

## Documentação: Desenvolvimento de um Sistema de Gestão de Inventário.

**Professor:**Abílio Coelho

**Disciplina:** Front End

**Alunos:** Franciane Santos,Mônica  
Gabrielly, Raísa Marques, Maria Hagata  
e Erika Guimarães.



## Descrição do projeto:

Neste trabalho foi desenvolvido um sistema de gestão de inventário, o sistema permite que os usuários gerenciem o estoque de uma loja ou armazém. As funcionalidades incluem o cadastro de produtos, controle de quantidades, entrada e saída de itens, além da geração de relatórios sobre a movimentação de estoque. Apenas o gerente tem acesso ao histórico de movimentações.

Foram utilizadas as tecnologias no desenvolvimento:

- **Node Js** : Para versatilidade do código e nas instalações de dependências.
- **Docker** : Utilizado para a implementação .
- **PostgreSQL**: Foi nosso gerenciador de banco de dados, para armazenar e organizar os dados do projeto.
- **Express**: Este framework nos ajudou na movimentação do banco de dados, e na simplificação de tarefas.
- **Prisma**: Simplificou o acesso ao banco de dados já que é compatível com Node e TypeScript.
- **CRUD**: São as quatro ações que foram utilizadas no decorrer do projeto. Create (criar), Read (ler), Update (atualizar) e Delete (apagar) que são fundamentais no desenvolvimento de aplicações que interagem com bancos de dados.
- **React JS**: Utilizado na parte Front -End , para o desenvolvimento das telas .

## Banco de dados (Modelo Relacional)

O modelo de relacionamento foi feito em um website online chamado DrawSQL.

### 1. Produtos

Tabela para armazenar produtos.

- id
- nome
- descrição
- preço
- quantidade

- data\_entrada
- data\_saida

Relacionamento:

- Um produto pode ter várias movimentações (1:N com Movimentações).

## 2. Relatório de Movimentações

Tabela para registrar todas as movimentações de entrada e saída de produtos.

- id
- produto\_id (FK) - Relaciona com Produtos
- tipo (entrada e saída)
- quantidade
- data
- usuario\_id (FK) - Relaciona com Gerente

Relacionamento:

- Cada movimentação pertence a um produto (N:1 com Produtos).
- Cada movimentação é registrada por um gerente (N:1 com Gerente).

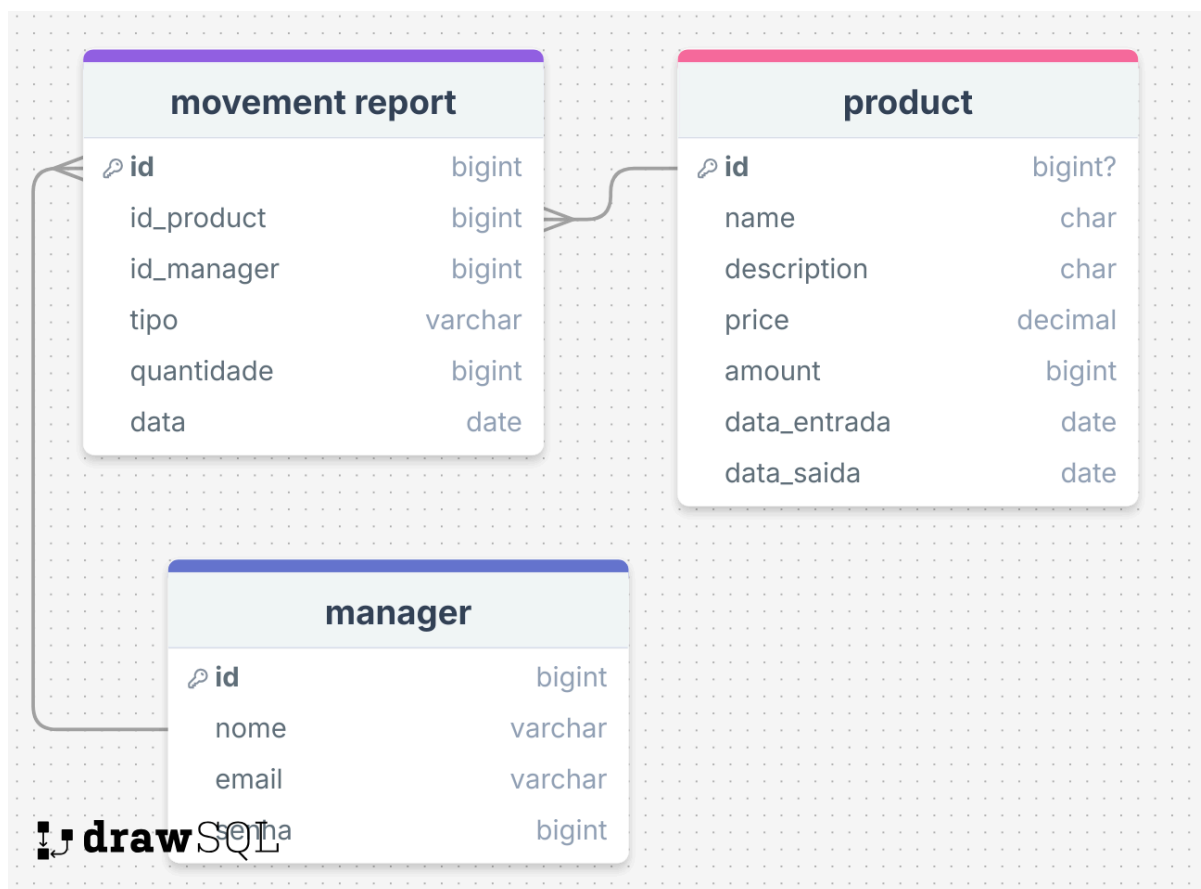
## 3. Usuário

Tabela para armazenar os dados dos gerentes.

