



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
MARANHÃO
Campus Timon

Instituto Federal de Educação, Ciência e Tecnologia - IFMA
Curso Tecnológico em Sistemas para Internet

Documentação da Atividade Colaborativa: Desenvolvimento de um Sistema Completo de Gerenciamento de Contato

Professor : Abílio Coelho

Disciplina: Back End e Front End

Alunos: Franciane Santos,
Isaac Costa ,Lucas Moura,
Kalline Ferreira ,Raisa Marques ,
Maria Hagata .

Descrição do projeto:

Neste trabalho foi desenvolvido um sistema de gerenciamento de Contato , onde desenvolvemos um sistema completo com tecnologias Front-End e Back-End . Os *requisitos funcionais* deste trabalho foram : ações de cadastrar , editar e remover os contatos. Os *requisitos não funcionais* foram: e buscar pelo nome.

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 .

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 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 contact (CRIA O DATABASE "Contact")
```

Crud:

Get Geral:



```
app.get("/contact", async (request: Request, response: Response) => {
  const contact = await prisma.contact.findMany();
  return response.json(contact);
});
```

Consulta: Obtém todos os contatos armazenados no banco de dados.

Resposta: Envia esses contatos como uma resposta JSON para o cliente que fez a solicitação.

Get ID:

```
app.get("/contact/:id", async (request: Request, response: Response) =>{
  const {id} = request.params
  const contact = await prisma.contact.findUnique({
    where:{
      id:Number(id)
    }
  })
  return response.json(contact)
})
```

Recebe ID: Recebe o Id do contato como parâmetro URL

Busca: Faz a busca pelo Id fornecido no banco de dados

Resposta:Envia o contato encontrado como resposta JSON para o cliente que fez a solicitação.

Post:

```
app.post("/contact", async (request: Request, response: Response) =>
{
  const { name, lastname, phone, adress, email } = request.body;
  const contact = await prisma.contact.create({
    data: {
      name,
      lastname,
      phone,
      adress,
      email
    },
  });
  return response.json(contact);
});
```

Recebe dados: Extrai os dados do contato (nome, sobrenome, telefone, endereço, e email) do corpo da solicitação.

Resposta: Envia o novo contato criado como uma resposta JSON para o cliente que fez a solicitação.

Get Busca:

```
//GET BUSCA
app.get("/contact/:name", async (request: Request, response:
Response) => {
  const { name } = request.params;

  try {
    const contact = await prisma.contact.findFirst({
      where: {
        name: {
          equals: name,
          mode: 'insensitive'
        },
      },
    });

    if (!contact) {
      return response.status(404).json({ error: "Contact not found"
});
    }

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

Recebe um nome: Recebe um nome de contato como parâmetro de URL.

Busca: Procura o primeiro contato no banco de dados cujo nome corresponde ao parâmetro fornecido, de maneira insensível a maiúsculas e minúsculas.

Resposta: Se encontrar o contato, retorna-o em formato JSON. Se não encontrar, retorna uma resposta 404 com uma mensagem de erro. Em caso de erro durante a busca, retorna uma resposta 500 com uma mensagem de erro.

Put para editar:

```
//PUT PARA EDITAR
app.put("/contact/:id", async (request: Request, response: Response) => {
  const { id } = request.params;
  const { name, lastname, adress, email, phone } = request.body;
  const contact = await prisma.contact.update({
    data: {
      name,
      lastname,
      adress,
      email,
      phone
    },
    where: {
      id: Number(id),
    },
  });
  return response.json(contact);
});
```

Recebe um ID: Recebe o ID do contato como parâmetro de URL.

Recebe dados atualizados: Recebe os dados atualizados do contato no corpo da solicitação.

Atualização: Atualiza o registro do contato no banco de dados com o ID correspondente, utilizando os novos dados fornecidos.

Resposta: Envia o contato atualizado como uma resposta JSON para o cliente que fez a solicitação.

Delete:

```
//DELETE
app.delete("/contact/:id", async (request: Request, response:
Response) => {
  const { id } = request.params;
  try {
    await prisma.contact.delete({
      where: {
        id: Number(id),
      },
    });
    response.send("Contact deleted");
  } catch (erro) {
    response.send(erro);
  }
});
```

Deleta: Deleta o registro do contato no banco de dados que corresponde ao ID fornecido.

Resposta: Envia uma resposta indicando que o contato foi deletado com sucesso.

Erro: Se ocorrer um erro durante a tentativa de deletar o contato, envia a mensagem de erro na resposta.

