Programação WEB III

CRUD completo com Angular 18 com MySQL

Prof. Dr. Danilo Barbosa

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

Conteúdo

- Tecnologias do projeto
- Definição do projeto
- Arquitetura do projeto
- Configuração do Backend com Node.js e MySQL
- Configuração do Frontend com Angular
- Configuração das rotas do Angular
- Testando a aplicação
- Referências

Tecnologias do projeto

- Frontend: Angular 18
- Backend: Node.js com Express
- Banco de Dados: MySQL
- Outras ferramentas:

CORS (para permitir requisições de diferentes origens)
Body-parser (para processamento de requisições JSON)
MySQL2 (para interação com o banco de dados)

Definição do projeto

- A construção de um sistema completo de CRUD (Create, Read, Update, Delete) com Angular 18 e MySQL, você precisará de duas partes principais:

  - Backend: Uma API em Node.js (por exemplo, utilizando Express) que interage com o banco de dados MySQL.
  - Frontend: Uma aplicação Angular que realiza operações CRUD através de requisições HTTP para a API.

Arquitetura do projeto

Copiar código

```
/my-crud-app
|
├── /backend                # Pasta do backend (Node.js)
|   ├── /node_modules       # Dependências do Node.js (gerado automaticamente pelo npm)
|   ├── /controllers        # Contém os controladores responsáveis por gerenciar a lógica
|   |   └── userController.js
|   ├── /models             # Contém os modelos (estruturas de dados) e funções de banco
|   |   └── userModel.js
|   ├── /routes             # Contém as rotas para a API
|   |   └── userRoutes.js
|   ├── /config             # Configurações do banco de dados e variáveis de ambiente
|   |   └── dbConfig.js
|   ├── /utils              # Funções auxiliares, como validações, autenticação, etc.
|   |   └── validate.js
|   ├── server.js           # Arquivo principal para rodar o servidor Express
|   ├── package.json        # Dependências do Node.js e scripts de execução
|   └── .env                # Arquivo c riáveis de ambiente (ex.: credenciais do banco
|
```

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

Arquitetura do projeto

Backend (Node.js)

O código do backend é dividido da seguinte forma:

- **server.js**: Configura o servidor Express e associa as rotas.
- **controllers/userController.js**: Contém a lógica das operações CRUD para os usuários.
- **models/userModel.js**: Define a interação com o banco de dados MySQL para usuários.
- **routes/userRoutes.js**: Define as rotas da API de usuários.
- **config/dbConfig.js**: Configura a conexão com o banco de dados MySQL.

Arquitetura do projeto

```
├── /frontend               # Pasta do frontend (Angular)
│   ├── /node_modules       # Dependências do Angular (gerado automaticamente pelo npm)
│   ├── /src                # Código-fonte do Angular
│   │   ├── /app            # Contém os componentes, serviços e módulos principais
│   │   │   ├── /components  # Componentes da aplicação
│   │   │   │   ├── user-list/
│   │   │   │   │   ├── user-list.component.ts   # Lógica do componente
│   │   │   │   │   ├── user-list.component.html # Template do componente
│   │   │   │   │   ├── user-list.component.css  # Estilos do componente
│   │   │   │   ├── user-form/
│   │   │   │   │   ├── user-form.component.ts   # Lógica do componente
│   │   │   │   │   ├── user-form.component.html # Template do componente
│   │   │   │   │   ├── user-form.component.css  # Estilos do componente
│   │   │   ├── /services   # Contém os serviços para comunicação com o backend
│   │   │   │   └── user.service.ts  # Serviço para interagir com a API
│   │   │   ├── /models     # Modelos para tipagem dos dados (ex.: User)
│   │   │   │   └── user.model.ts  # Modelo do usuário
│   │   │   ├── app.module.ts  # Módulo principal da aplicação
│   │   │   └── app.component.ts  # Componente principal
│   │   ├── /assets         # Arquivos estáticos, como imagens, fontes, etc.
│   │   ├── /environments   # Configura[illegible]s específicas para diferentes ambientes (desenv
│   │   └── index.html      # HTML principal da aplicação
```

Arquitetura do projeto

Frontend (Angular)

A estrutura do frontend é organizada em:

- **user-list.component.ts**: Componente que exibe todos os usuários.
- **user-form.component.ts**: Componente que permite adicionar ou editar usuários.
- **user.service.ts**: Serviço que comunica com a API backend.
- **app.component.ts**: Componente principal.
- **app.module.ts**: Define os módulos e configurações principais da aplicação.

