi removido com sucesso");
    return new ResponseEntity<RespostaModelo>(rm, HttpStatus.OK);
}
}
```

Rota para remover produtos

Acessa produto controle

```
@DeleteMapping("/remover/{codigo}")
public ResponseEntity<RespostaModelo> remover(@PathVariable long codigo){
    return ps.remover(codigo);
}
```

Vamos testar



FASE 2

CORS

Erros de CORS ocorrem devido uma segurança implementada nos navegadores. Uma página que esteja sendo executada em uma porta (exemplo: 4000), não pode ter acesso a dados em outros locais. Supondo que uma aplicação em Spring esteja na porta 8080 e uma aplicação em React esteja na porta 3000, o navegador irá recusar essa conexão.

Para resolvermos o problema de CORS, precisamos fazer com que nosso back-end libere o acesso para outras origens, utilizando o Spring há o comando **Crossrigin**, onde podemos especificar quais portas podem realizar requisições para nossa aplicação.

Crie uma pagina html

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Teste CORS</title>
```

```
<script>
```

```
    fetch('http://localhost:8080/listar') // Realiza a requisição

    .then(produtos => produtos.json())    // Converte o valor recebido em JSON

    .then(retorno => console.log(retorno)) // Exibindo os produtos
```

```
</script>
```

```
</head>
```

```
<body>
```

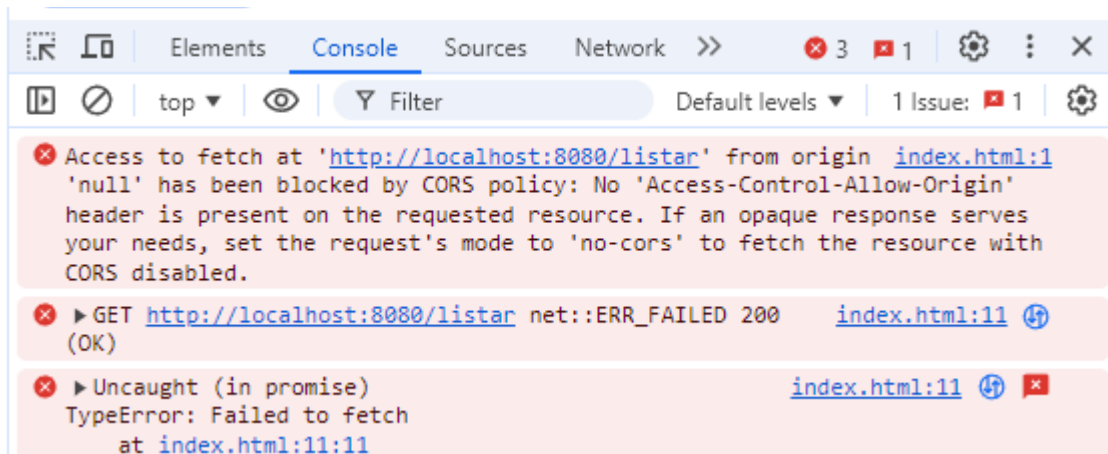
```
</body>
```

```
</html>
```

Salva a página como index.html

Abre a página> clica com botão direito>inspecionar>console

Aparecerá esse erro abaixo:

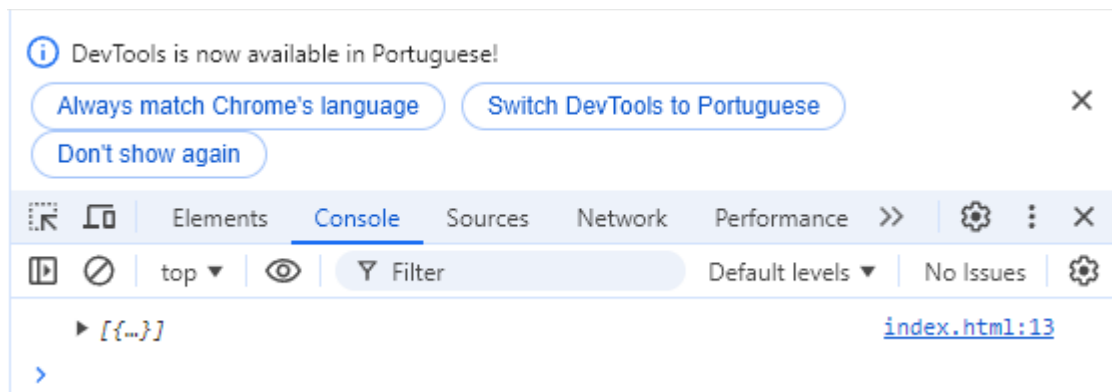


Em Produto Controle

Digite a anotation CrossOrigin

```
@RestController
@CrossOrigin(origins = "**")
public class ProdutoControle {
```

Após isso atualize o index.html



O erro será corrigido

Criando projeto React

Terminal -> New Terminal

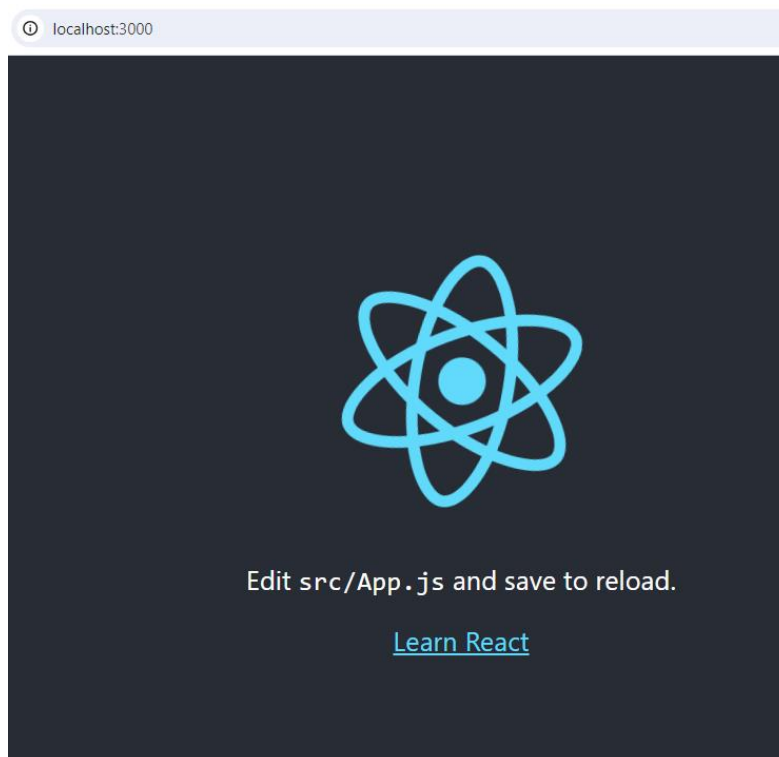
mkdir C:\Users\aluno.den\AppData\Roaming\npm

digite: **npx create-react-app front-end**

Selecione a pasta do projeto (File -> Open Folder)

Projeto aberto, abra o terminal do Visual Studio Code e digite: **npm start**

Espere alguns segundos e você terá em seu navegador o projeto em funcionamento, caso não apareça, abra uma aba do seu navegador e digite: **localhost:3000**



Criando componentes no React

Criar componentes responsáveis por gerar um formulário e uma tabela

Em src cria dois arquivos:

Formulário.js

Clica no arquivo e digita o código abaixo

```
JS Formulário.js > [?] default
function Formulário(){
  return(
    <h1>Formulário</h1>
  )
}

export default Formulário;
```

Tabela.js

Clica no arquivo e digita o código abaixo

```
> JS Tabela.js > [?] default
1 function Tabela(){
2   return(
3     <h1>Tabela</h1>
4   )
5 }
6
7 export default Tabela;
```

Clica no arquivo App.js

Apaga as informações do header, deixando apenas a div.

```
App.css
JS App.js
App.test.js
JS Formulário.js
index.css
JS index.js
logo.svg
JS reportWebVitals.js
JS setupTests.js
JS Tabela.js
.gitignore
package-lock.json
package.json
README.md

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
```

```
function App() {  
  return (  
    <div className="App">  
      |  
    </div>  
  );  
}  
  
export default App;
```

```
import './App.css';  
import Formulario from './Formulario';  
import Tabela from './Tabela';  
  
function App() {  
  return (  
    <div className="App">  
      <Formulario/>  
      <Tabela/>  
    </div>  
  );  
}  
  
export default App;
```

Digite <Formulário, caso não apareça a biblioteca automaticamente, digite.