Front-End:

Node Js

Terminal:

npm create vite@latest Contact : (CRIA UM NOVO PROJETO VITE USANDO A VERSÃO MAIS RECENTE DO VITE. ELE PERMITE QUE VOCÊ ESCOLHA UM TEMPLATE PARA O PROJETO.)

cd Contact : (MUDA O DIRETÓRIO ATUAL PARA O DIRETÓRIO CHAMADO "Contact")

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

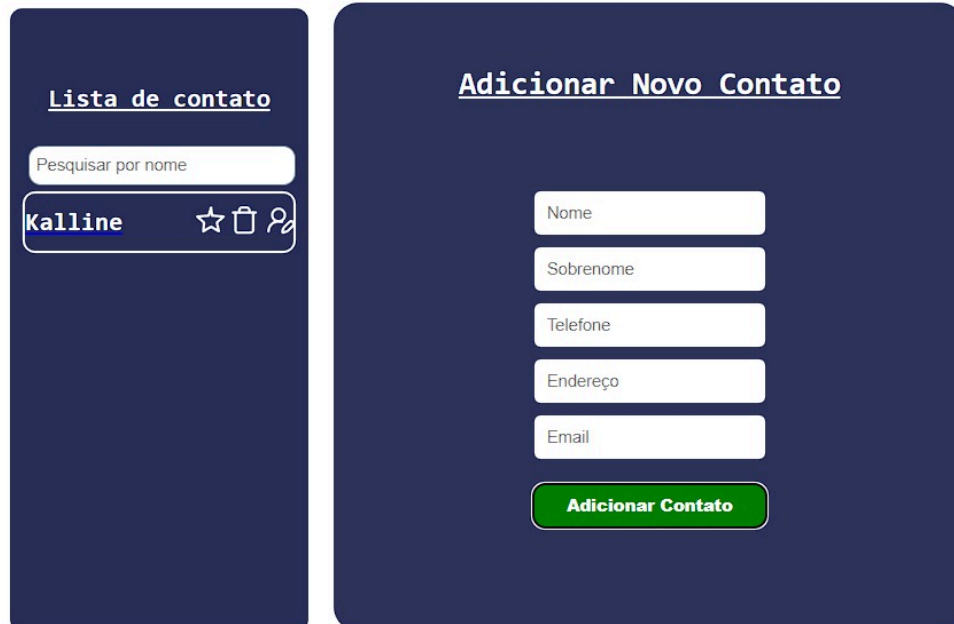
npm i react-router-dom : (*INSTALA A BIBLIOTECA “react-router-dom”, QUE É USADA PARA GERENCIAR ROTAS EM APLICATIVOS REACT. ELA PERMITE QUE VOCÊ CRIE ROTAS E NAVEGUE ENTRE DIFERENTES PÁGINAS EM UM APLICATIVO REACT.*)

npm run dev : (*INICIA O SERVIDOR E AO RODAR NO TERMINAL IRÁ APARECER A URL DA PÁGINA A SER DESENVOLVIDA*)

TELAS:

Adicionar novos contatos:

No lado direito desta tela é possível cadastrar um novo usuário para contato, com os campos de Nome , sobrenome , telefone , endereço e email.