Configuração do Backend com Node.js e MySQL

- Instalar o Node.js

Primeiro, você precisa instalar o Node.js, que pode ser feito em nodejs.org.

- Criar um novo projeto Node.js

Abra o terminal e crie um diretório para o projeto backend.

mkdir backend

cd backend

npm init -y

Configuração do Backend com Node.js e MySQL

- Instalar dependências

Instale as dependências necessárias:

npm install express mysql2 body-parser cors

express: Framework web para criar a API.
mysql2: Cliente MySQL para conectar ao banco de dados.
body-parser: Middleware para analisar o corpo das requisições.
cors: Middleware para permitir requisições de diferentes origens (necessário para o frontend Angular).

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
const express = require('express');  // Importa o framework Express, utilizado para criar o servidor e gerenciar rotas
const mysql = require('mysql2');     // Importa o cliente MySQL para interagir com o banco de dados MySQL
const bodyParser = require('body-parser');  // Importa o middleware body-parser para processar o corpo das requisições
const cors = require('cors');  // Importa o middleware CORS para permitir requisições de diferentes origens (necessário para o frontend)

const app = express();  // Cria uma instância do Express
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Configurações do CORS
app.use(cors());  // Permite que a API seja acessada por outras origens (frontends em outros servidores ou portas)

// Configura o Body Parser para lidar com JSON em requisições
app.use(bodyParser.json());
// Configuração de conexão com o banco de dados MySQL
const db = mysql.createConnection({
   host: 'localhost',  // Endereço do banco de dados
   user: 'root',  // Usuário do MySQL
   password: '',  // Senha do MySQL
   database: 'crud_example'  // Nome do banco de dados
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Conecta ao banco de dados MySQL
db.connect(err => {
    if (err) throw err;  // Se houver um erro, ele é lançado
        console.log('Conectado ao banco de dados MySQL');  // Se a conexão for bem-sucedida, exibe a mensagem no console
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Rota para criar um novo usuário
app.post('/api/users', (req, res) => {
    const { name, email } = req.body;  // Desestrutura os dados do corpo da requisição (nome e e-mail)
   const query = 'INSERT INTO users (name, email) VALUES (?, ?)';  // Query SQL para inserir o usuário no banco de dados
   db.query(query, [name, email], (err, result) => {
      if (err) throw err;  // Se houver erro, lança um erro
          res.status(201).send({ id: result.insertId, name, email });  // Retorna a resposta com o usuário recém-criado
   });
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Rota para obter todos os usuários
app.get('/api/users', (req, res) => {
    const query = 'SELECT * FROM users';  // Query SQL para obter todos os usuários
   db.query(query, (err, results) => {
      if (err) throw err;  // Se houver erro, lança um erro
      res.status(200).json(results);  // Retorna a lista de usuários
   });
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Rota para obter um usuário específico
app.get('/api/users/:id', (req, res) => {
    const query = 'SELECT * FROM users WHERE id = ?';  // Query SQL
para obter um usuário específico com base no ID
    db.query(query, [req.params.id], (err, results) => {
        if (err) throw err;  // Se houver erro, lança um erro
         res.status(200).json(results[0]);  // Retorna o usuário específico
encontrado
    });
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Rota para atualizar um usuário
app.put('/api/users/:id', (req, res) => {
    const { name, email } = req.body;  // Desestrutura os dados de nome e e-mail do corpo da requisição
   const query = 'UPDATE users SET name = ?, email = ? WHERE id = ?';  // Query SQL para atualizar os dados do usuário
   db.query(query, [name, email, req.params.id], (err, result) => {
      if (err) throw err;  // Se houver erro, lança um erro
         res.status(200).send({ id: req.params.id, name, email });  // Retorna a resposta com os dados do usuário atualizado
   });
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Rota para excluir um usuário
app.delete('/api/users/:id', (req, res) => {
    const query = 'DELETE FROM users WHERE id = ?';  // Query SQL
para excluir um usuário com base no ID
   db.query(query, [req.params.id], (err, result) => {
      if (err) throw err;  // Se houver erro, lança um erro
         res.status(200).send({ message: 'User deleted' });  // Retorna
uma mensagem indicando que o usuário foi excluído
   });
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
// Inicia o servidor na porta 3000
app.listen(3000, () => {
    console.log('Servidor rodando na porta 3000');  // Exibe a mensagem no console indicando que o servidor está ativo
});
```

Configuração do Backend com Node.js e MySQL

- Criar o arquivo server.js

