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 initializr 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

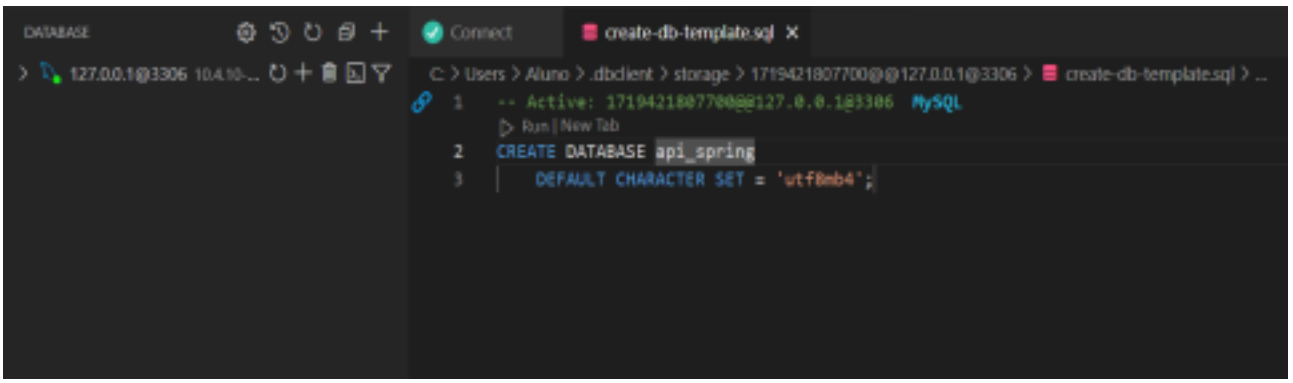
3-Criando Base

3-Clique em Run

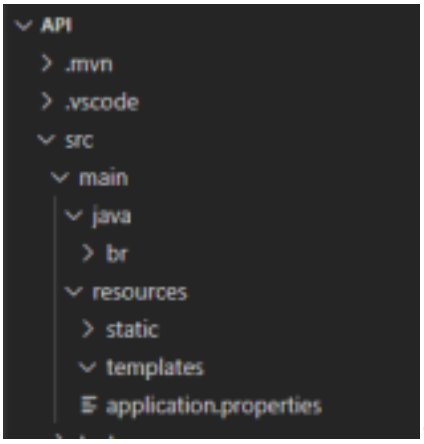
1-Clica em new database

2-Digite o nome da base de dados

Spring_react



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 termos 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

```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

```java
package br.com.api.produtos.repositorio;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

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

@Repository
public interface ProdutoRepositorio extends CrudRepository<ProdutoModelo, Long> {



}
```

Configurando controle

Clica em ProdutoControle

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

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

@RestController
public class ProdutoContole {

    @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

```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

```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

```java
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

```java
@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);
        }
    }
}
```