- id
- nome
- email
- senha

Relacionamento:

- Um gerente pode estar associado a várias movimentações (1:N com Movimentações).



## Instruções de Instalação e Configuração:

### Back-end:

No Back-end foram usadas as seguintes tecnologias já citadas anteriormente , que são: Node Js , Docker , PostgreSQL , Express, Prisma, Cors e CRUD. Vamos percorrer por cada uma delas .

### Node Js:

No node foi utilizada a versão mais recente **20.15.1 LTS** no momento de criação , para evitar qualquer problema com bugs e compatibilidade. É por ele que vamos instalar todas as dependências .

**npm init -y** (PARA INICIAR PROJETO NODE)

**npm i express** (PARA INSTALAR EXPRESS)

**npm i @types/express -d** (PARA INSTALAR O EXPRESS PARA TYPESCRIPT)



**npm i typescript ts-node @types/node -d** (PARA INSTALAR TYPESCRIPT)

**npm i prisma -d** (PARA INSTALAR O PRISMA)

**npx tsc --init** (PARA INICIAR O TYPESCRIPT E CRIAR O ARQUIVO "TSCONFIG.TS")

**npx prisma init** (PARA INICIAR O PRISMA E CRIAR O ".env" E O "schema.prisma")

**npx prisma migrate dev --name init** ()

## Docker:

Os comandos utilizados foram:

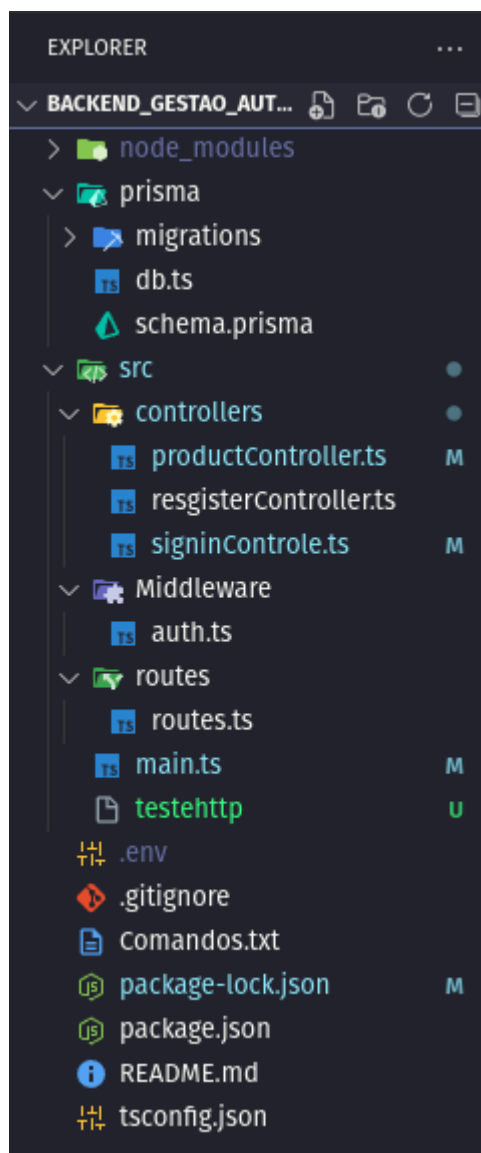
**docker rm postgres** (PARA APAGAR INSTÂNCIA CASO EXISTA)

**docker run --name postgres ip 5432:5432 -e POSTGRES\_PASSWORD=secret -d postgres** (PARA CRIAR A INSTÂNCIA)

**docker exec -it postgres psql -U postgres** (RODA O POSTGRES PSQL PARA LISTAR OS BANCOS DE DADOS EXISTENTES)

**create database product**(CRIA O DATABASE "Product")

## Organização das pastas



## Prisma client



```
1 import { PrismaClient } from "@prisma/client";
2
3 const prismaClientSingleton = () => {
4   return new PrismaClient();
5 };
6
7 declare const globalThis: {
8   prismaGlobal: ReturnType<typeof prismaClientSingleton>;
9 } & typeof global;
10
11 const prisma = globalThis.prismaGlobal ?? prismaClientSingleton();
12
13 export default prisma;
14
15 if (process.env.NODE_ENV !== "production") globalThis.prismaGlobal = prisma;
```

## Schema Prisma

```
5 datasource db {
6   provider = "postgresql"
7   url      = env("DATABASE_URL")
8 }
9
10 model User {
11   id        String   @id @default(uuid())
12   name      String
13   email     String   @unique
14   password  String
15   products  Product[]
16 }
17
18 model Product {
19   id          String      @id @default(uuid())
20   userId      String
21   name        String
22   description  String
23   price       String
24   amount      String
25   createdAt   DateTime    @default(now())
26   updatedAt   DateTime    @updatedAt
27   user        User        @relation(fields: [userId], references: [id])
28   history     ProductHistory[]
29 }
30
31 model ProductHistory {
32   id          String      @id @default(uuid())
33   productId   String
34   action      String
35   name        String
36   description  String
37   price       String
38   amount      String
39   createdAt   DateTime    @default(now())
40   product     Product     @relation(fields: [productId], references: [id])
41 }
```