```
const express = require('express');  // Importa o framework Express, utilizado para criar o servidor e gerenciar rotas
const mysql = require('mysql2');   // Importa o cliente MySQL para interagir com o banco de dados MySQL
const bodyParser = require('body-parser');  // Importa o middleware body-parser para processar o corpo das requisições
const cors = require('cors');  // Importa o middleware CORS para permitir requisições de diferentes origens (necessário para o frontend)

const app = express();  // Cria uma instância do Express

// Configurações do CORS
app.use(cors());  // Permite que a API seja acessada por outras origens (frontends em outros servidores ou portas)

// Configura o Body Parser para lidar com JSON em requisições
app.use(bodyParser.json());

// Configuração de conexão com o banco de dados MySQL
const db = mysql.createConnection({
    host: 'localhost',  // Endereço do banco de dados
    user: 'root',  // Usuário do MySQL
    password: '',  // Senha do MySQL
    database: 'crud_example'  // Nome do banco de dados
});

// Conecta ao banco de dados MySQL
db.connect(err => {
    if (err) throw err;  // Se houver um erro, ele é lançado
    console.log('Conectado ao banco de dados MySQL');  // Se a conexão for bem-sucedida, exibe a mensagem no console
});

// Rota para criar um novo usuário
app.post('/api/users', (req, res) => {
    const { name, email } = req.body;  // Desestrutura os dados do corpo da requisição (nome e e-mail)
    const query = 'INSERT INTO users (name, email) VALUES (?, ?)';  // Query SQL para inserir o usuário no banco de dados
    db.query(query, [name, email], (err, result) => {
        if (err) throw err;  // Se houver erro, lança um erro
        res.status(201).send({ id: result.insertId, name, email });  // Retorna a resposta com o usuário recém-criado
    });
});

// Rota para obter todos os usuários
app.get('/api/users', (req, res) => {
    const query = 'SELECT * FROM users';  // Query SQL para obter todos os usuários
    db.query(query, (err, results) => {
        if (err) throw err;  // Se houver erro, lança um erro
        res.status(200).json(results);  // Retorna a lista de usuários
    });
});

// Rota para obter um usuário específico
app.get('/api/users/:id', (req, res) => {
    const query = 'SELECT * FROM users WHERE id = ?';  // Query SQL para obter um usuário específico com base no ID
    db.query(query, [req.params.id], (err, results) => {
        if (err) throw err;  // Se houver erro, lança um erro
        res.status(200).json(results[0]);  // Retorna o usuário específico encontrado
    });
});

// Rota para atualizar um usuário
app.put('/api/users/:id', (req, res) => {
    const { name, email } = req.body;  // Desestrutura os dados de nome e e-mail do corpo da requisição
    const query = 'UPDATE users SET name = ?, email = ? WHERE id = ?';  // Query SQL para atualizar os dados do usuário
    db.query(query, [name, email, req.params.id], (err, result) => {
        if (err) throw err;  // Se houver erro, lança um erro
        res.status(200).send({ id: req.params.id, name, email });  // Retorna a resposta com os dados do usuário atualizado
    });
});

// Rota para excluir um usuário
app.delete('/api/users/:id', (req, res) => {
    const query = 'DELETE FROM users WHERE id = ?';  // Query SQL para excluir um usuário com base no ID
    db.query(query, [req.params.id], (err, result) => {
        if (err) throw err;  // Se houver erro, lança um erro
        res.status(200).send({ message: 'User deleted' });  // Retorna uma mensagem indicando que o usuário foi excluído
    });
});

// Inicia o servidor na porta 3000
app.listen(3000, () => {
    console.log('Servidor rodando na porta 3000');  // Exibe a mensagem no console indicando que o servidor está ativo
});
```

Configuração do Backend com Node.js e MySQL

- Criar a tabela no MySQL

```
CREATE DATABASE crud_example;

USE crud_example;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100)
);
```

Configuração do Frontend com Angular

- Instalar o Angular CLI

Se ainda não tiver o Angular CLI instalado, instale-o globalmente:

npm install -g @angular/cli

Configuração do Frontend com Angular

- Criar um novo projeto Angular

Crie um novo projeto Angular.