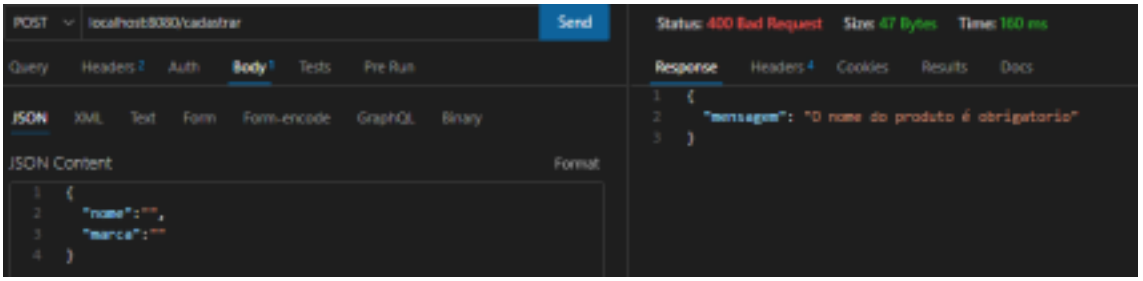
Rota para cadastrar produtos

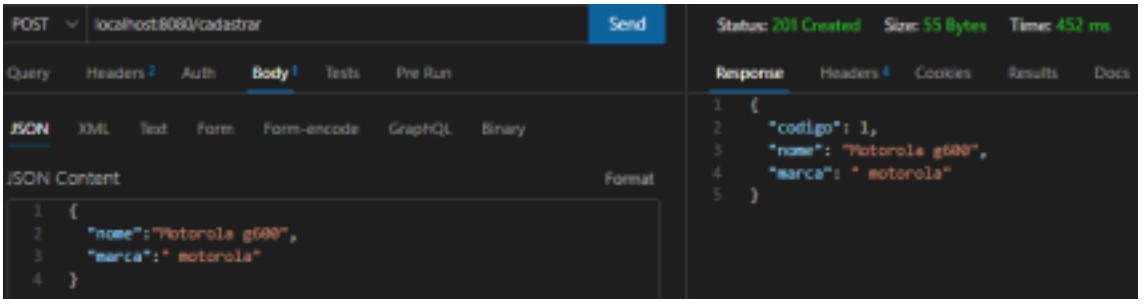
Acessa ProdutoControle

```java
@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

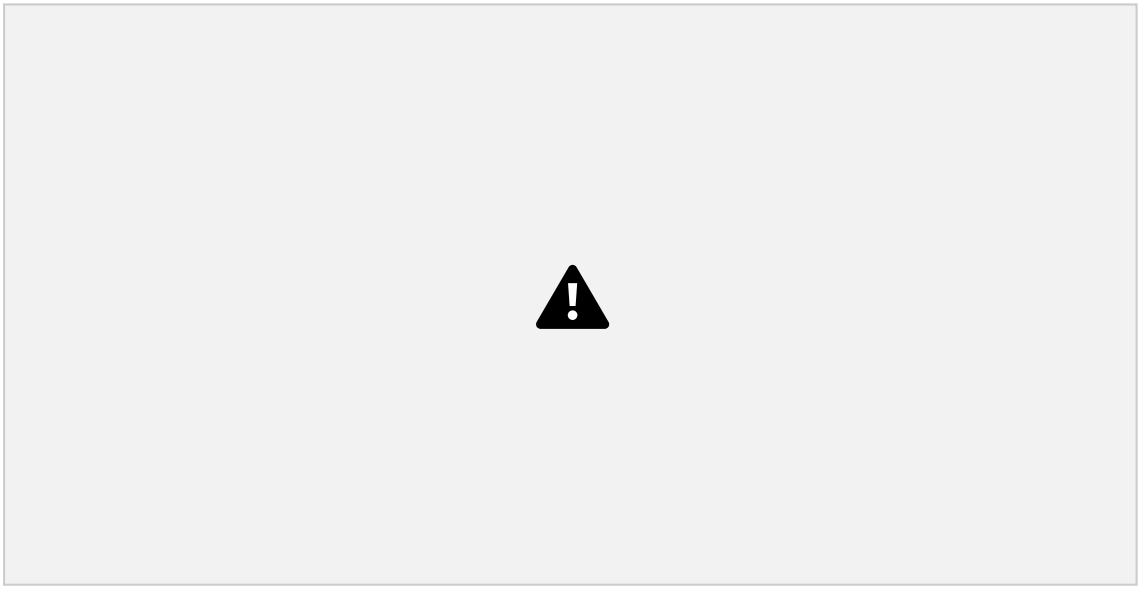




“Pare o processo”

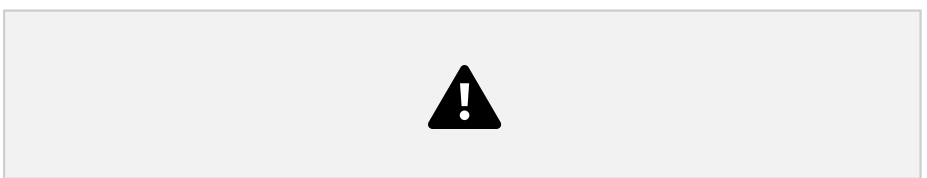
Serviço para alterar produtos

Acessa ProdutoServico



Rota para alterar produtos

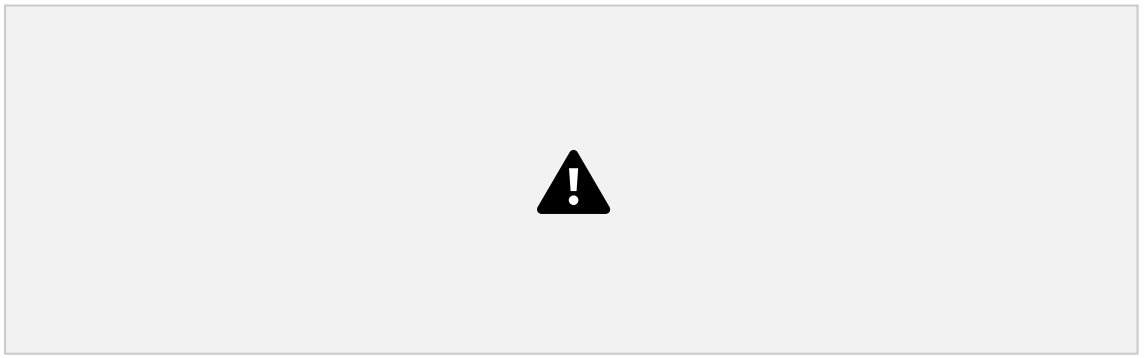
Acessa ProdutoControle



“Execute o processo”

Acessa: localhost:8080/listar

Vamos testar



“Pare o processo”

Serviço para remover produtos
acessa produto serviço

Rota para remover produtos
Acessa produto controle

Vamos testar

FASE 2

CORS

Erros de CORS ocorrem devido uma segurança implementanda 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.

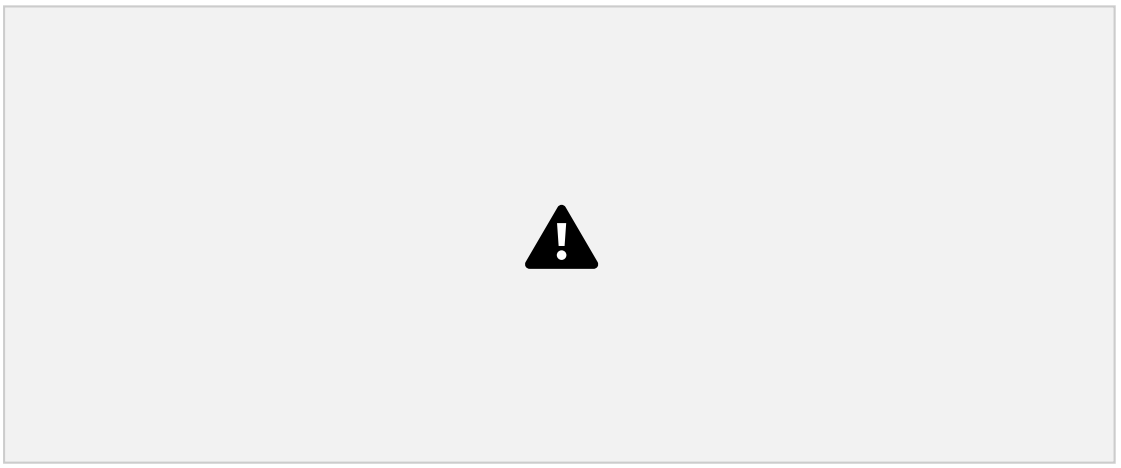
Crie uma pagina html

```
<!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

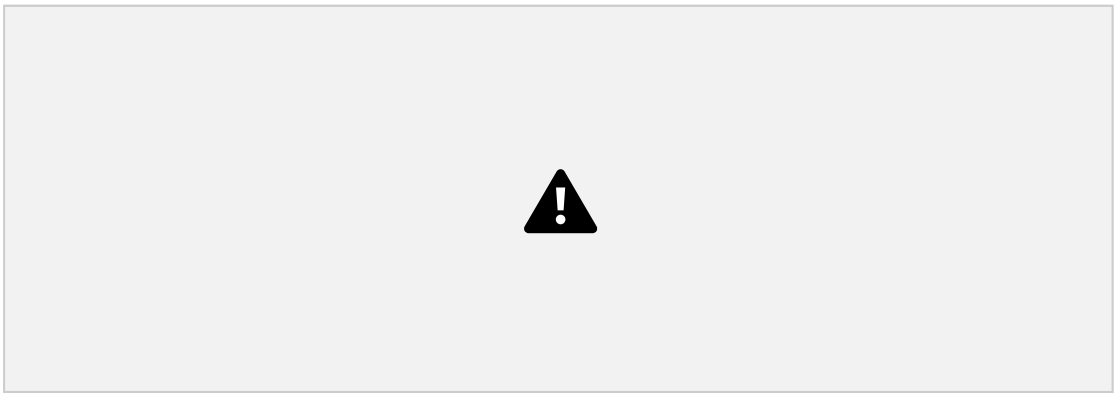
Aparecerá esse erro abaixo:



Em Produto Controle

Digite a anotation CrossOrigin

Após isso atualize o index.html



O erro será corrigido

Criando projeto React

Terminal -> New Terminal

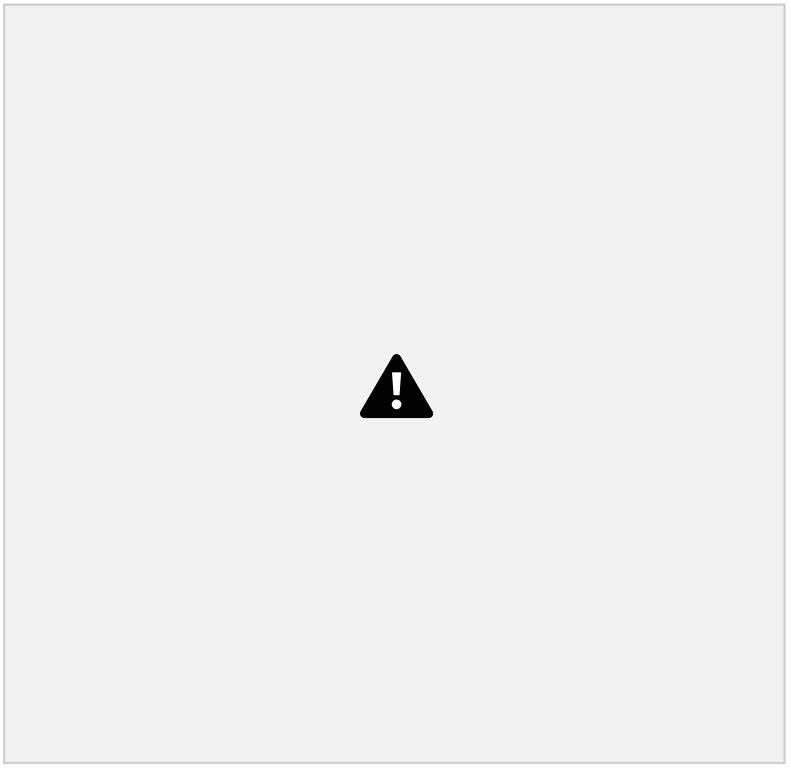
mkdir C:\Users\aluno.den\AppData\Roaming\npm