Faça a mesma coisa para tabela

No navegador acessa o endereço

localhost:3000

Formulário

Tabela

Estrutura do formulário

No arquivo tabela crie a estrutura abaixo

```
function Formulario(){
  return(
    <form>
      <input type='text' placeholder="Nome" />
      <input type='text' placeholder="Marca" />

      <input type="button" value="Cadastrar" />
      <input type="button" value="Alterar" />
      <input type="button" value="Remover" />
      <input type="button" value="Cancelar" />
    </form>
  )
}

export default Formulario;
```

Estrutura da tabela

No arquivo tabela crie a estrutura abaixo:

```
function Tabela(){
  return(
    <table>
      <thead>
        <tr>
          <th>#</th>
          <th>Nome</th>
          <th>Marca</th>
          <th>Selecionar</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td></td>
          <td></td>
          <td></td>
          <td><input type="checkbox" /></td>
        </tr>
      </tbody>
    </table>
  )
}

export default Tabela;
```

Estilizando componentes

Acessa: <https://getbootstrap.com/>



Incluir via CDN

Quando você só precisa incluir o CSS ou JS compilado do Bootstrap, você pode usar [o jsDelivr](#). Veja-o em ação com nosso [início rápido](#) simples ou [navegue pelos exemplos](#) para dar início ao seu próximo projeto. Você também pode escolher incluir o Popper e nosso JS [separadamente](#).

Copia o link

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/b
```

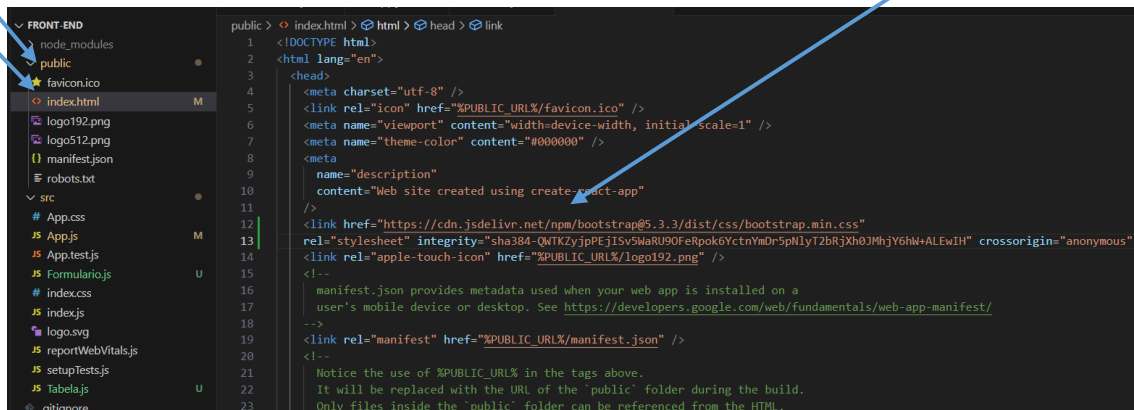


```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/b
```



2-Colar o link

1-Clica em
public, depois
em index.html



Abra o arquivo Formulario.js

Faz as seguintes alterações:

```
function Formulario(){
  return(
    <form>
      <input type="text" placeholder="Nome" className="form-control" />
      <input type="text" placeholder="Marca" className="form-control"/>

      <input type="button" value="Cadatrar" className="btn btn-primary"/>
      <input type="button" value="Alterar" className="btn btn-warning"/>
      <input type="button" value="Remover" className="btn btn-danger"/>
      <input type="button" value="Cancelar" className="btn btn-secondary"/>

    </form>
  )
}

export default Formulario;
```

Abra o arquivo Tabela.js

Faça as seguintes alterações:

```
<table className="table">
```

Em App.js apague o className="APP" da div

```
<div className="App">
```

Em App.css apague todas as informações que tem e digite:

```
form{
  width: 500px;
  margin: 30px auto;
  text-align: center;
}

form input{
  margin-bottom: 5px;
}

form input[type="button"]{
  margin-right: 10px;
}
```

Visibilidade dos botões

Implementar a visibilidade dos botões em nosso formulário, onde inicialmente o botão de cadastro estará ativo e os demais ocultos.

Em App.js

```
function App() {  
  const [btnCadastrar, setBtnCadastrar] = useState(true);  
  return (  
    <div>  
      <Formulario botao = {btnCadastrar}/>  
      <Tabela/>  
    </div>  
  );  
}  
  
export default App;
```

Em formulário.js

```
function Formulario({botao}){
```

```
  return(  
    <form>  
      <input type="text" placeholder="Nome" className="form-control" />  
      <input type="text" placeholder="Marca" className="form-control"/>  
      {  
        botao  
        ?  
        <input type="button" value="Cadastrar" className="btn btn-primary"/>  
        :  
        <div>  
          <input type="button" value="Alterar" className="btn btn-warning"/>  
          <input type="button" value="Remover" className="btn btn-danger"/>  
          <input type="button" value="Cancelar" className="btn btn-secondary"/>  
        </div>  
      }  
    )  
  )
```

FASE 3

Obtendo os produtos

File>new window

Abrindo a nova janela, clique em file> new folder> produtos

Executa o processo e minimiza a tela

Em App.js

```
import { useEffect, useState } from 'react';
import './App.css';
import Formulario from './Formulario';
import Tabela from './Tabela';

function App() {
  const [btnCadastrar, setBtnCadastrar] = useState(true)
  const [produtos, setProdutos] = useState([])

  useEffect(()=>{
    fetch("http://localhost:8080/listar")
      .then(retorno => retorno.json())
      .then(retorno_convertido => setProdutos(retorno_convertido));
  },[]);
  return (
    <div>
      <p>{JSON.stringify(produtos)}</p>
      <Formulario botao = {btnCadastrar}/>
      <Tabela/>
    </div>
  );
}
```

Ir  aparece os dados cadastrados

[{"c digo":2,"nome":"motorola g650","marca":"motorola"}]

Nome	Marca	Cadastrar
------	-------	-----------

#	Nome	Marca	Selecionar
---	------	-------	------------

Exibindo os produtos na tabela

Em App.js

```
return (
  <div>
    <Formulario botao = {btnCadastrar}/>
    <Tabela vetor={produtos}/>
  </div>
);
```

← Cria a propriedade vetor
que recebe produtos

Clica em Tabela.js

```
function Tabela({vetor}){
```

← Passa como par metro a
palavra vetor

```

<tbody>
  <tr>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
  </tr>
</tbody>

```

Recorta os tr's e os td's

Abra chaves

```

{
  vetor.map((obj, indice) => {
    <tr key={indice}>
      <td>{indice+1}</td>
      <td>{obj.nome}</td>
      <td>{obj.marca}</td>
      <td><button className="btn btn-success">Selecionar</button></td>
    </tr>
  })
}

```

Objeto produto

Criar nosso objeto responsável por manipularmos uma informação do tipo produto.

Clica em App.js

```

function App() {
  //Objeto produto
  const produto = {
    codigo: 0,
    nome: "",
    marca: ""
  }
}

```

UseState de produto

Criar o useState do tipo produto, assim conseguiremos manipular essa informação e fazer requisições em nossa API em Spring Boot.

Em App.js

```

const [btnCadastrar, setBtnCadastrar] = useState(true)
const [produtos, setProdutos] = useState([])
const [objProduto, setObjproduto] = useState(produto);

```

Cria uma usestate

Vamos testar

```
return (  
  <div>  
    <p>{JSON.stringify(objProduto)}</p>  
    <Formulario botao = {btnCadastrar}/>  
    <Tabela vetor={produtos}/>  
  </div>  
)
```

Escreva