```
ng new crud-angular
cd crud-angular
```

Configuração do Frontend com Angular

- Instalar o módulo HTTP Client

No Angular, você precisa importar o módulo HttpClientModule para fazer requisições HTTP. Abra o arquivo app.module.ts e adicione:

```
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  imports: [
    HttpClientModule,
    // outros módulos
  ],
  // outros parâmetros
})
export class AppModule { }
```

Configuração do Frontend com Angular

- Criar o serviço de CRUD

Crie um serviço Angular para interagir com a API backend. Use o comando abaixo np bash:

ng generate service user

Configuração do Frontend com Angular

- Criar o serviço de CRUD

No arquivo user.service.ts, adicione o seguinte código:

```
import { Injectable } from '@angular/core';  // Importa o decorador Injectable para permitir que o serviço seja injetado em outros componentes
import { HttpClient } from '@angular/common/http';  // Importa o módulo HttpClient para fazer requisições HTTP
import { Observable } from 'rxjs';  // Importa o Observable para lidar com dados assíncronos

interface User {  // Interface para representar um usuário
  id?: number;  // O ID pode ser opcional (não é necessário ao criar um usuário)
  name: string;
  email: string;
}
```

Configuração do Frontend com Angular

- Criar o serviço de CRUD

No arquivo user.service.ts, adicione o seguinte código:

```
@Injectable({
  providedIn: 'root'  // Isso significa que o serviço será fornecido na raiz do aplicativo, tornando-o acessível globalmente
})
export class UserService {
  private apiUrl = 'http://localhost:3000/api/users';  // URL base da API para usuários

  constructor(private http: HttpClient) { }  // Injeta o HttpClient para poder fazer requisições HTTP
```

Configuração do Frontend com Angular

- Criar o serviço de CRUD

No arquivo user.service.ts, adicione o seguinte código:

```
 // Método para obter todos os usuários
 getUsers(): Observable<User[]> {
     return this.http.get<User[]>(this.apiUrl);  // Faz uma requisição GET para obter todos os usuários
 }

 // Método para obter um usuário específico
 getUser(id: number): Observable<User> {
    return this.http.get<User>(`${this.apiUrl}/${id}`);  // Faz uma requisição GET para obter um usuário pelo ID
 }
```

Configuração do Frontend com Angular

- Criar o serviço de CRUD

No arquivo user.service.ts, adicione o seguinte código:

```
 // Método para adicionar um novo usuário
 addUser(user: User): Observable<User> {
    return this.http.post<User>(this.apiUrl, user);  // Faz uma requisição POST para adicionar um novo usuário
 }

 // Método para atualizar os dados de um usuário
 updateUser(id: number, user: User): Observable<User> {
   return this.http.put<User>(`${this.apiUrl}/${id}`, user);  // Faz uma requisição PUT para atualizar um usuário pelo ID
 }
```

Configuração do Frontend com Angular

- Criar o serviço de CRUD

No arquivo user.service.ts, adicione o seguinte código:

```
 // Método para excluir um usuário
 deleteUser(id: number): Observable<any> {
     return  this.http.delete<any>(`${this.apiUrl}/${id}`);   // Faz uma requisição
DELETE para excluir um usuário pelo ID
  }
}
```

Configuração do Frontend com Angular

- Criar os componentes

Crie os componentes para listar, adicionar, editar e excluir usuários no bash:

ng generate component user-list
ng generate component user-form

Configuração do Frontend com Angular

- Criar os componentes

No componente user-list.component.ts, adicione a lógica para listar os usuários:

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../user.service';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css']
})
```

Configuração do Frontend com Angular

- Criar os componentes

No componente user-list.component.ts, adicione a lógica para listar os usuários:

```
export class UserListComponent implements OnInit {
  users = [];

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    this.userService.getUsers().subscribe(data => {
      this.users = data;
    });
  }
```

Configuração do Frontend com Angular

- Criar os componentes

No componente user-list.component.ts, adicione a lógica para listar os usuários:

```
  deleteUser(id: number): void {
    this.userService.deleteUser(id).subscribe(() => {
      this.users = this.users.filter(user => user.id !== id);
    });
  }
}
```

Configuração do Frontend com Angular

- Criar os componentes

No componente user-list.component.ts, adicione a lógica para listar os usuários:

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../user.service';