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 se 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

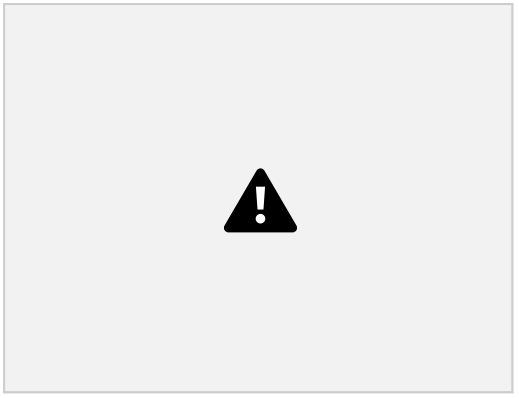
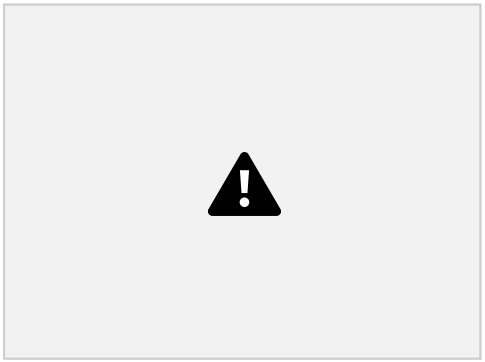


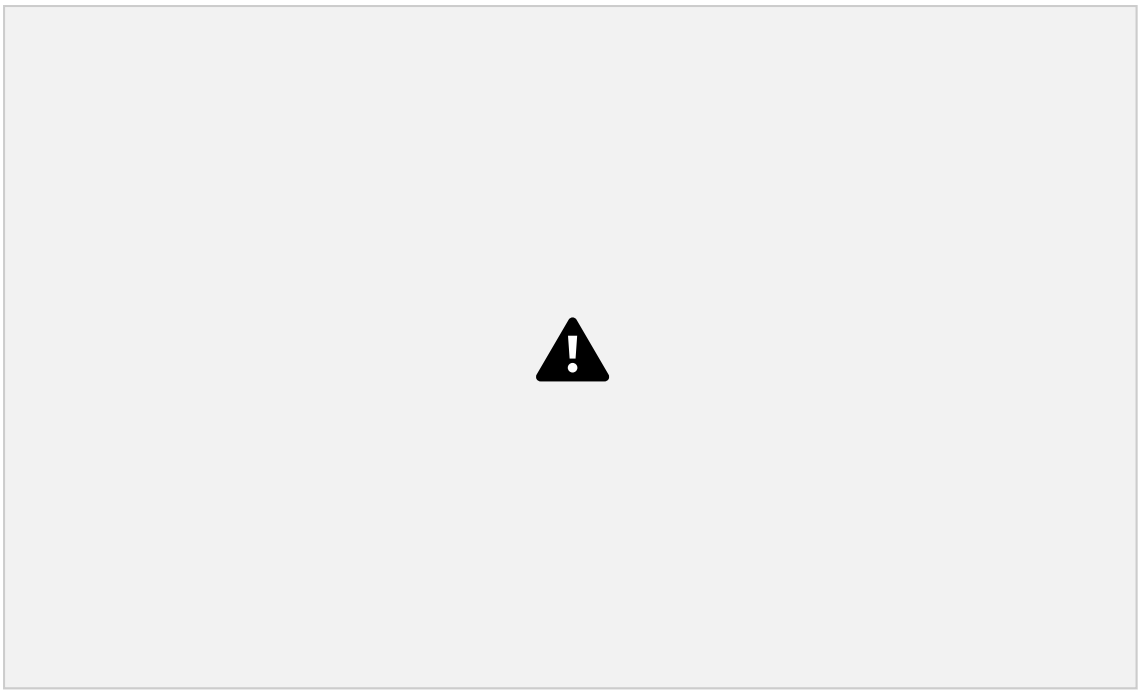
Tabela.js

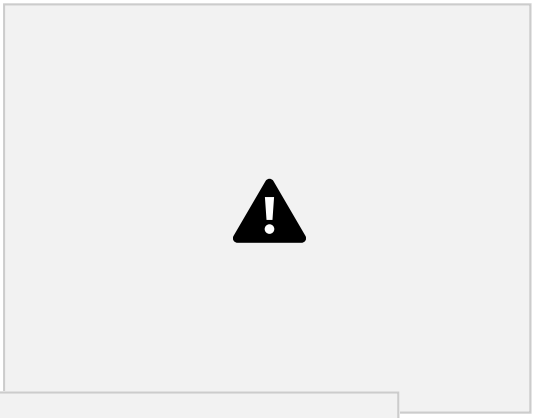
Clica no arquivo e digita o código abaixo

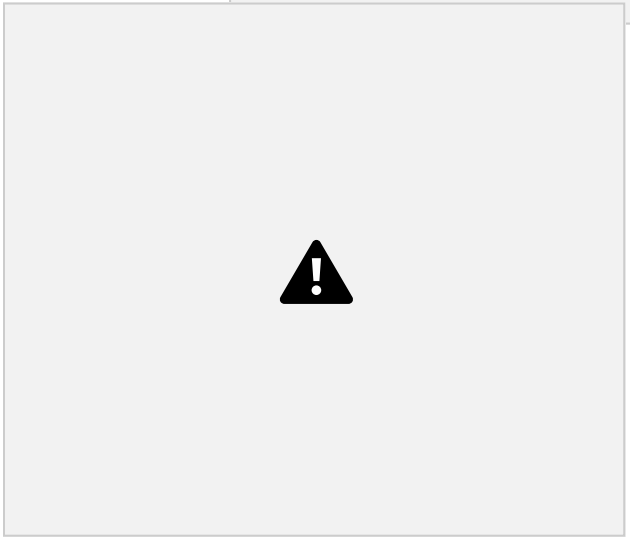


Clica no arquivo App.js

Apaga as informações do header, deixando apenas a div.



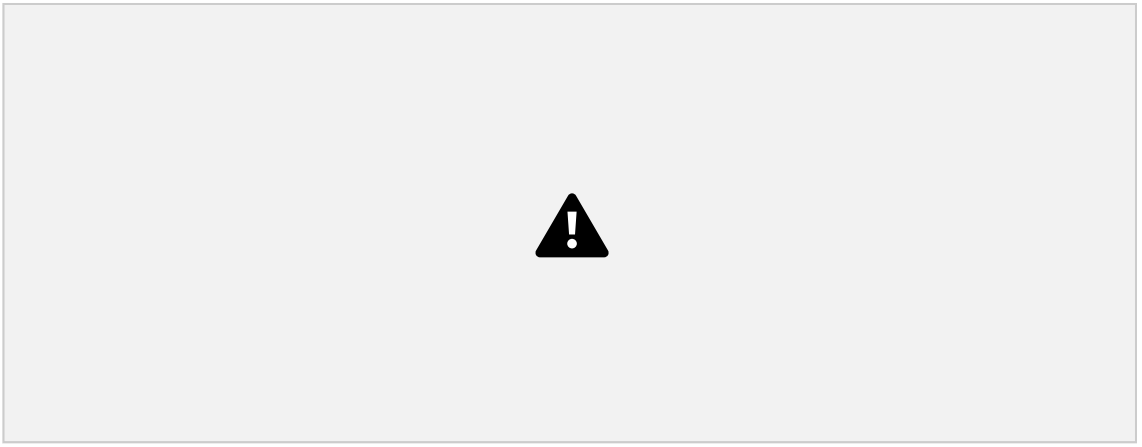




Digite <Formulário, caso não apareça a biblioteca automaticamente, digite.