```
{"código":0,"nome":"","marca":""}
```

Nome	Marca	Cadastrar
------	-------	-----------

Obtendo dados do formulário

Em App.js

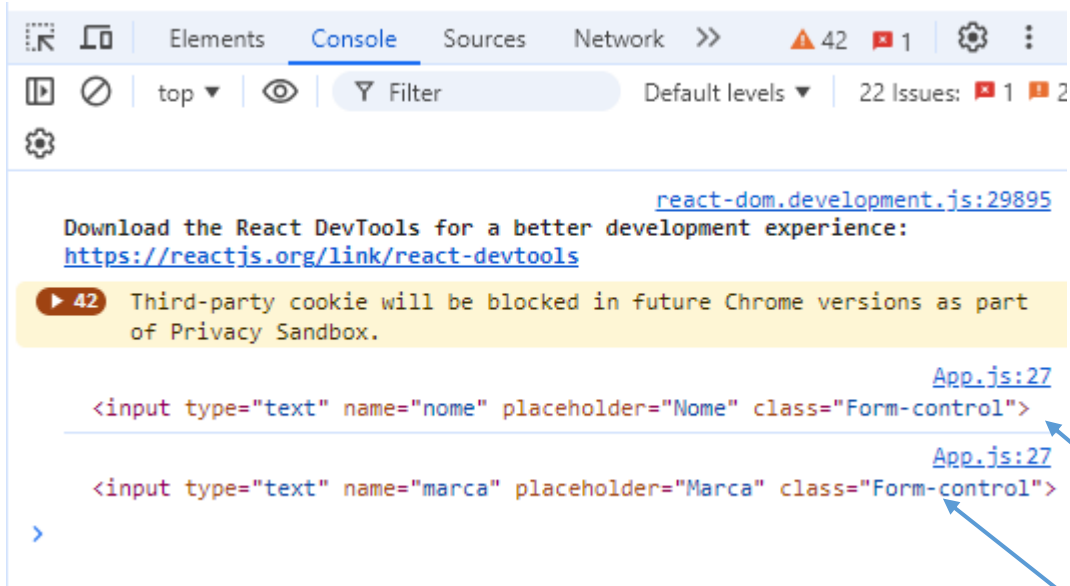
```
const aoDigitar = (e) =>{  
  console.log(e.target);  
}  
return (  
  <div>  
    <p>{JSON.stringify(objProduto)}</p>  
    <Formulario botao = {btnCadastrar} eventoTeclado={aoDigitar}/>  
    <Tabela vetor={produtos}/>  
  </div>  
)
```

Em formulário

```
function Formulario(botao, eventoTeclado){  
  return(  
    <form>  
  
      <input type="text" onChange={ eventoTeclado}name="nome" placeholder="Nome" className=  
      <input type="text" onChange={ eventoTeclado}name="marca" placeholder="Marca" className=
```

Vamos testar

Na página, clica com o botão direito>inspecionar>console



Se digitar no campo
nome e marca
aparecerá o input

O evento está trazendo a informação

Em App.js

```
const aoDigitar = (e) =>{  
  setObjproduto({...objProduto, [e.target.name]:e.target.value});  
}  
return (
```

no navegador

Ao digitar no campo
nome e marca
aparecerá na
listagem

{ "codigo":0,"nome":"moto g","marca":"motorola" }

Cadastrar produto

```
const aoDigitar = (e) =>{
  setObjproduto({...objProduto, [e.target.name]:e.target.value});
}

//Cadastrar Produto
const cadastrar = () =>{
  fetch("http://localhost:8080/cadastrar",{
    method:"post",
    body:JSON.stringify(objProduto),
    headers:{
      "Content-type":"application/json",
      "Accept":"application/json"
    }
  })
  .then(retorno =>retorno.json())
  .then(retorno_convertido=>{
    console.log(retorno_convertido);
  })
}
return (
  <div>

    <Formulario botao={btnCadastrar} eventoTeclado={aoDigitar} cadastrar={cadastrar}/>
    <Tabela vetor={produtos}/>
  </div>
)
```

Criamos o método
cadastrar

Adicionamos ao
botão no formulário

Passa o parâmetro
cadastrar e ação
onclick no botão
cadastrar

Em formulário.js