@Component({
  selector: 'app-user-list',
  templateUrl: './user-list.component.html',
  styleUrls: ['./user-list.component.css']
})
export class UserListComponent implements OnInit {
  users = [];

  constructor(private userService: UserService) {}

  ngOnInit(): void {
    this.userService.getUsers().subscribe(data => {
      this.users = data;
    });
  }

  deleteUser(id: number): void {
    this.userService.deleteUser(id).subscribe(() => {
      this.users = this.users.filter(user => user.id !== id);
    });
  }
}
```

Configuração do Frontend com Angular

- Criar os componentes

No user-list.component.html, crie um layout simples para exibir os usuários:

```
<h2>Lista de Usuários</h2>
<table>
 <tr>
  <th>Nome</th>
  <th>Email</th>
  <th>Ações</th>
 </tr>
 <tr *ngFor="let user of users">
  <td>{{ user.name }}</td>
  <td>{{ user.email }}</td>
  <td>
   <button (click)="deleteUser(user.id)">Excluir</button>
  </td>
 </tr>
</table>
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

No user-form.component.ts, crie o formulário para adicionar ou editar usuários:

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../user.service';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-user-form',
  templateUrl: './user-form.component.html',
  styleUrls: ['./user-form.component.css']
})
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

No user-form.component.ts, crie o formulário para adicionar ou editar usuários:

```
export class UserFormComponent implements OnInit {
  user = { name: '', email: '' };
  isEditing = false;

  constructor(
    private userService: UserService,
    private route: ActivatedRoute,
    private router: Router
  ) {}
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

No user-form.component.ts, crie o formulário para adicionar ou editar usuários:

```
ngOnInit(): void {
  const userId = this.route.snapshot.paramMap.get('id');
  if (userId) {
    this.isEditing = true;
    this.userService.getUser(+userId).subscribe(data => {
      this.user = data;
    });
  }
}
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

No user-form.component.ts, crie o formulário para adicionar ou editar usuários:

```
 saveUser(): void {
   if (this.isEditing) {
     this.userService.updateUser(this.user.id, this.user).subscribe(() => {
       this.router.navigate(['/']);
     });
   } else {
     this.userService.addUser(this.user).subscribe(() => {
       this.router.navigate(['/']);
     });
   }
 }
}
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

No user-form.component.ts, crie o formulário para adicionar ou editar usuários:

```
import { Component, OnInit } from '@angular/core';
import { UserService } from '../user.service';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-user-form',
  templateUrl: './user-form.component.html',
  styleUrls: ['./user-form.component.css']
})
export class UserFormComponent implements OnInit {
  user = { name: '', email: '' };
  isEditing = false;

  constructor(
    private userService: UserService,
    private route: ActivatedRoute,
    private router: Router
  ) {}

  ngOnInit(): void {
    const userId = this.route.snapshot.paramMap.get('id');
    if (userId) {
      this.isEditing = true;
      this.userService.getUser(+userId).subscribe(data => {
        this.user = data;
      });
    }
  }

  saveUser(): void {
    if (this.isEditing) {
      this.userService.updateUser(this.user.id, this.user).subscribe(() => {
        this.router.navigate(['/']);
      });
    } else {
      this.userService.addUser(this.user).subscribe(() => {
        this.router.navigate(['/']);
      });
    }
  }
}
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

E no arquivo user-form.component.html:

```
<h2>{{ isEditing ? 'Editar Usuário' : 'Adicionar Usuário' }}</h2>
<form (ngSubmit)="saveUser()">
  <label for="name">Nome:</label>
  <input id="name" [(ngModel)]="user.name" name="name" required />

  <label for="email">Email:</label>
  <input id="email" [(ngModel)]="user.email" name="email" required />

  <button type="submit">{{ isEditing ? 'Salvar' : 'Adicionar' }}</button>
</form>
```