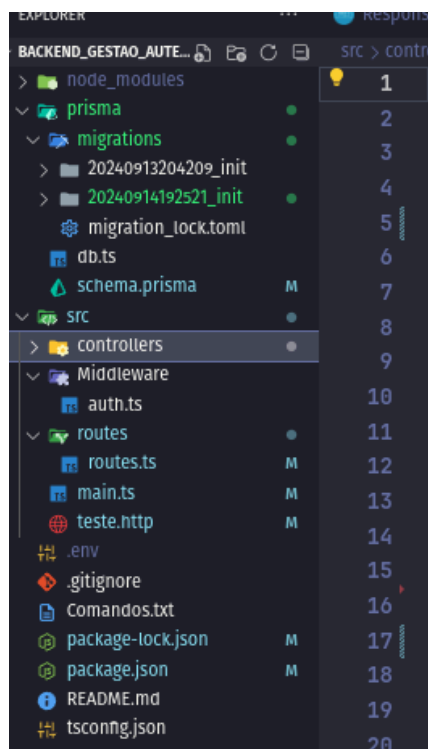


## Routers

```
BACKEND_GESTAO_AUTENTICACAO
├── node_modules
├── prisma
├── migrations
│   └── 20240913204209_init
│       └── migration_lock.toml
├── db.ts
├── schema.prisma
├── src
│   ├── controllers
│   │   ├── productController.ts
│   │   ├── registerController.ts
│   │   └── signinController.ts
│   ├── middleware
│   │   └── auth.ts
│   └── routes
│       └── routes.ts
├── main.ts
├── teste.http
├── .env
├── .gitignore
├── Comandos.txt
├── package-lock.json
├── package.json
├── README.md
└── tsconfig.json
```

```
src > routes > routes.ts > ...
1  import { Router } from "express";
2  import { ProductController } from "../controllers/productController";
3  import { authMiddleware } from "../Middleware/auth";
4  import { registerController } from "../controllers/registerController";
5  import { signinController } from "../controllers/signinController";
6
7  const routes = Router();
8  const productController = new ProductController();
9
10
11 routes.post("/register", registerController);
12 routes.post("/signin", signinController);
13
14
15 routes.post("/product", authMiddleware, productController.createProduct);
16 routes.get("/product", authMiddleware, productController.getAllProducts);
17 routes.get("/product/:id", authMiddleware, productController.getProductById);
18 routes.put("/product/:id", authMiddleware, productController.updateProduct);
19 routes.delete("/product/:id", authMiddleware, productController.deleteProduct);
20 routes.get("/product/:id/history", authMiddleware, productController.getProductHistory);
21
22 routes.get("/history", authMiddleware, productController.getAllProductHistories);
23
24 export { routes };
25
```

## Controllers



**Organização :** Dentro da pasta controllers encontramos três subpastas, **Product controller** , **Register Controllers**, **Signin Controllers**, cada uma com suas respectivas funções.



## Product controller

Função getAllProducts.

```
import { Request, Response } from "express";
import prisma from "../../prisma/db";

export class ProductController {

  public async getAllProducts(request: Request, response:
Response) {
    try {
      const products = await prisma.product.findMany();
      return response.json(products);
    } catch (error) {
      console.error("Error fetching products:", error);
      return response.status(500).json({ error: "Internal
Server Error" });
    }
  }
}
```



**Consulta:** Busca todos os produtos existentes.

**Resposta:** Mostra para o usuário todos os produtos existentes.

## Create Product

```
public async createProduct(request: Request, response:
Response) {
  const { name, description, price, amount, userId } =
request.body;
  try {
    const product = await prisma.product.create({
      data: {
        name,
        description,
        price,
        amount,
        userId,
      },
    });
    await prisma.productHistory.create({
      data: {
        productId: product.id,
        action: 'created',
        name: product.name,
        description: product.description,
        price: product.price,
        amount: product.amount,
      },
    });

    return response.status(201).json(product);
  } catch (error) {
    console.error("Error creating product:", error);
    return response.status(500).json({ error: "Internal
Server Error" });
  }
}
```



**Busca:** Cria novos produtos.

**Resposta:** Adiciona um produto novo aos produtos existentes.

## GetProductById

```
public async getProductById(request: Request, response:
Response) {
  const { id } = request.params;
  try {
    const product = await prisma.product.findUnique({
      where: { id: String(id) },
    });
    if (!product) {
      return response.status(404).json({ error: "Product not
found" });
    }
    return response.json(product);
  } catch (error) {
    console.error("Error fetching product:", error);
    return response.status(500).json({ error: "Internal
Server Error" });
  }
}
```

**Busca:** Busca informações do produto com base no **id** de um produto específico.

**Resposta:** Mostra as informações do produto específico.



## UpdateProduct:

```
public async updateProduct(request: Request, response:
Response) {
  const { id } = request.params;
  const { name, description, price, amount } = request.body;
  try {
    const existingProduct = await prisma.product.findUnique({
      where: { id: String(id) },
    });
    if (!existingProduct) {
      return response.status(404).json({ error: "Product not
found" });
    }
    const updatedProduct = await prisma.product.update({
      where: { id: String(id) },
      data: { name, description, price, amount },
    });
    await prisma.productHistory.create({
      data: {
        productId: updatedProduct.id,
        action: 'updated',
        name: updatedProduct.name,
        description: updatedProduct.description,
        price: updatedProduct.price,
        amount: updatedProduct.amount,
      },
    });
    return response.json(updatedProduct);
  } catch (error) {
    console.error("Error updating product:", error);
    return response.status(500).json({ error: "Internal
Server Error" });
  }
}
```



**Recebe dados:** Modifica os dados de produto específico usando o **id** como parâmetro de busca.

**Resposta:** Se ocorrer tudo bem atualiza os dados de um produto em específico, caso contrário exibe uma mensagem de erro.