```
function Formulario({botao, eventoTeclado, cadastrar}){
  return(
    <form>

    <input type="text" onChange={ eventoTeclado} name="nome" placeholder="Nome" className="Form-control"/>
    <input type="text" onChange={ eventoTeclado} name="marca" placeholder="Marca" className="Form-control"/>

    {
      botao
      ?
      <input type="button" value="Cadastrar" onClick={cadastrar}className="btn btn-primary" />
    }
  )
}
```

Vamos testar (Botão direito>inspecionar>console)

Nome	Marca	Cadastrar
------	-------	-----------

Nome	Marca	Selecionar
motorola g650	motorola	Selecionar
Moto g50	motorola	Selecionar
galaxy 355	sansumg	Selecionar
iphone 55	apple	Selecionar

DevTools is now available in Portuguese!

Always match Chrome's language | Switch DevTools to Portuguese | Don't show again

Download the React DevTools for a better development experience:
<https://reactjs.org/link/react-devtools>

POST http://localhost:8080/cadastrar 400 (Bad Request) App.js:32

{mensagem: 'O nome do produto é obrigatorio'} App.js:42

Cadastrando produto

 Cadastrar

► {codigo: 153, nome: 'sas', marca: 'sasa'}

[App.js:42](#)

Atualizando a página

#	Nome	Marca	Selecionar
1	motorola g650	motorola	<button>Selecionar</button>
2	Moto g50	motorola	<button>Selecionar</button>
3	galáxia 355	Sansugestão	<button>Selecionar</button>
4	iphone 55	maçã	<button>Selecionar</button>
5	sas	sasa	<button>Selecionar</button>

preencher

Vamos

```
.then(retorno =>retorno.json())
.then(retorno_convertido=>{
  if(retorno_convertido.mensagem !==undefined){
    alert(retorno_convertido.mensagem);
  }else{
    setProdutos([...produtos, retorno_convertido]);
    alert("Produto cadastrado com sucesso!")
  }
})
```

FASE 4

Limpar formulário

Em App.js crie o método limpar formulário

```
}

//Limpar Formulário
const limparFormulario = ()=>{
  setObjproduto(produto);
}
return (
```

Passa a função para o componente do formulário

```
return (
  <div>
    <Formulario botao = {btnCadastrar} eventoTeclado={aoDigitar} cadastrar={cadastrar} obj={objProduto}/>
    <Tabela vetor={produtos}/>
  </div>
)
```

```
.then(retorno =>retorno.json())
.then(retorno_convertido=>{
  if(retorno_convertido.mensagem !==undefined){
    alert(retorno_convertido.mensagem);
  }else{
    setProdutos([...produtos, retorno_convertido]);
    alert("Produto cadastrado com sucesso!")
    limparFormulario();
  }
})
```

No final do método
cadastrar informe o
método limpar

Em formulário.js

Passe o parâmetro obj

```
function Formulario({botao, eventoTeclado, cadastrar, obj})
```

```
<form>
  <input type="text" value={obj.nome} onChange={ eventoTeclado} />
  <input type="text" value={obj.marca} onChange={ eventoTeclado} />
```

Informa os valores no
input

Vamos testar

Cadastre o produto e verifique se os campos limpam

Selecionar produto

Em App.js crie o método selecionar produto

```
//Selecionar Produto
const selecionarProduto =(indice) =>{
  setObjproduto(produtos[indice]);
  setBtnCadastrar(false);
}
return (
```

Informa a função para o
componente tabela

```
<Tabela vetor={produtos} selecionar = {selecionarProduto}/>
```

Em Tabela.js

```
function Tabela({vetor, selecionar}){
```

Passa o parâmetro

No botão passa um
evento de click

```
<td><button onClick={()=>{selecionar(indice)}} className="btn btn-success">Selecionar</button></td>
```

Vamos testar

cc	vvv	
Alterar	Remover	Cancelar

Marca	Selecionar
motorola	Selecionar
motorola	Selecionar
apple	Selecionar
apple	Selecionar
sasa	Selecionar
ssssssss	Selecionar
nnnnnnnn	Selecionar
vvv	Selecionar

Clicando no botão
selecionar aparecerá as
informações e os
botões

Cancelando alteração e exclusão

Reutilizando a função
limpar, utilize o
setBtnCadastrar(true)
para exibir o botão
cadastrar

```
//Limpar Formulário
const limparFormulario = ()=>{
  setObjProduto(produto);
  setBtnCadastrar(true);
}
```

No retorno formulário informe a propriedade cancelar recebendo limpar formulário

```
<Formulario botao = {btnCadastrar} eventoTeclado={aoDigitar} cadastrar={cadastrar} obj={objProduto} cancelar={limparFormulario}/>
```

Em formulário.js passe o parâmetro cancelar

```
function Formulario({botao, eventoTeclado, cadastrar, obj, cancelar}){
```

No botão de cancelar escreva a função de click

```
<input type="button" value="Alterar" className="btn btn-warning"/>
<input type="button" value="Remover" className="btn btn-danger" />
<input type="button" value="Cancelar" onClick={cancelar} className="btn btn-secondary" />
```

Remover produto

Em App.js

```

//Remover Produto
const remover = () =>{
  fetch("http://localhost:8080/remover/"+objProduto.codigo,{
    method:"delete",

    headers:{
      "Content-type":"application/json",
      "Accept":"application/json"
    }
  })
  .then(retorno =>retorno.json())
  .then(retorno_convertido=>{

    //Mensagem

    alert(retorno_convertido.mensagem);

    //cópia do vetor produtos
    let vetorTemp = [...produtos];

    //indice
    let indice = vetorTemp.findIndex((p)=>{
      return p.codigo===objProduto.codigo;
    });

    //Remover produto do vetor
    vetorTemp.splice(indice, 1);

    //Atualizar o vetor de produtos

    setProdutos(vetorTemp);

    //limpar formulário
    limparFormulario();
  })
}

```

No componente formulário em app.js escreva propriedade

```
remover={remover}/>
```

Em formulário.js passe o parâmetro

```
function Formulario({botao, eventoTeclado, cadastrar, obj, cancelar, remover}){
```

No botão remover

```
<input type="button" value="Remover" onClick={remover} className="btn btn-danger" />
```

Alterar produto

Em App.js

```
//Alterar Produto
const alterar = () =>{
  fetch("http://localhost:8080/alterar",{
    method:"put",
    body:JSON.stringify(objProduto),
    headers:{
      "Content-type":"application/json",
      "Accept":"application/json"
    }
  })
  .then(retorno =>retorno.json())
  .then(retorno_convertido=>{
    if(retorno_convertido.mensagem !== undefined){
      alert(retorno_convertido.mensagem);
    }else{
      //mensagem
      alert("Produto alterado com sucesso!");

      //cópia do vetor produtos
      let vetorTemp = [...produtos];

      //indice
      let indice = vetorTemp.findIndex((p)=>{
        return p.codigo===objProduto.codigo;
      });

      //Alterar produto do vetor
      vetorTemp[indice] = objProduto;
      //Atualizar o vetor de produtos

      setProdutos(vetorTemp);

      //limpar formulario
      limparFormulario();
    }
  })
}
```

No componente formulário

```
alterar={alterar}/>
```

Em Formulario.js

Passa o parâmetro alterar, no botão alterar informa a função onclick.

```
function Formulario({botao, eventoTeclado, cadastrar, obj, cancelar, remover, alterar}){
  return(
    <form>

    <input type="text" value={obj.nome} onChange={ eventoTeclado} name="nome" placeholder="Nome" className="Form-control"/>
    <input type="text" value={obj.marca} onChange={ eventoTeclado} name="marca" placeholder="Marca" className="Form-control"/>

    {
      botao
      ?
      <input type="button" value="Cadastrar" onClick={cadastrar} className="btn btn-primary" />
      :
      <div>
        <input type="button" value="Alterar" onClick={alterar} className="btn btn-warning"/>
        <input type="button" value="Remover" onClick={remover} className="btn btn-danger" />
        <input type="button" value="Cancelar" onClick={cancelar} className="btn btn-secondary" />
      </div>
    }
  )
}
```