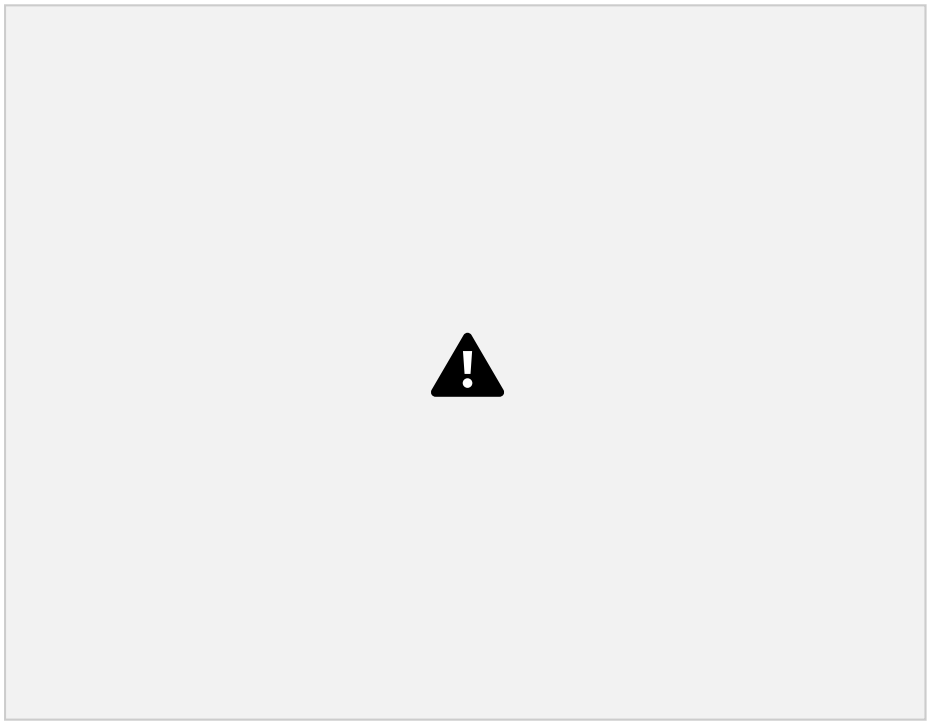
Faça a mesma coisa para tabela

No navegador acessa o endereço



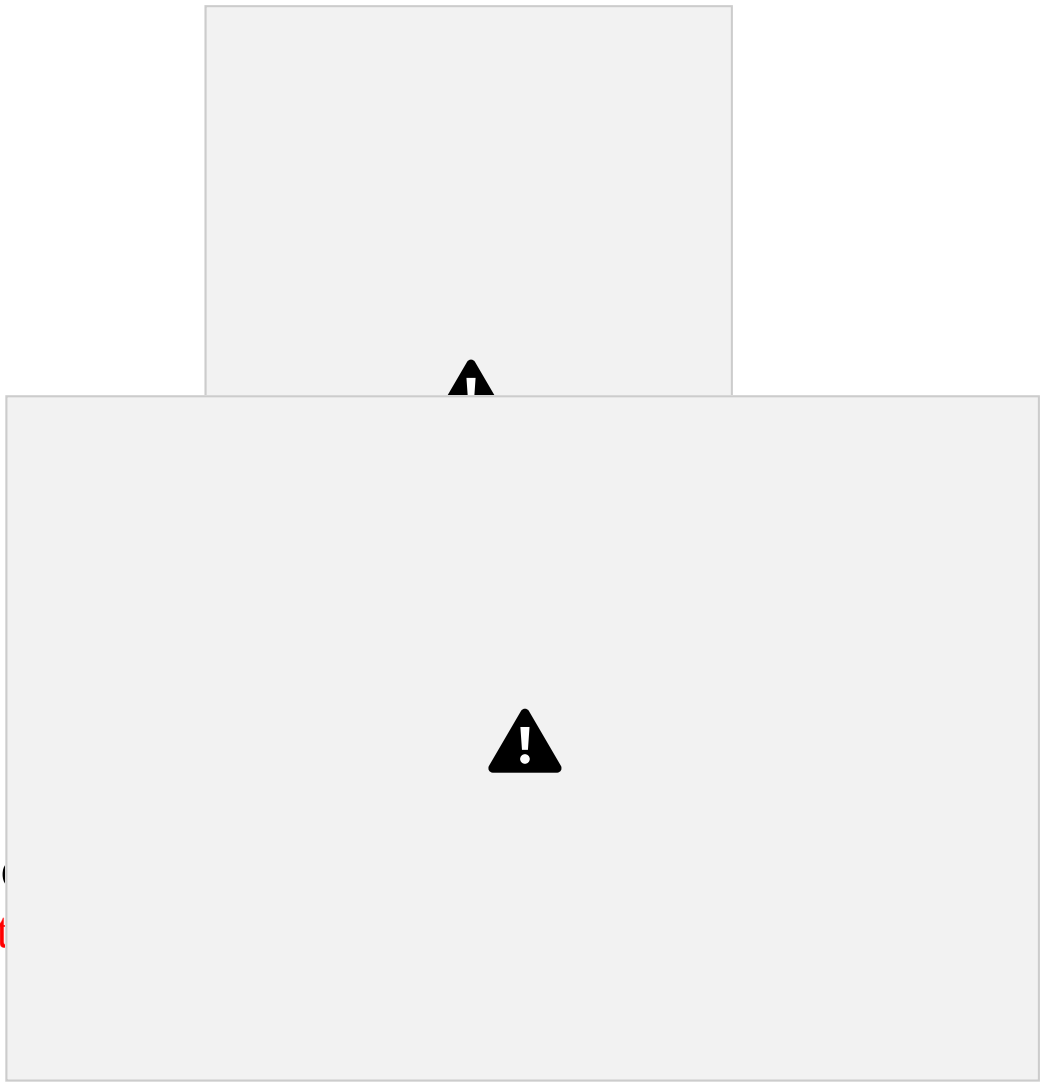
Estrutura do formulário

No arquivo tabela crie a estrutura abaixo



Estrutura da tabela

No arquivo tabela crie a estrutura



Estilizan

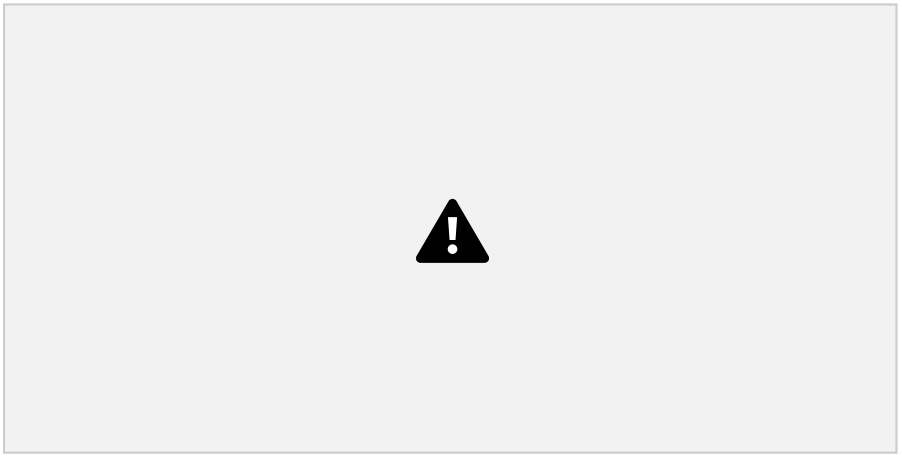
Acessa: htt

Copia o link

2-Cola o link

1-Clica em public, depois em index.html

Abra o arquivo Formulario.js
Faz as seguintes alterações:

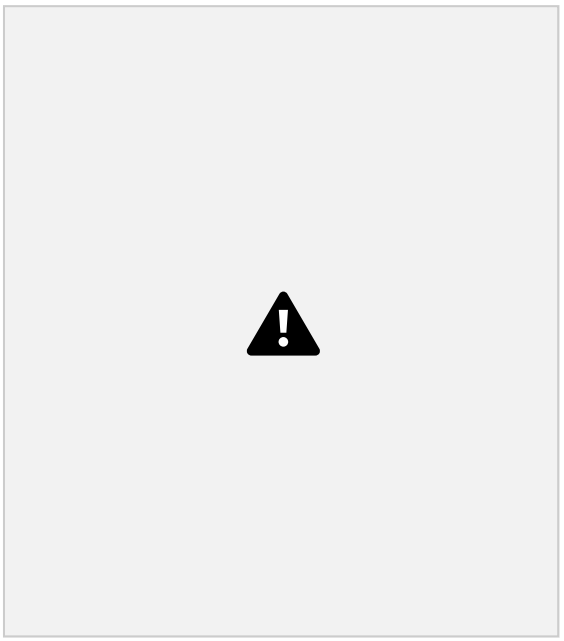


Abra o arquivo Tabela.js

Faça as seguintes alterações:

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

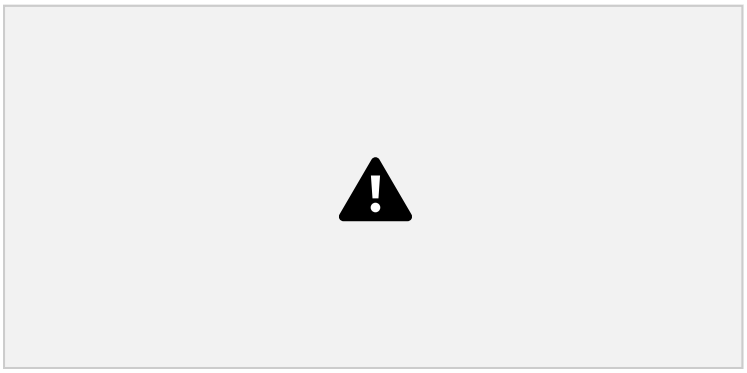
Em App.css apague todas as informações que tem e

digite: 

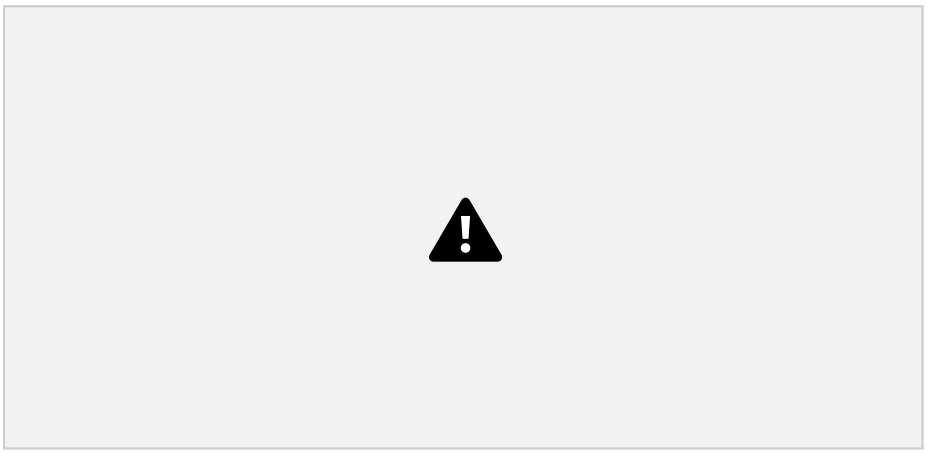
Visiblidade dos botões

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

Em App.js



Em formulário.js



FASE 3

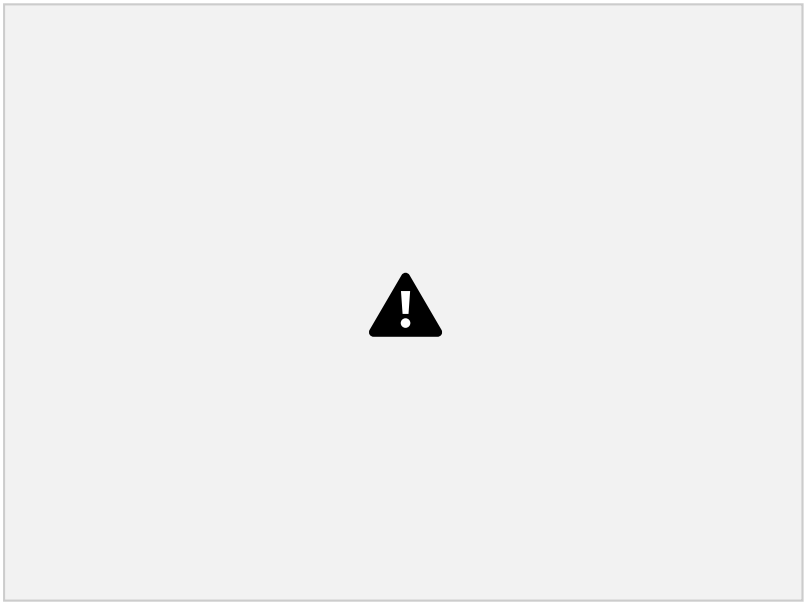
Obtendo os produtos

File>new window

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

produtos Executa o processo e minimiza a tela

Em App.js



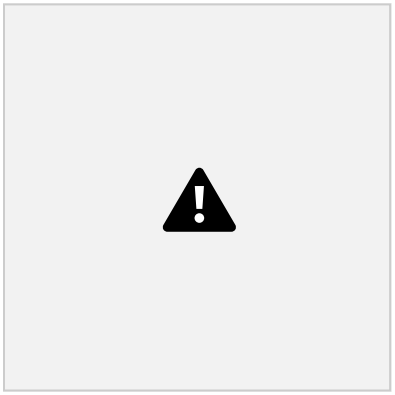
Irá aparece os dados cadastrados

tabela

Cria a propriedade vetor que recebe produtos

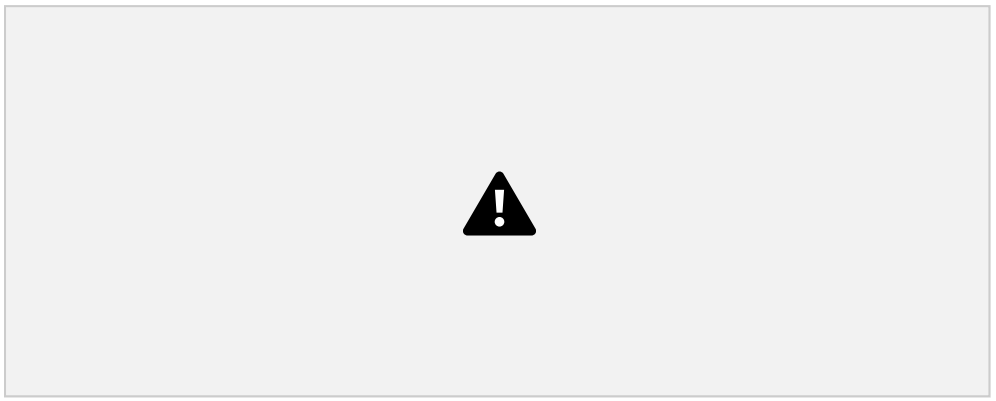
Clica em Tabela.js

Passa como parâmetro a palavra vetor



Recorta os tr´s e os td´s

Abra chaves



Objeto produto

Criar nosso objeto responsável por manipularmos uma informação do tipo produto. Clica em App.js

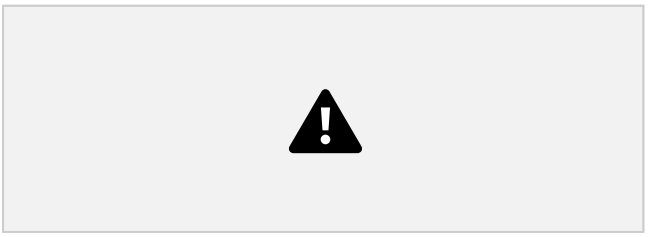
UseState de produto

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

Em App.js

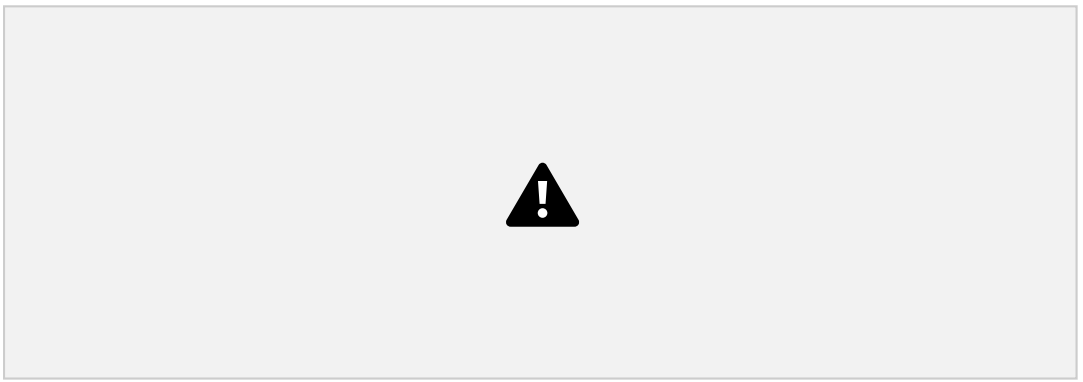

Cria uma usestate

Vamos testar


Escreva

Obtendo dados do formulário

Em App.js

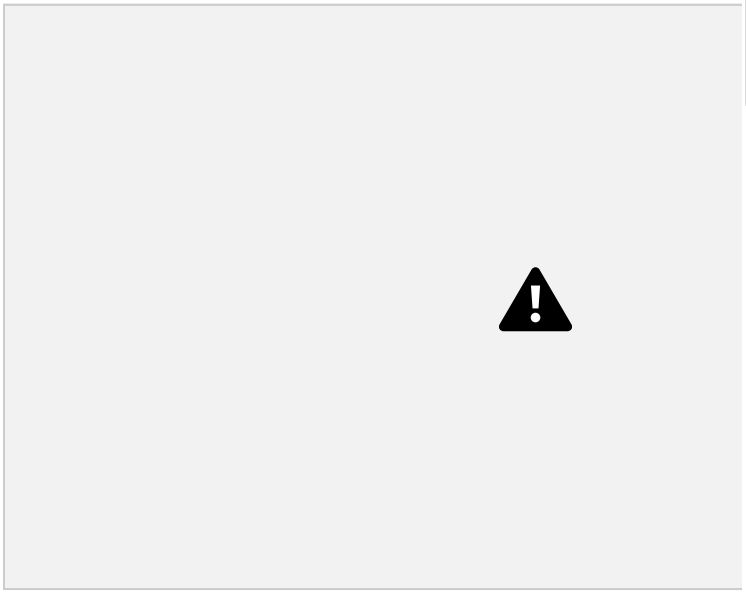


Em formulário

Vamos testar

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

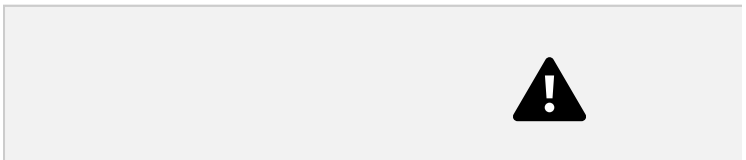
no navegador



O evento está trazendo a informação

Se digitar no campo nome e marca aparecerá o input

Em App.js

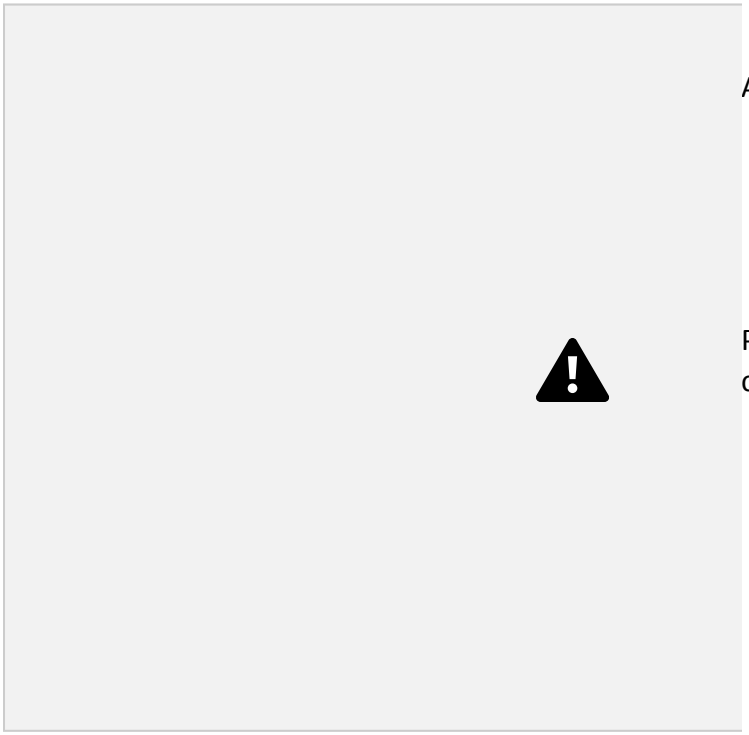


Vamos testar (Botão direito>inspecionar>console)

Criamos o método cadastrar

Ao digitar no campo nome e marca aparecerá na
listagem

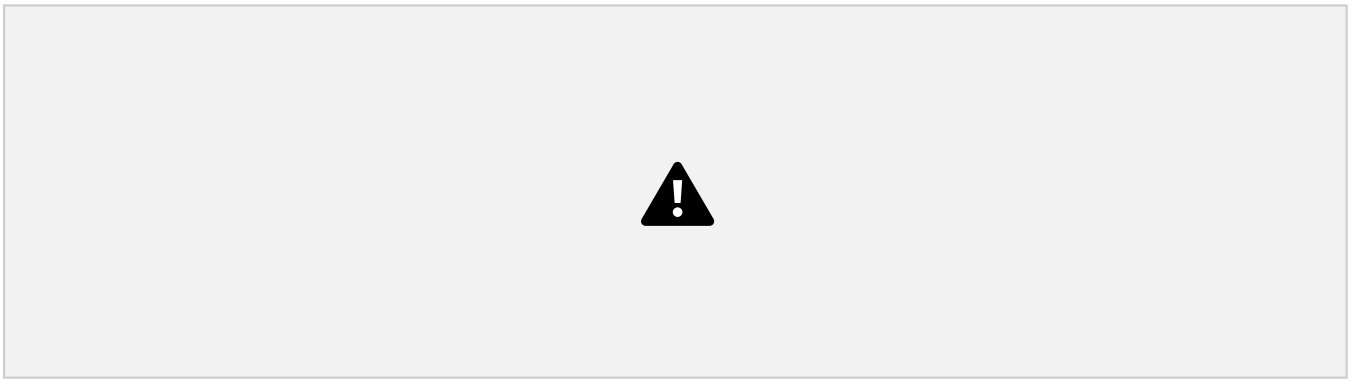
Cadastrar produto



Adicionamos ao botão no formulário

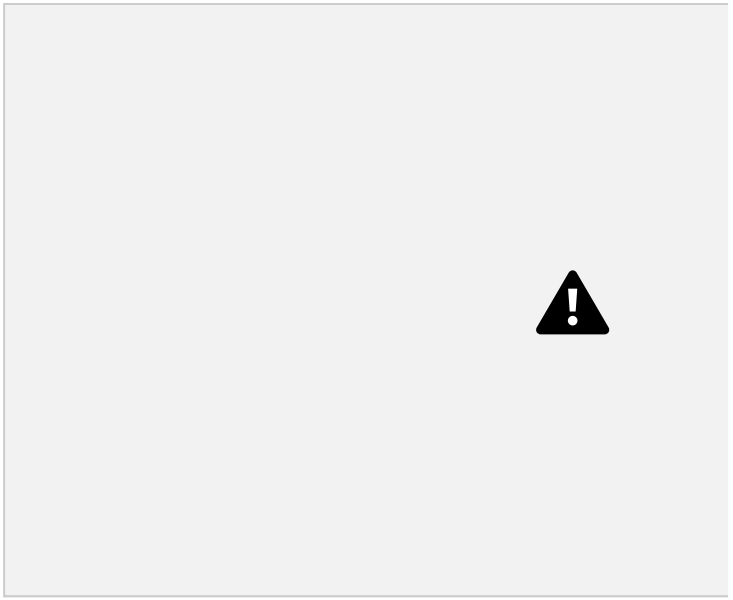
Passa o parâmetro cadastrar e ação onclick no botão cadastrar

Em formulário.js

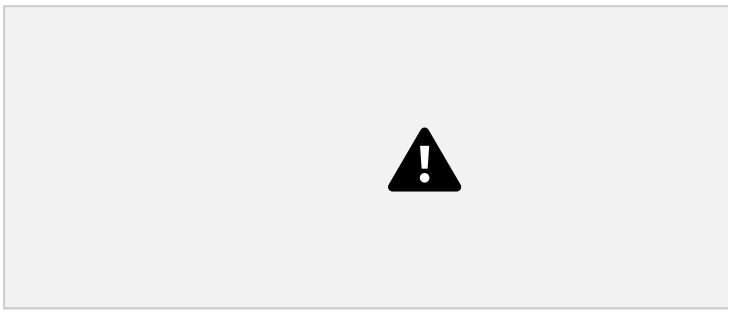


Cadastrando produto

Atualizando a página



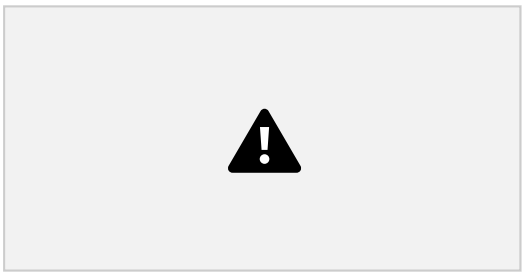
preencher



FASE 4

Limpar formulário

Em App.js crie o método limpar formulário



Passa a função para o componente do formulário

Vamos

No final do método cadastrar informe o método limpar

Em formulário.js

Informa os valores no input

Vamos testar
Cadastre o produto e verifique se os campos limpam

Selecionar produto

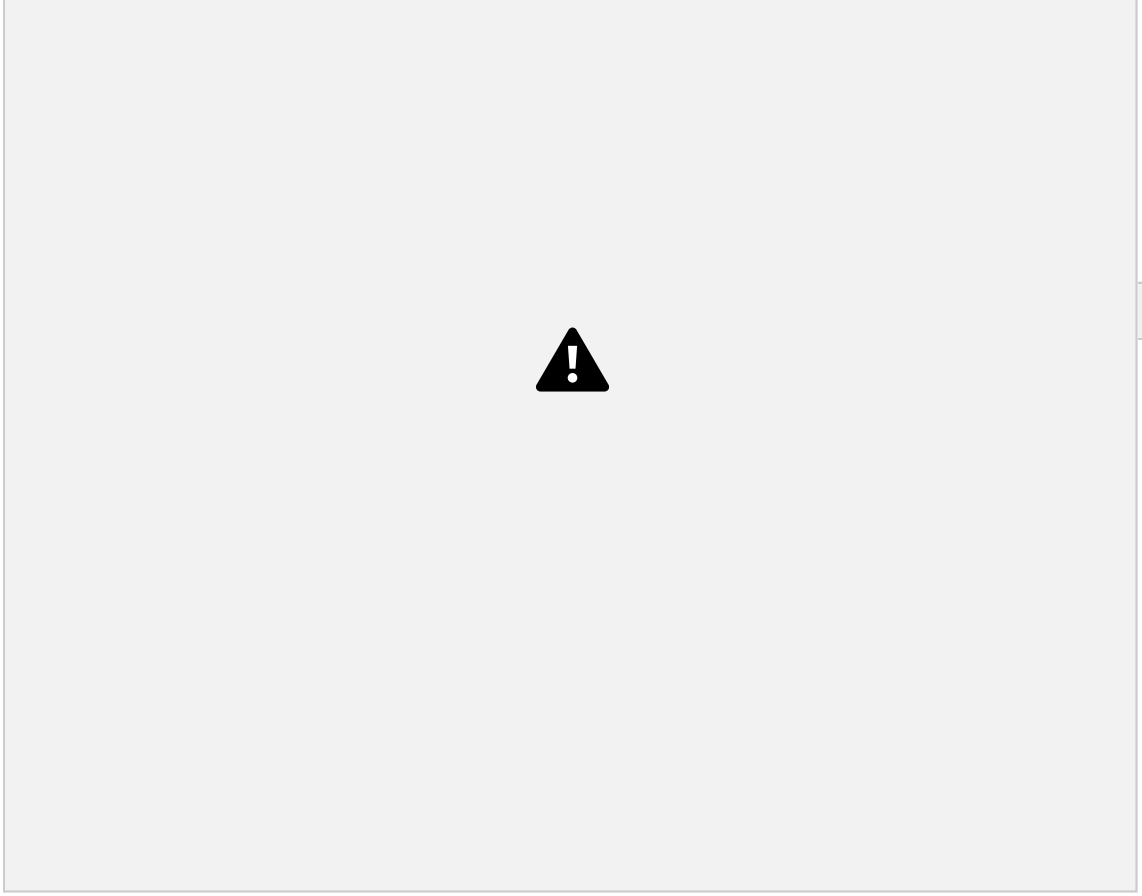
Em App.js crie o método selecionar produto

Informa a função para o componente tabela
Em Tabela.js

Passe o parâmetro

Passe o parâmetro obj

No botão passa um evento de click



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

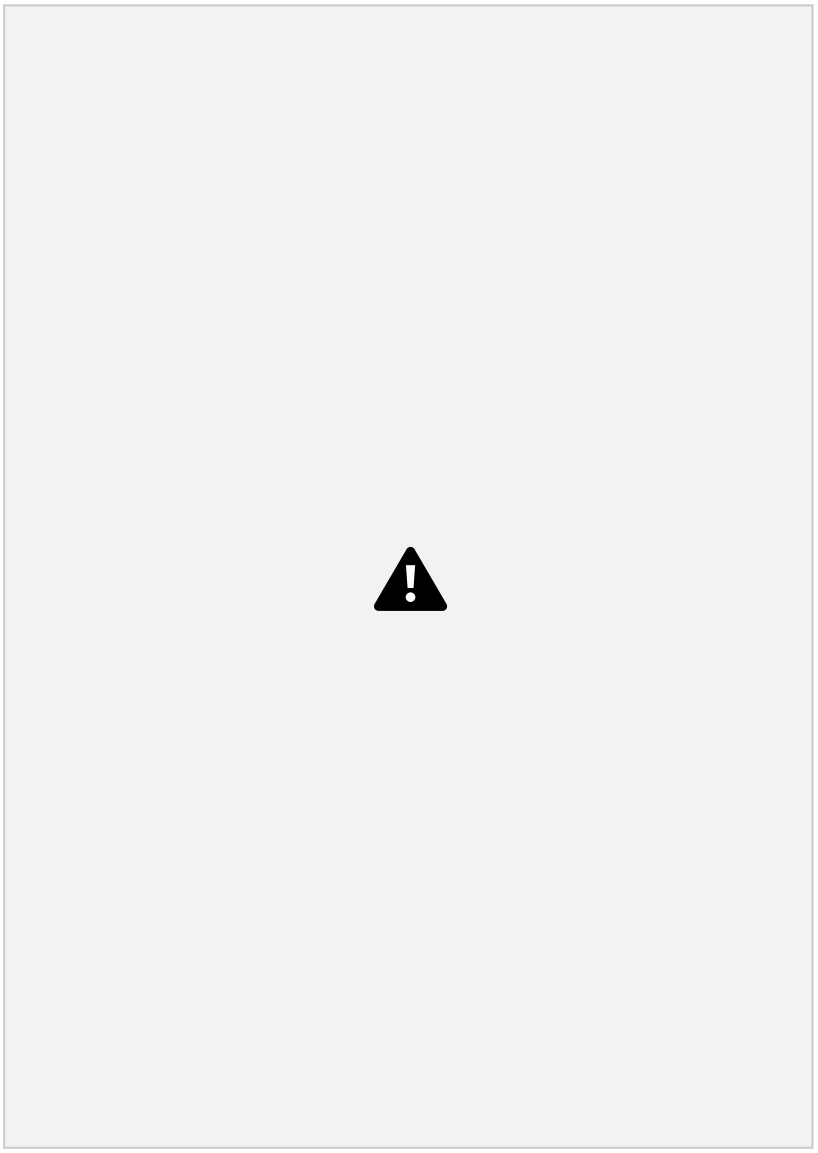
No retorno formulário informe a propriedade cancelar recebendo limpar formulario

Em formulário.js passe o parâmetro cancelar

No botão de cancelar escreva a função de click

Remover produto

Em App.js



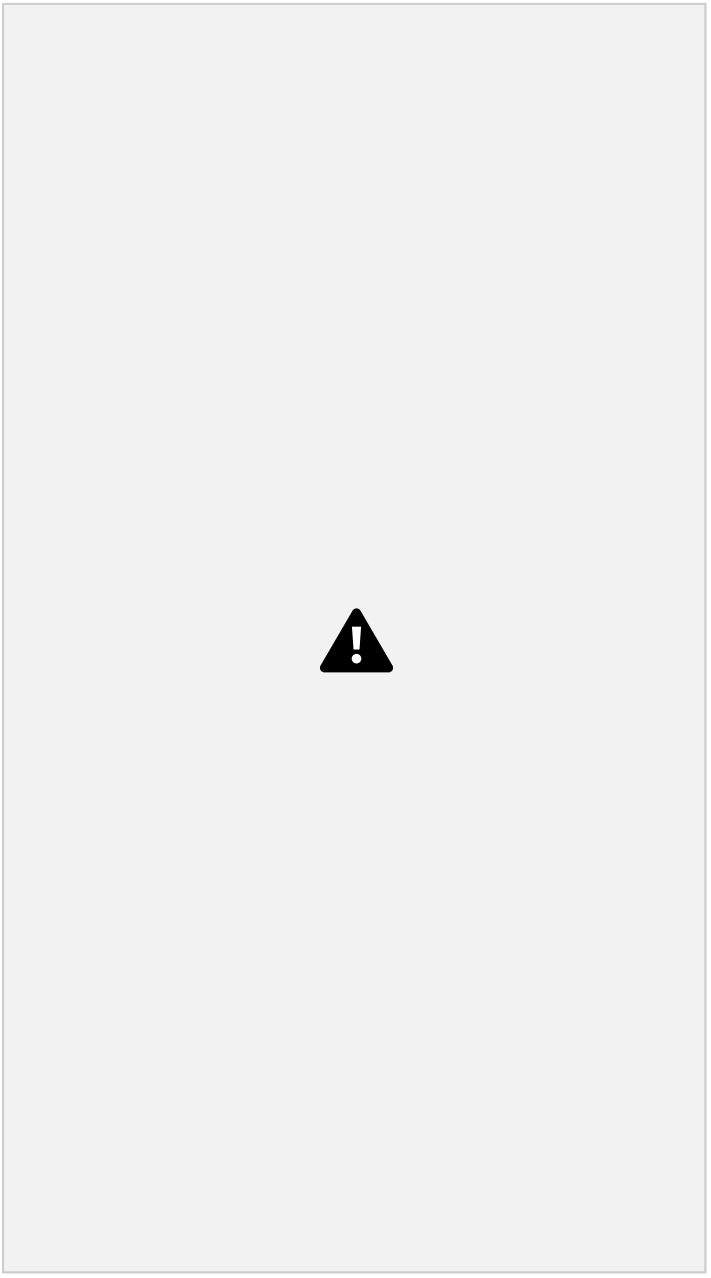
No componente formulário em app.js escreva propriedade

Em formulário.js passe o parâmetro

No botão remover

Alterar produto

Em App.js



No componente formulário

Em Formulario.js

no botão alterar informa a função onclick.

Passa o parâmetro alterar,