### DeleteProduct:

```
public async deleteProduct(request: Request, response: Response) {
  const { id } = request.params;
  try {
    const product = await prisma.product.findUnique({
      where: { id: String(id) },
    });
    if (!product) {
      return response.status(404).json({ error: "Product not found"
});
    }
    await prisma.productHistory.create({
      data: {
        productId: String(id),
        action: "deleted",
        name: product.name,
        description: product.description,
        price: product.price,
        amount: product.amount,
      },
    });
    await prisma.productHistory.deleteMany({
      where: { productId: String(id) },
    });
    await prisma.product.delete({
      where: { id: String(id) },
    });
    return response.json({ message: "Product deleted successfully"
});
  } catch (error) {
    console.error("Error deleting product:", error);
    return response.status(500).json({ error: "Internal Server
Error" });
  }
}
```



**Recebe um ID:** Deleta um produto específico usando o **id** como parâmetro de busca.

**Resposta:** Se ocorrer tudo bem mostra ao usuário uma mensagem informando que o produto foi excluído com sucesso, caso contrário exibe uma mensagem de erro.

## GetAllProductHistories

```
public async getAllProductHistories(request: Request,  
response: Response) {  
  try {  
    const histories = await prisma.productHistory.findMany({  
      orderBy: { createdAt: 'desc' },  
    });  
    return response.json(histories);  
  } catch (error) {  
    console.error("Error fetching all product histories:",  
error);  
    return response.status(500).json({ error: "Internal  
Server Error" });  
  }  
}
```

**Busca:** Mostra o histórico de vários produtos.

**Resposta:** Exibe o histórico de vários produtos.





INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA  
MARANHÃO  
Campus Timon

## Instituto Federal de Educação, Ciência e Tecnologia - IFMA

### Curso Tecnológico em Sistemas para Internet

## Register Controllers



```
import { Request, Response } from "express";

import prisma from "../../prisma/db";
import bcrypt from "bcryptjs"

export async function registerController(request: Request,
response: Response) {

  const {name, email, password} = request.body

  const userExist = await prisma.user.findFirst({
    where: {
      email,
    },
  })
  if(userExist){
    return response.json({erro: "Usuário já existe"})
  }
  const hashedPassword = await bcrypt.hash(password, 10)
  const user = await prisma.user.create({
    data: {
      name: name,
      email: email,
      password: hashedPassword
    }
  })
  return response.json(user)
}
```

**Busca:** Controla o registro dos usuários no sistema.

**Resposta:** Registra usuários no sistema.

## Signin Controllers



```
import { Request, Response } from "express";
import prisma from "../../prisma/db";
import bcrypt from "bcryptjs";
import jwt from "jsonwebtoken";

export const SECRET_KEY = "Do&A0:P50)wNg|Qb<l[Y9L]>3ZkM3";
const TOKEN_EXPIRATION = "24h";

export async function signinController(request: Request, response:
Response) {
  const { email, password } = request.body;
  if (!email || !password) {
    return response.status(400).json({ error: "Email e senha são
obrigatórios" });
  }
  try {
    const userExist = await prisma.user.findFirst({
      where: {
        email,
      },
    });
    if (!userExist) {
      return response.status(401).json({ error: "Credenciais
inválidas" });
    }
    const isValidPassword = await bcrypt.compare(password,
userExist.password);
    if (!isValidPassword) {
      return response.status(401).json({ error: "Credenciais
inválidas" });
    }
    const token = jwt.sign({
      id: userExist.id,
      name: userExist.name,
```



```
email: userExist.email,  
  }, SECRET_KEY, { expiresIn: TOKEN_EXPIRATION });  
  
  return response.json({ token });  
} catch (error) {  
  console.error("Erro ao fazer login:", error);  
  return response.status(500).json({ error: "Erro interno do  
servidor" });  
}  
}
```

**Busca:** Controla o login dos usuários no sistema.

**Resposta:** Loga os usuários no sistema.

## Middleware auth.

```
import { Request, Response, NextFunction } from "express";
import jwt, { JwtPayload } from 'jsonwebtoken'
import { SECRET_KEY } from "../controllers/signinControle"

type RequestWithUser = Request & {
  user?: string | JwtPayload
}

export async function authMiddleware(request: RequestWithUser, response: Response, next: NextFunction) {
  const authHeader = request.headers['authorization'];
  if (!authHeader) {
    return response.status(401).json({ erro: "Token não fornecido" });
  }

  const token = authHeader.split(' ')[1];
  if (!token) {
    return response.status(401).json({ erro: "Token não fornecido" });
  }

  try {
    const decoded = jwt.verify(token, SECRET_KEY);
    request.user = decoded;
    next();
  } catch (error) {
    return response.status(401).json({ erro: "Token Inválido" });
  }
}
```

**Busca:** Verifica se a solicitação contém um token JWT válido no cabeçalho. Se o token for válido, as informações decodificadas do token são adicionadas à solicitação e o middleware permite que a solicitação continue para a próxima etapa.

**Resposta:** Verifica o token, decodifica-o, adiciona o token à solicitação e permite que passe para a próxima etapa.

## Front-End:

**Tecnologias usadas:** Node Js, vite, lucid icons, react router dom

## Terminal:

**npm install:** ( INSTALAÇÃO DAS DEPENDÊNCIAS. )

**npm i :** ( INSTALA AS DEPENDÊNCIAS DO PROJETO )

**npm i react-router-dom :** ( INSTALA A BIBLIOTECA “react-router-dom”, QUE É USADA PARA GERENCIAR ROTAS EM APLICATIVOS REACT. ELA PERMITE QUE VOCÊ CRIE ROTAS E NAVEGUE ENTRE DIFERENTES PÁGINAS )

**npm run dev :** ( INICIA O SERVIDOR )

## Login

```
import { useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faUserCircle } from '@fortawesome/free-solid-svg-icons';

export default function Login() {
  const [email, setEmail] = useState<string>('');
  const [password, setPassword] = useState<string>('');
  const [error, setError] = useState<string>('');
  const [success, setSuccess] = useState<string>('');
  const [hoveredButton, setHoveredButton] = useState<string | null>(null);
  const navigate = useNavigate();

  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    setError('');
    setSuccess('');

    try {
      const response = await axios.post('http://10.24.31.147:3333/signin', {
        email,
        password,
      });

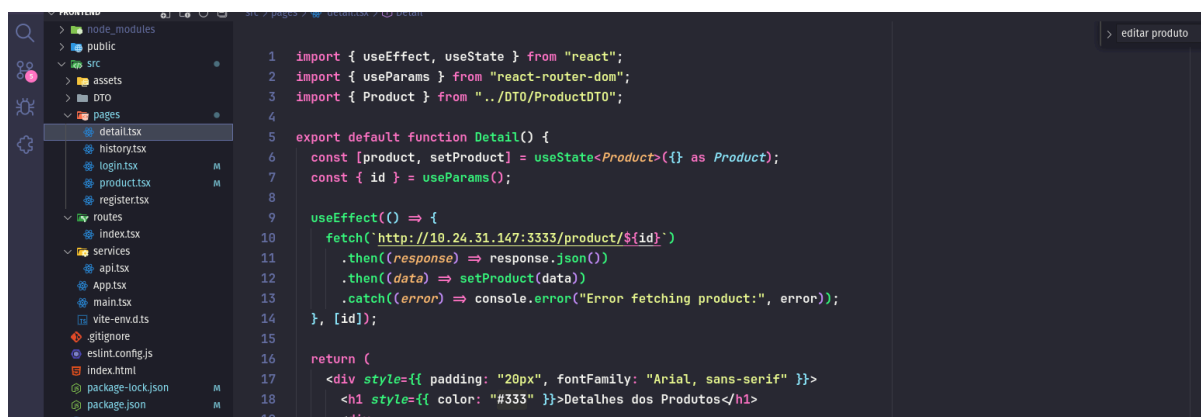
      localStorage.setItem('token', response.data.token);

      setSuccess('Login bem-sucedido!');
      navigate('/product');
    } catch (err) {
      console.error('Erro ao fazer login:', err);
      setError('Erro ao fazer login. Verifique suas credenciais.');
```

- Estados: Utiliza useState para gerenciar o e-mail, a senha, mensagens de erro e sucesso.

- Envio do Formulário: Quando o formulário é enviado, faz uma requisição POST para um servidor para verificar as credenciais.
- Armazenamento e Navegação: Se o login for bem-sucedido, armazena um token no `localStorage` e redireciona o usuário para a página de produtos. Se falhar, exibe uma mensagem de erro.
- Redirecionamento: Também permite redirecionar para a página de registro com uma função

## Detail do Product



```
1 import { useEffect, useState } from "react";
2 import { useParams } from "react-router-dom";
3 import { Product } from "../DTO/ProductDTO";
4
5 export default function Detail() {
6   const [product, setProduct] = useState<Product>({} as Product);
7   const { id } = useParams();
8
9   useEffect(() => {
10     fetch(`http://10.24.31.147:3333/product/${id}`)
11       .then((response) => response.json())
12       .then((data) => setProduct(data))
13       .catch((error) => console.error("Error fetching product:", error));
14   }, [id]);
15
16   return (
17     <div style={{ padding: "20px", fontFamily: "Arial, sans-serif" }}>
18       <h1 style={{ color: "#333" }}>Detalhes dos Produtos</h1>
19       <div>
```

Ele usa `useState` para armazenar o estado do produto e `useParams` para obter o ID da URL. Com `useEffect`, faz uma requisição à API para buscar os detalhes do produto com base no ID. Os dados recebidos (nome, descrição, preço e quantidade) são renderizados com estilização simples. Melhorias possíveis incluem validação de estado vazio, feedback de carregamento e tratamento de erros visíveis ao usuário.

## Register

```
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';
import { api } from '../services/api';
import { faUserTie } from '@fortawesome/free-solid-svg-icons';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';

export default function Register() {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [success, setSuccess] = useState('');

  const navigate = useNavigate();

  const handleSubmit = async (event: React.FormEvent<HTMLFormElement>) => {
    event.preventDefault();
    setError('');
    setSuccess('');

    axios.defaults.headers.post['Content-Type'] = 'application/json;charset=utf-8';
    axios.defaults.headers.post['Access-Control-Allow-Origin'] = '*';

    try {
      const response = await api.post('/register', {
        name,
        email,
        password,
      });

      if (response.data.error) {
        setError(response.data.error);
      } else {
        setSuccess('Usuário registrado com sucesso!');
        setName('');
        setEmail('');
        setPassword('');

        navigate('/');
      }
    } catch {
      setError('Erro ao registrar usuário.');
    }
  };
}
```

O componente Register é responsável por registrar um novo usuário (gestor). Ele utiliza useState para controlar os campos de entrada (nome, email, senha) e as mensagens de erro ou sucesso. Ao submeter o formulário, faz uma requisição POST para a API com Axios e, em caso de sucesso, limpa o formulário, exibe uma mensagem de sucesso e redireciona o usuário para a página inicial. Estilos inline são usados para personalizar o layout e a interface visual.



## Produtos

### Home

```
import { useState, useEffect } from "react";
import { Link } from "react-router-dom";
type Product = {
  id: string;
  name: string;
  description: string;
  price: string;
  amount: string;
};
export default function Home() {
  const [products, setProducts] = useState<Product[]>([]);
  const [newProduct, setNewProduct] = useState({
    name: "",
    description: "",
    price: "",
    amount: "",
  });
  const [editProduct, setEditProduct] = useState<Product | null>(null);
  const [showDeleteConfirmModal, setShowDeleteConfirmModal] =
    useState(false);
  const [deleteProductId, setDeleteProductId] = useState<string |
    null>(null);
  const [visible, setVisible] = useState(false);
  const [searchTerm, setSearchTerm] = useState("");

  useEffect(() => {
    fetch("http://10.24.31.147:3333/product")
      .then((response) => {
        if (!response.ok) {
          throw new Error("Failed to fetch products");
        }
        return response.json();
      })
      .then((data) => setProducts(data))
  });
}
```



```
        .catch((error) => console.error("Error fetching products:",  
error));  
    }, []);  
  
    const formatPrice = (price: string) => {  
        const parsedPrice = parseFloat(price).toFixed(2);  
        return `R$ ${parsedPrice.replace(".", ",")}`;  
    };  
  
    const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) =>  
{  
        const { name, value } = e.target;  
        setNewProduct((prevProduct) => ({  
            ...prevProduct,  
            [name]: name === "amount" ? String(value) : value,  
        }));  
    };  
};
```

A função `handleInputChange` é usada para atualizar o estado do novo produto quando o usuário faz alterações nos campos de entrada

## AddProduct

```
const handleAddProduct = () => {  
    if (!newProduct.name || !newProduct.description ||  
!newProduct.price) {  
        alert("Por favor, preencha todos os campos antes de adicionar um  
produto.");  
        return;  
    }  
  
    fetch("http://10.24.31.147:3333/product", {  
        method: "POST",  
        headers: {  
            "Content-Type": "application/json",  
        },  
        body: JSON.stringify(newProduct),  
    })
```



```
    })  
  
    .then((response) => {  
      if (!response.ok) {  
        throw new Error("Failed to add product");  
      }  
      return response.json();  
    })  
    .then((data) => {  
      setProducts((prevProducts) => [...prevProducts, data]);  
      setNewProduct({  
        name: "",  
        description: "",  
        price: "",  
        amount: "",  
      });  
    })  
    .catch((error) => console.error("Error adding product:", error));  
  };  
};
```

A função AddProduct adiciona um produto à lista exibida no aplicativo, validando os dados, enviando-os para o servidor e atualizando a interface do usuário conforme necessário.

## DeleteProduct

```
const handleDeleteProduct = (id: string) => {  
  setDeleteProductId(id);  
  setShowDeleteConfirmModal(true);  
};  
  
const confirmDeleteProduct = () => {  
  if (deleteProductId !== null) {  
    fetch(`http://10.24.31.147:3333/product/${deleteProductId}`, {  
      method: "DELETE",  
    })  
    .then((response) => {  
      if (!response.ok) {  
        throw new Error("Failed to delete product");  
      }  
    })  
  }  
};
```



```
setProducts((prevProducts) =>
  prevProducts.filter((product) => product.id !==
deleteProductId)
);
setShowDeleteConfirmModal(false);

setDeleteProductId(null);
})
.catch((error) => console.error("Error deleting product:",
error));
};
};
```

Essas funções juntas permitem que o usuário selecione um produto para exclusão, confirme a exclusão, e, se confirmado, remova o produto da lista e atualize a interface do usuário.

## EditProduct

```
const handleEditProduct = (product: Product) => {
  setEditProduct(product);
  setVisible(true);
};
```

A função `handleEditProduct` configura o produto a ser editado e mostra a interface de edição, permitindo ao usuário modificar as informações do produto selecionado.

## UpdateProduct

```
const handleUpdateProduct = () => {
  if (editProduct) {
    fetch(`http://10.24.31.147:3333/product/${editProduct.id}`, {
      method: "PUT",
```



```
headers: {
  "Content-Type": "application/json",
},
body: JSON.stringify(editProduct),
})

.then((response) => {
  if (!response.ok) {
    throw new Error("Failed to update product");
  }
  return response.json();
})

.then((data) => {
  setProducts((prevProducts) =>
    prevProducts.map((product) =>
      product.id === data.id ? data : product
    )
  );
  setVisible(false);
})

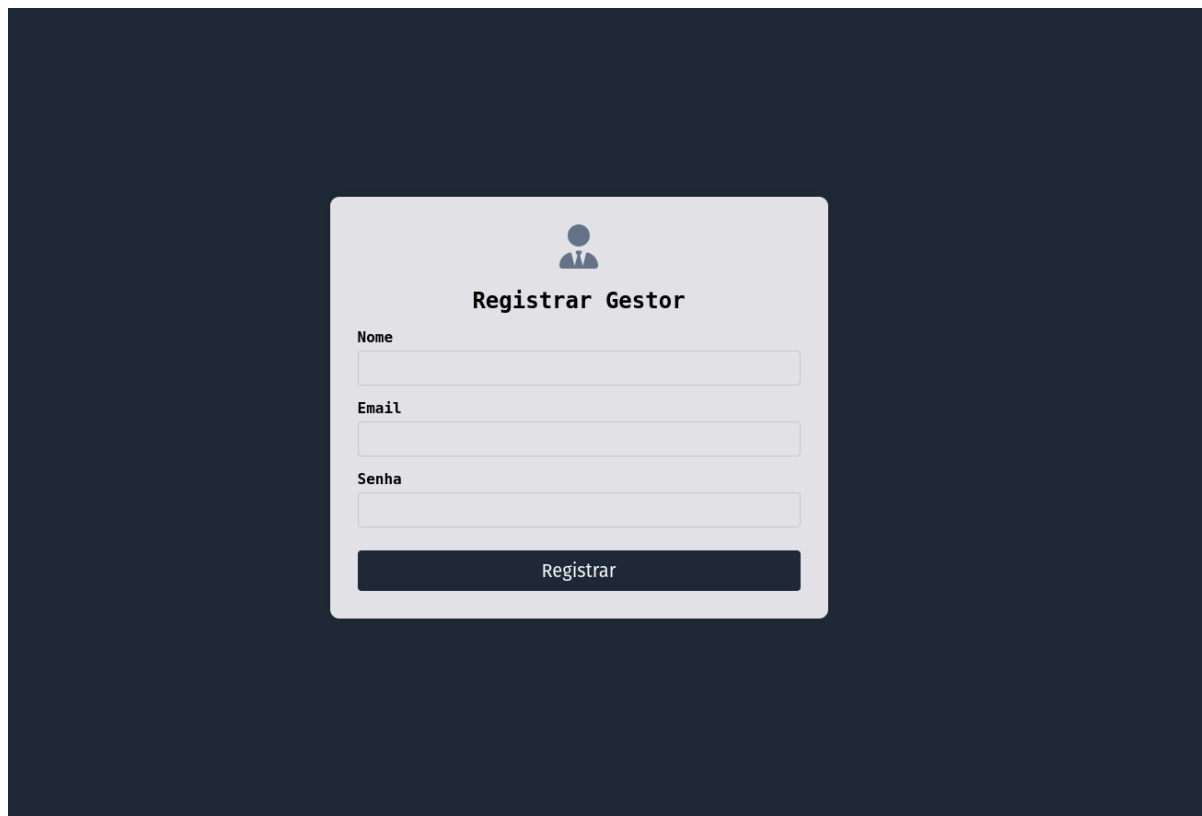
.catch((error) => console.error("Error updating product:", error));
};

const filteredProducts = products.filter((product) =>
  product.name.toLowerCase().includes(searchTerm.toLowerCase())
);
```


- `handleUpdateProduct`: Atualiza um produto na API e na interface do usuário, e fecha a interface de edição se a atualização for bem-sucedida.
- `filteredProducts`: Filtra os produtos com base no termo de busca fornecido, permitindo que o usuário encontre produtos específicos.

TELAS:

Tela de registrar:



The screenshot shows a web form titled "Registrar Gestor" centered on a dark blue background. The form is a light gray box with rounded corners. At the top of the form is a user icon. Below the icon is the title "Registrar Gestor". The form contains three input fields labeled "Nome", "Email", and "Senha". At the bottom of the form is a dark blue button with the text "Registrar".



**Registrar Gestor**

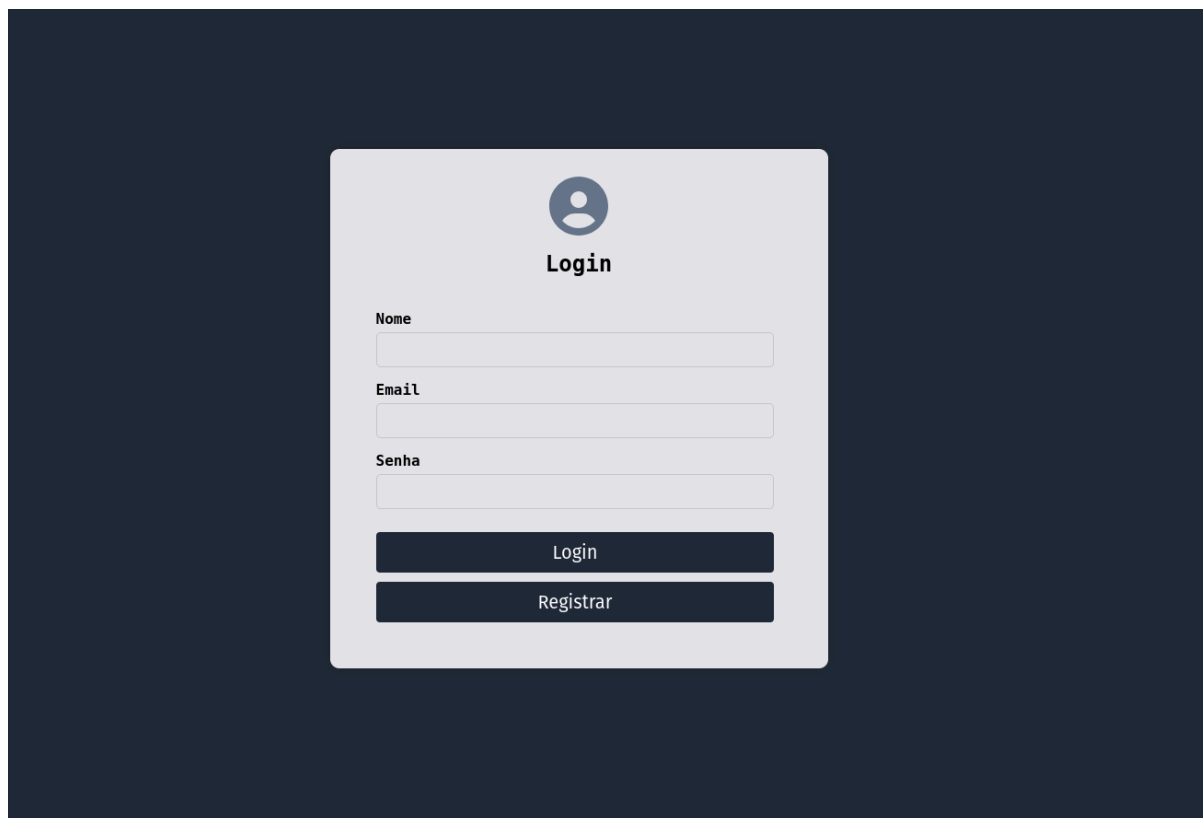
Nome

Email


Senha

**Registrar**

### Tela de Login:



The login screen features a dark blue background. In the center is a light gray rounded rectangle containing a user icon (a circle with a person silhouette) and the word "Login" in bold. Below this are three input fields labeled "Nome", "Email", and "Senha". At the bottom of the form are two dark blue buttons with white text: "Login" and "Registrar".

**Login**

Nome

Email

Senha

Login

Registrar

Adicionar novos produtos:

### Gestão de Inventário

Adicionar Produto

Ver Histórico

tapioca

R\$ 5,00
















































Quantidade: 2

Editar

Excluir



## Tela de detalhes do produto:



Detalhes dos Produtos

tapioca

Descrição:

Preço: \$5

Quantidade: 2

## Tela editar produto:

Gestão de Inventário

Pesquisar por nome

Nome

Descrição

Preço

Qtd

Editar Produto

tapioca

Descrição

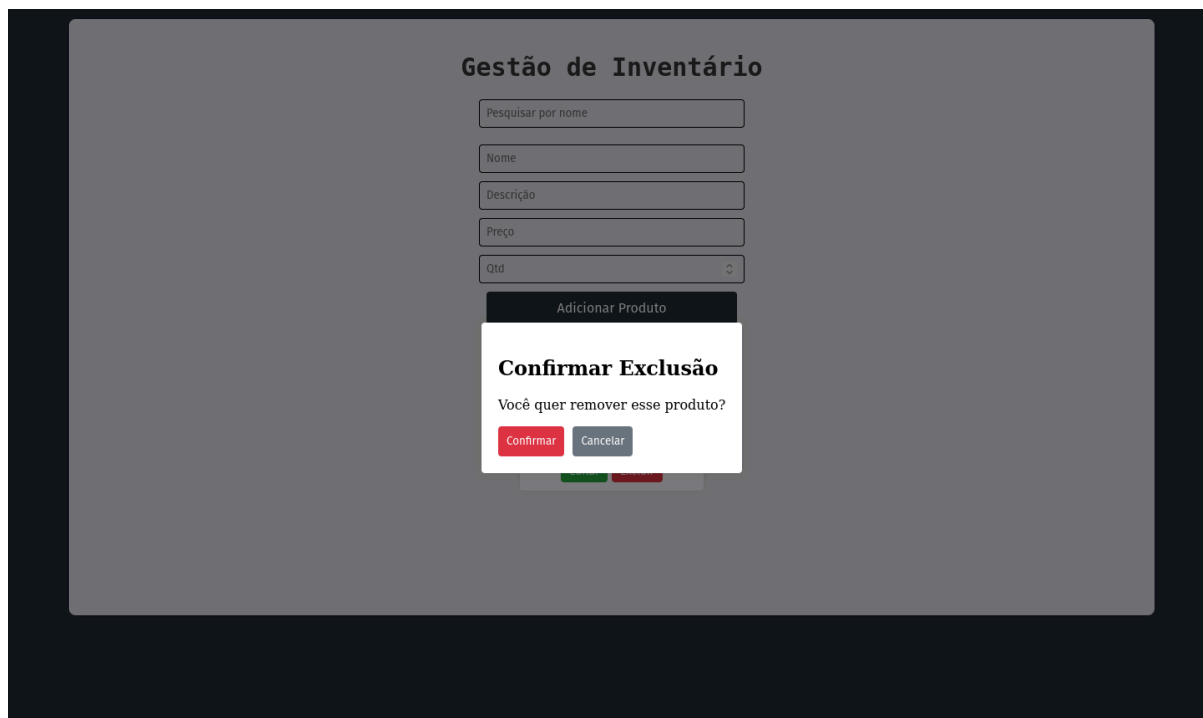
5

2

Atualizar Produto



Cancelar

Tela deletar produto:



The screenshot displays a web application interface for inventory management. The main heading is "Gestão de Inventário". Below it, there is a search bar labeled "Pesquisar por nome". A list of product fields is visible: "Nome", "Descrição", "Preço", and "Qtd" (Quantity), each with a corresponding input field. A button labeled "Adicionar Produto" is positioned below the fields. A modal dialog box is open in the center, titled "Confirmar Exclusão", with the text "Você quer remover esse produto?". It contains two buttons: "Confirmar" (red) and "Cancelar" (gray).

## Tela do Histórico:

  localhost:5173/history	
<h3>Histórico</h3>	
<p><b>Product ID:</b> 400c6995-e580-4ae2-a8d4-e45af6d9b963 <b>Name:</b> tapioca <b>Status:</b> deleted <b>Price:</b> \$5.00 <b>Amount:</b> 2 <b>Date:</b> 16/09/2024 17:11:46</p>	
<hr/>	
<p><b>Product ID:</b> 400c6995-e580-4ae2-a8d4-e45af6d9b963 <b>Name:</b> tapioca <b>Status:</b> deleted <b>Price:</b> \$5.00 <b>Amount:</b> 2 <b>Date:</b> 16/09/2024 17:11:03</p>	
<hr/>	