Lista de contato

Pesquisar por nome

Kalline ☆ 🗑️ 👤

Adicionar Novo Contato

Nome

Sobrenome

Telefone

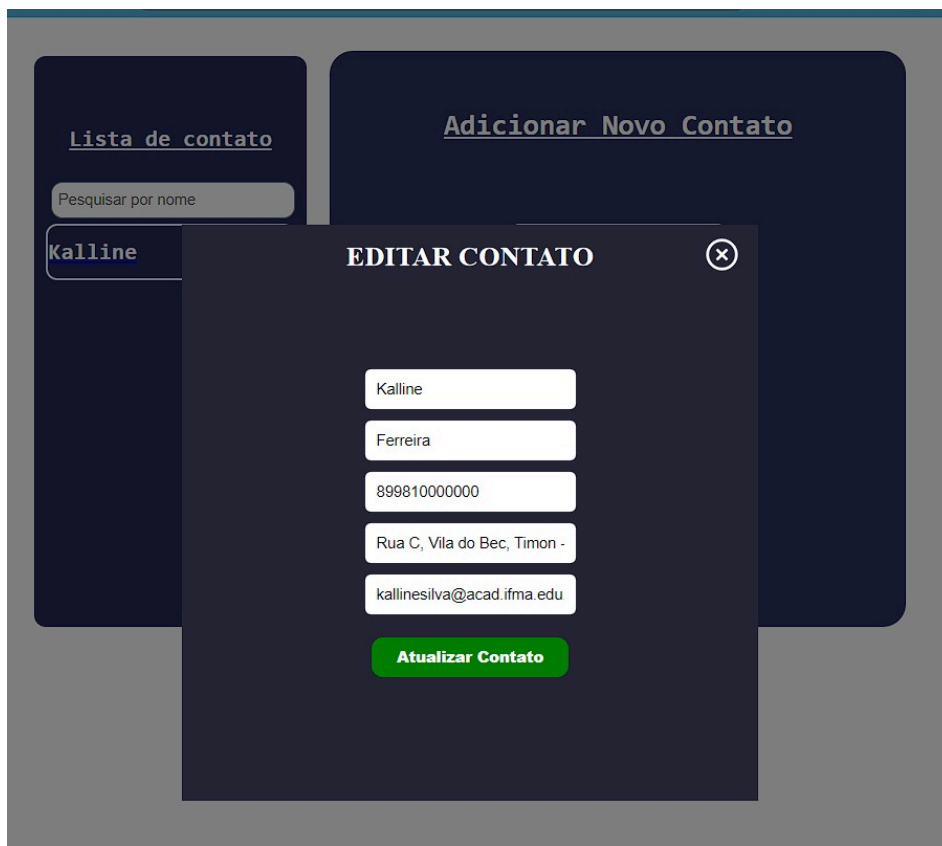
Endereço

Email

Adicionar Contato

Ao lado esquerdo temos a lista de contatos , onde será mostrado todos os novos contatos cadastrados.Nele podemos observar as seguintes ações como : favoritar , deletar, buscar e editar o cadastro.

Tela editar contato:



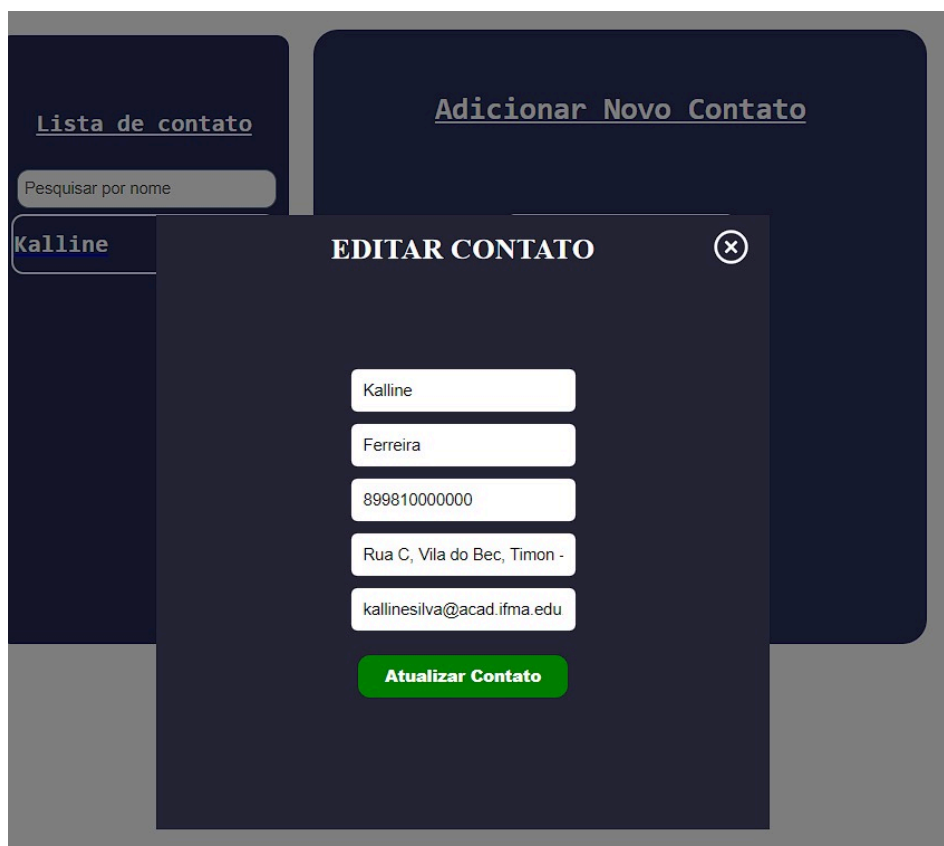
The screenshot shows a web application interface with a dark blue background. On the left, there is a sidebar with the title 'Lista de contato' and a search bar labeled 'Pesquisar por nome'. Below the search bar, the name 'Kalline' is listed. On the right, there is a main area with the title 'Adicionar Novo Contato'. Overlaid on this is a modal window titled 'EDITAR CONTATO' with a close button (X) in the top right corner. The modal contains several input fields with the following text: 'Kalline', 'Ferreira', '899810000000', 'Rua C, Vila do Bec, Timon -', and 'kallinesilva@acad.ifma.edu'. At the bottom of the modal is a green button labeled 'Atualizar Contato'.

Tela deletar contato:



The screenshot shows the same web application interface as the previous one. In the sidebar, the name 'Kalline' is now followed by three icons: a star, a trash can, and a person. The main area still shows the 'Adicionar Novo Contato' form. Overlaid on the main area is a modal window with the text 'Tem certeza que deseja deletar?' (Are you sure you want to delete?). At the bottom of the modal are two buttons: a red button labeled 'Sim' (Yes) and a grey button labeled 'Não' (No).

Tela editar contato:



Favoritar:

Para termos a ação de favoritar foi colocado no Front end o seguinte trecho de código.

```
const toggleFavoriteContact = (id: number) => {  
  setFavoriteContacts((prevFavorites) =>  
    prevFavorites.includes(id)  
      ? prevFavorites.filter((favoriteId) => favoriteId !== id)  
      : [...prevFavorites, id]  
  );  
};
```

Ele não está conectado ao Back end, contudo sua funcionalidade ainda não está interligada, fazendo parte apenas visualmente.

Código

```
1 import React, { useState, useEffect } from "react";
2 import { Link } from "react-router-dom";
3 import { Contact } from "../DTO/ContactDTO";
4
```

Na primeira parte do código temos as importações necessárias para utilizar o React, seus hooks (*useState* e *useEffect*), o componente Link do React Router e o objeto Contact definido no arquivo *ContactDTO.js*

```
1 export default function Home() {
2   const [contacts, setContacts] = useState<Contact[]>([]);
3   const [newContact, setNewContact] = useState({
4     name: "",
5     lastname: "",
6     phone: "",
7     adress: "",
8     email: "",
9   });
}
```

Esse trecho de código define um componente funcional Home em React que mantém dois estados locais:

contacts: Uma lista de contatos, inicialmente vazia.

newContact: Um objeto que armazena informações de um novo contato que está sendo adicionado, inicialmente com campos vazios.

Esses estados são cruciais para gerenciar e exibir informações dinâmicas dentro do componente Home

```
1  const [editContact, setEditContact] = useState({
2    id: 0,
3    name: "",
4    lastname: "",
5    phone: "",
6    adress: "",
7    email: "",
8  });
```

Nesta parte, cria um estado local *editContact* em um componente React funcional. Este estado é utilizado para armazenar os dados de um contato que está sendo editado, permitindo que os campos sejam preenchidos com os dados do contato atualmente em edição.

```
1  const fetchContacts = () => {
2    fetch("http://localhost:3333/contact")
3      .then((response) => {
4        if (!response.ok) {
5          throw new Error("Failed to fetch contacts");
6        }
7        return response.json();
8      })
9      .then((data) => setContacts(data))
10     .catch((error) => console.error("Error fetching contacts:", error));
11  };
```

Aqui, faz uma requisição para obter uma lista de contatos de um servidor (local ou remoto), trata a resposta para verificar se foi bem sucedida, e então atualiza o estado do componente com os dados recebidos.

Estilização

```
1  return (  
2    <div  
3      style={{  
4        display: "flex",  
5        flexDirection: "column",  
6        justifyContent: "center",  
7        alignItems: "center",  
8        padding: 20,  
9      }}  
10   >  
11     <div style={{ display: "flex", marginBottom: 30 }}>  
12       <div  
13         style={{  
14           marginRight: 20,  
15           background: "#282e57",  
16           marginTop: 5,  
17           borderRadius: 10,  
18         }}  
19       >
```

Neste trecho de código React renderiza um componente que exibe uma lista de contatos, oferece funcionalidades de pesquisa, marcação de favoritos, edição e exclusão de contatos. Vamos destacar as principais funcionalidades e elementos presentes no código. Este componente oferece uma interface interativa para gerenciar uma lista de contatos, com funcionalidades completas de CRUD (*Create, Read, Update, Delete*) e interações visuais para melhorar a experiência do usuário.