Configuração do Frontend com Angular

- Adicionar o formulário de criação e edição de usuários

E no arquivo user-form.component.css:

```
/* Estilo para o título do formulário (h2) */
h2 {
  text-align: center; /* Centraliza o título */
  color: #4CAF50; /* Define uma cor verde para o título */
}

/* Estilos para o formulário (form) */
form {
  width: 300px; /* Define uma largura fixa de 300px para o formulário */
  margin: 0 auto; /* Centraliza o formulário horizontalmente na página */
  padding: 20px; /* Adiciona um espaçamento interno de 20px ao redor do formulário */
  border: 1px solid #ddd; /* Adiciona uma borda fina e de cor cinza claro */
  border-radius: 8px; /* Arredonda os cantos do formulário */
  background-color: #f9f9f9; /* Define a cor de fundo do formulário como um tom claro de cinza */
}

/* Estilos para os rótulos de cada campo (label) */
label {
  display: block; /* Faz o rótulo ocupar toda a largura disponível */
  margin-bottom: 8px; /* Adiciona uma margem abaixo do rótulo para separar do campo de entrada */
  font-weight: bold; /* Deixa o texto em negrito */
}

/* Estilos para os campos de entrada (input) */
input {
  width: 100%; /* Faz o campo de entrada ocupar 100% da largura do formulário */
  padding: 8px; /* Adiciona um preenchimento interno de 8px para tornar o campo mais espaçoso */
  margin-bottom: 16px; /* Adiciona uma margem abaixo do campo para separar os campos */
  border: 1px solid #ccc; /* Define uma borda de cor cinza claro */
  border-radius: 4px; /* Arredonda os cantos dos campos de entrada */
  font-size: 16px; /* Define o tamanho da fonte dentro do campo de entrada */
}

/* Estilos para o botão de envio (button) */
button {
  width: 100%; /* Faz o botão ocupar 100% da largura disponível */
  padding: 10px; /* Adiciona um preenchimento interno de 10px ao botão */
  background-color: #4CAF50; /* Define a cor de fundo do botão como verde */
  color: white; /* Define a cor do texto do botão como branco */
  border: none; /* Remove a borda padrão do botão */
  border-radius: 4px; /* Arredonda os cantos do botão */
  font-size: 16px; /* Define o tamanho da fonte dentro do botão */
  cursor: pointer; /* Exibe o cursor como uma mãozinha quando passa sobre o botão */
}

/* Estilos para o botão quando o mouse passa por cima (hover) */
button:hover {
  background-color: #45a049; /* Altera a cor de fundo para um verde mais escuro ao passar o mouse */
}

/* Estilos para o botão quando ele está em foco (quando é selecionado) */
button:focus {
  outline: none; /* Remove o contorno padrão quando o botão é focado */
}

/* Estilos para o campo de entrada quando está em foco (quando é selecionado) */
input:focus {
  border-color: #4CAF50; /* Muda a cor da borda para verde quando o campo é focado */
  outline: none; /* Remove o contorno padrão quando o campo de entrada é focado */
}

/* Estilos responsivos para telas pequenas */
@media (max-width: 600px) {
  form {
    width: 80%; /* Reduz a largura do formulário para 80% da tela em dispositivos com largura menor que 600px */
  }
}
```

Configuração das rotas do Angular

No arquivo app-routing.module.ts, defina as rotas:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { UserListComponent } from './user-list/user-list.component';
import { UserFormComponent } from './user-form/user-form.component';

const routes: Routes = [
  { path: '', component: UserListComponent },
  { path: 'add', component: UserFormComponent },
  { path: 'edit/:id', component: UserFormComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

Testando a Aplicação

- Inicie o servidor backend no bash:

node server.js

- Inicie a aplicação Angular no bash:

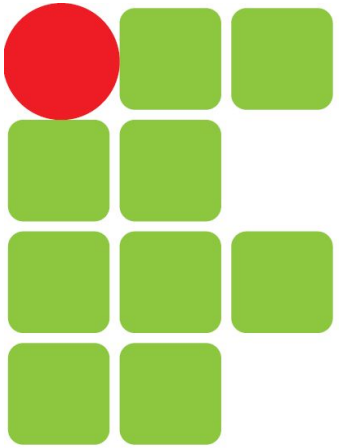
ng serve

Referências

- Site oficial do angular: https://angular.dev/
- Introdução ao Angular https://codelabs.developers.google.com/introduction-to-angular?hl=pt-br#0
- Angular Documentation https://devdocs.io/angular/
- Guide to AngularJS Documentation https://docs.angularjs.org/guide

Referências

- Projeto Angular: https://github.com/savanihd/Angular-18-CRUD-Application-Tutorial-Example
- Aplicação CRUD Angular 18 - Tutorial Exemplo https://github.com/daniel-abella/angular18-crud//?tab=readme-ov-file#arquivo-srcapppostedite ditcomponenthtml
- Angular 18 Básico https://github.com/daniel-abella/angular1 co



INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO