	Curso: MBA em Full Stack Web Development
	Disciplina: Angular Bootcamp
	Docente: Nicoly Figueredo Pessoa de Almeida
	Assunto: Construção do projeto Angular

Após ter configurado o ambiente e criado o projeto com base no tutorial. Abra o projeto no VSCode e siga os passos a seguir para continuar o desenvolvimento:

1. Primeiro vamos rodar a nossa aplicação, digite no terminal:
   `ng serve`

   - Acesse: http://localhost:4200/ e veja como a aplicação está agora.
   - O que você está vendo é a tela padrão de um projeto Angular, ela está definida em **src/app/app.component.html**

2. Continuando a construção do nosso projeto, vamos realizar importações de todos os componentes do Angular Material, para isso vamos criar um módulo do Material, abra um novo terminal e digite:
   `ng generate module shared/material`

   - Esse comando irá criar o módulo dentro de uma pasta chamada shared, usaremos essa pasta para armazenar tudo que será compartilhado por vários componentes (models, services, dentre outros).

3. No arquivo que acabamos de criar (**src/app/shared/material/material.module.ts**) vamos importar todos os componentes do Angular Material para isso seu código deve ficar igual ao do seguinte arquivo:
   https://docs.google.com/document/d/1ogwdn7srZdGsnCUOABtfqf_d4dytH7jdNq_IIm9EWmI/edit?usp=sharing

4. Em **src/app/app.module.ts** precisamos importar alguns módulos, sendo eles:

   - **CommonModule:** Este módulo fornece diretivas e serviços comuns utilizados na construção de componentes. É importado para que você possa usar diretivas básicas como ngIf e ngFor.
   - **FormsModule**: Este módulo fornece suporte para a criação de formulários no Angular. Ele inclui diretivas para binding de dados bidirecional, como ngModel, que permite que você crie e manipule formulários no Angular.
   - **ReactiveFormsModule**: Este módulo é usado para criar formulários reativos no Angular, onde você pode controlar e validar campos de formulário usando objetos do tipo FormControl e FormGroup.
   - **BrowserAnimationsModule**: Este módulo é usado para adicionar animações à aplicação Angular quando está sendo executada em um ambiente de navegador. Ele permite que você utilize funcionalidades de animação do Angular, como transições de estado e animações CSS.

- **RouterModule**: Este módulo é usado para configurar e gerenciar as rotas da aplicação Angular. Ele é necessário para definir as rotas que a aplicação seguirá e como os componentes são carregados quando as rotas são acessadas.
- **HttpClientModule**: Este módulo fornece as funcionalidades para fazer solicitações HTTP em uma aplicação Angular. Ele é usado para realizar requisições AJAX, recuperar dados de APIs e se comunicar com serviços externos.
- **MaterialModule**: Este é o módulo que acabamos de criar para encapsular componentes, diretivas e estilos para criar uma interface do usuário consistente e agradável. Ele permite que você use componentes e estilos específicos do Material Design em sua aplicação Angular.
- **Schemas: [CUSTOM_ELEMENTS_SCHEMA]**: O schema é uma configuração opcional que permite flexibilidade na validação de templates. CUSTOM_ELEMENTS_SCHEMA é usado quando você deseja permitir elementos HTML personalizados (por exemplo, componentes da Web personalizados) em seus templates, estendendo a validação do Angular para incluí-los.

Ao final o arquivo deverá ficar assim:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { MaterialModule } from './shared/material/material.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule,
    MaterialModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

5. Vamos criar uma interface cliente (uma interface é um contrato, uma especificação de uma entidade), para isso crie uma pasta models dentro de shared, e crie um arquivo Cliente.ts . Ficará um caminho **src/app/shared/models/Cliente.ts** , o arquivo deverá ficar assim**:**

```
export interface Cliente {
    id: number;
    nome: string;
    cpf: string;
    email: string;
    observacoes: string;
    ativo: boolean;
}
```

6. A partir de agora iremos consumir o endpoint de clientes da nossa API:
    - A API está disponível no link: https://angular-api.xwhost.com.br/api/
    - Antes de iniciarmos nosso serviço de consumo, vamos configurar o arquivo de environment. Para isso iremos utilizar o comando:
      `ng generate environments`
    - Esse comando está disponível a partir do Angular 15.1 (vou deixar ao final do arquivo uma matéria sobre environments no Angular para quem tiver interesse).
    - Ele irá criar dois arquivo em **src/environments**, iremos utilizar para fins de teste o arquivo **environments.development.ts**, que deverá ficar assim:

```
export const environment = {
    production: false,
    api: 'https://angular-api.xwhost.com.br/api'
};
```

7. Para criar o serviço de clientes do nosso projeto, digite no terminal:
   `ng generate service shared/services/cliente`
8. Agora acesse o arquivo **src/app/shared/services/cliente.service.ts** (que acabamos de criar) para configurarmos os métodos de GET, PUT, DELETE e POST. Ao final o arquivo deverá ficar como esse:

```
import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environment.development';
import { Cliente } from '../models/Cliente';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
```

```
export class ClienteService {
  api = `${environment.api}/clientes/`;

  constructor(private clienteHttp: HttpClient) { }

  inserir(novoCliente: Cliente): Observable<Cliente> {
    return this.clienteHttp.post<Cliente>(
      this.api, novoCliente);
  }

  listar(): Observable<Cliente[]> {
    return this.clienteHttp.get<Cliente[]>(this.api);
  }

  listar_paginado(page: number, pageSize: number):
Observable<Cliente[]> {
    return this.clienteHttp
    .get<Cliente[]>(`${this.api}?page=${page}&pageSize=${pageSize}`);
  }

  deletar(idCliente: number): Observable<object> {
    return this.clienteHttp.delete(`${this.api}${idCliente}`);
  }

  pesquisarPorId(id: number): Observable<Cliente> {
    return this.clienteHttp.get<Cliente>(`${this.api}${id}`);
  }

  atualizar(cliente: Cliente): Observable<Cliente> {
    return this.clienteHttp.put<Cliente>(`${this.api}${cliente.id}`,
cliente);
  }
}
```

- No código acima cada função é responsável por um método HTTP, a função **inserir** vai cadastrar um novo cliente na aplicação, a **listar** irá fazer um GET para listar todos os clientes, a função **deletar** irá remover um cliente específico, a **pesquisarPorId** irá buscar um cliente específico e a **atualizar** irá editar um cliente específico.

- Cada função recebe um parâmetro que será utilizado na requisição e tem como retorno um Observable (são como mensageiros que entregam informações de forma assíncrona. Eles permitem que diferentes partes da aplicação saibam quando algo aconteceu e reajam a isso), ao final do tutorial deixei uma matéria que fala mais sobre observables para quem tiver interesse.

9. Agora vamos criar nosso módulo de clientes (nele iremos concentrar todos os componentes relacionados a clientes. Para isso digite no terminal:
`ng generate module pages/cliente`

10. Agora vamos instalar o **NGX MASK** (serve para criarmos máscaras, como para CPFs, números de celular. Ex: “XXX.XXX.XXX-XX” e “(XX) XXXXX-XXXX”). No terminal, digite:

```
npm install --save ngx-mask
```

11. Agora acesse **src/app/pages/cliente/cliente.module.ts** para realizarmos as importações necessárias no módulo que acabamos de criar (já expliquei anteriormente para que serve cada uma das importações). Segue como ficará o arquivo:

```
import { CUSTOM_ELEMENTS_SCHEMA, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';
import { RouterModule } from '@angular/router';
import { MaterialModule } from
'src/app/shared/material/material.module';
import { HttpClientModule } from '@angular/common/http';

@NgModule({
  declarations: [],
  imports: [
    CommonModule,
    FormsModule,
    BrowserAnimationsModule,
    ReactiveFormsModule,
    RouterModule,
    HttpClientModule,
    MaterialModule,
    NgxMaskDirective,
    NgxMaskPipe
   ],
  schemas: [CUSTOM_ELEMENTS_SCHEMA],
  providers: [provideNgxMask()],
  exports: []
})
export class ClienteModule { }
```

12. Acesse **src/apps/app.component.module** e realize as importações necessárias para utilizarmos o NGX MASK:

```
imports: [
    BrowserModule,
```

```
    AppRoutingModule,
    BrowserAnimationsModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule,
    MaterialModule,
    NgxMaskDirective,
    NgxMaskPipe
  ],
  providers: [provideNgxMask()],
```

13. É necessário criar agora o componente de cadastro de clientes, para isso digite no terminal:
    `ng generate component pages/cliente/cadastro-cliente`
    - O comando irá criar nosso componente de cadastro em **src/app/pages/cliente**
    - Observe que um componente no Angular é formado por 4 arquivos, um arquivo **.html** que contém toda a parte visual, um arquivo **.scss** onde podemos personalizar o estilo do componente, um arquivo **.spec.ts** para configuração de testes e um arquivo **.ts** onde fica a lógica de funcionamento do componente
14. Acesse **src/pages/cliente/cadastro-cliente/cadastro-cliente.component.html** e vamos configurar o código de exibição do formulário de cadastro de novo cliente. Segue um modelo de html (podem personalizar e exibir a lista de clientes como preferirem):

```
<div class="container mt-5 d-flex justify-content-center" >
    <div>
        <h1>Novo cliente</h1>

            <form [formGroup]="formGroup" class="example-form mt-4"
(ngSubmit)="cadastrar()">

            <!--Input de nome do cliente-->
            <small class="text-danger mb-2"
*ngIf="formGroup.get('nome')?.errors &&
            formGroup.get('nome')?.hasError('required')">Campo
obrigatório</small>
            <mat-form-field class="example-full-width">
                <mat-label>Nome</mat-label>
                <input matInput formControlName="nome">
            </mat-form-field>

            <!--Input de cpf do cliente-->
            <small class="text-danger mb-2"
*ngIf="formGroup.get('cpf')?.errors &&
```

```
            formGroup.get('cpf')?.hasError('required')">Campo
    obrigatório</small>
            <mat-form-field class="example-full-width">
                <mat-label>CPF</mat-label>
                <input matInput formControlName="cpf"
    mask="000.000.000-00">
            </mat-form-field>

            <!--Input de email do cliente-->
            <small class="text-danger mb-2"
    *ngIf="formGroup.get('email')?.errors &&
            formGroup.get('email')?.hasError('required')">Campo
    obrigatório</small>
            <mat-form-field class="example-full-width">
                <mat-label>Email</mat-label>
                <input matInput formControlName="email">
            </mat-form-field>

            <!--Input de observacoes do cliente-->
            <small class="text-danger mb-2"
    *ngIf="formGroup.get('observacoes')?.errors &&
            formGroup.get('observacoes')?.hasError('required')">Campo
    obrigatório</small>
            <mat-form-field class="example-full-width">
                <mat-label>Observações</mat-label>
                <textarea matInput
    formControlName="observacoes"></textarea>
            </mat-form-field>

            <!--Bolas de selecao de status do cliente-->
            <mat-radio-group formControlName="ativo">
                <mat-radio-button [value]="true">Ativo</mat-radio-
    button>
                <mat-radio-button [value]="false">Inativo</mat-radio-
    button>
            </mat-radio-group>

            <!--Botao de submissao do formulario-->
            <div class="d-flex justify-content-center">
                <button [disabled]="!formGroup.valid" type="submit"
    mat-raised-button color="primary">
                    Cadastrar
                </button>
            </div>
        </form>
```

```
        </div>
    </div>
```

- ** :** Isso define uma div com classes de estilo para centralizar seu conteúdo verticalmente na tela.
- **<h1>Novo cliente</h1>:** Um título "Novo cliente" é exibido acima do formulário.
- **<form [formGroup]="formGroup" class="example-form mt-4" (ngSubmit)="cadastrar()">:** Aqui, você cria um formulário que está vinculado ao formGroup no seu componente. O FormGroup do Angular Forms é usado para gerenciar o estado do formulário. O evento (ngSubmit) é acionado quando o formulário é enviado e chama a função cadastrar() no seu componente.
- **<small>:** Pequenos elementos HTML são usados para exibir mensagens de erro caso os campos do formulário não sejam preenchidos corretamente. As mensagens são exibidas condicionalmente usando a diretiva *ngIf com base nos erros no campo específico.
- **<mat-form-field>:** Isso é um componente de campo do Angular Material usado para envolver cada campo de entrada no formulário. O atributo matInput é usado nos inputs para aplicar estilos e funcionalidades específicas do Angular Material.
- **<input>:** Inputs de texto para os campos "Nome", "CPF" e "Email". Cada input está vinculado a um formControlName, que corresponde a uma propriedade no formGroup do componente.
- **<textarea>:** Um textarea para o campo "Observações". Assim como os inputs, ele também está vinculado a um formControlName.
- **<mat-radio-group>:** Isso cria um grupo de radio buttons para o campo "Ativo". Os radio buttons são definidos dentro do <mat-radio-group> e cada um possui um [value] associado a true ou false, correspondendo às opções "Ativo" e "Inativo".
- **<button [disabled]="!formGroup.valid" type="submit" mat-raised-button color="primary">Cadastrar</button>:** Este é um botão de submissão que é desabilitado ([disabled]="!formGroup.valid") a menos que o formulário esteja válido. O botão está configurado para enviar o formulário quando clicado.

15. Acesse **src/pages/cliente/cadastro-cliente/cadastro-cliente.component.ts** e vamos configurar a lógica de cadastro de novo cliente. Segue um exemplo de configuração:

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { Cliente } from 'src/app/shared/models/Cliente';
import { ClienteService } from
'src/app/shared/services/cliente.service';
import Swal from 'sweetalert2';

@Component({
```

```
  selector: 'app-cadastro-cliente',
  templateUrl: './cadastro-cliente.component.html',
  styleUrls: ['./cadastro-cliente.component.scss']
})
export class CadastroClienteComponent implements OnInit{
  formGroup: FormGroup;

  constructor(private clienteService: ClienteService, private router:
Router){
    this.formGroup = new FormGroup({
      id: new FormControl(null),
      nome: new FormControl('', Validators.required),
      cpf: new FormControl('', Validators.required),
      email: new FormControl('', [Validators.required,
Validators.email]),
      observacoes: new FormControl('', Validators.required),
      ativo: new FormControl(true)
    });
  }

  ngOnInit(): void {

  }

  cadastrar() {
    const cliente: Cliente = this.formGroup.value;
    this.clienteService.inserir(cliente).subscribe({
      next: () => {
        Swal.fire({
          icon: 'success',
          title: 'Sucesso',
          text: 'Cliente cadastrado com sucesso!',
          showConfirmButton: false,
          timer: 1500
        })
        this.router.navigate(['/cliente'])
      },
      error: (error) => {
        console.error(error)
        Swal.fire({
          icon: 'error',
          title: 'Oops...',
          text: 'Erro ao cadastrar cliente!',
        })
      }})
```

```
  }
}
```

- **selector:** Define o seletor do componente como 'app-cadastro-cliente', o que significa que você pode usar esse componente em seus templates HTML com a tag <app-cadastro-cliente></app-cadastro-cliente>.
- **templateUrl e styleUrls:** Define os arquivos de template HTML e de estilo CSS associados a este componente. Isso permite que você defina a aparência e o layout do componente.
- **O construtor** é um método que é executado quando uma instância deste componente é criada.
- Ele recebe duas injeções de dependência: clienteService e router. O clienteService é usado para interagir com os dados dos clientes e o router é usado para navegar entre páginas/componentes.
- Dentro do construtor, o formGroup é inicializado. Ele contém uma série de FormControl, cada um representando um campo no formulário de cadastro de clientes. Os campos incluem id, nome, cpf, email, observacoes, e ativo. As validações são aplicadas aos campos nome, cpf, email, e observacoes, garantindo que sejam campos obrigatórios e que o email esteja no formato correto.
- **Função cadastrar():** Esta função é chamada quando o formulário é enviado, normalmente através do evento (ngSubmit) definido no formulário. Dentro da função, os valores do formulário são obtidos usando this.formGroup.value e armazenados na variável cliente como uma instância da classe Cliente. Isso permite que os dados do formulário sejam transformados em um objeto Cliente.
- Em seguida, a função chama o método inserir do serviço clienteService para inserir o cliente no sistema. Provavelmente, o serviço fará uma solicitação HTTP (por exemplo, POST) para enviar os dados do cliente para o servidor.
- A função subscribe é usada para ouvir a resposta da chamada HTTP. Se a chamada for bem-sucedida (next), uma mensagem de sucesso é exibida usando Swal.fire, informando que o cliente foi cadastrado com sucesso. Se ocorrer um erro (error), uma mensagem de erro é exibida usando Swal.fire, informando que ocorreu um erro ao cadastrar o cliente.

16. Para conseguir visualizar o arquivo que está desenvolvendo você pode ir em **src/app/app.component.html** apague tudo que está no arquivo e adicione o seletor do nosso componente:

```
<app-cadastro-cliente></app-cadastro-cliente>
```

- Lembrando que para conseguir visualizar é necessário exportar o arquivo no módulo de clientes (assim ele ficará visível e disponível para os outros componentes). Vá para **src/app/pages/cliente/cliente.module.ts** e nos exports do módulo coloque:

```
    exports: [ CadastroClienteComponent ]
```

- Vá até o navegador para ver como ficou o nosso componente

17. Agora vamos criar outro componente, no terminal digite:

```
ng generate component pages/cliente/listagem-cliente
```

18. Acesse **src/pages/cliente/listagem-cliente/listagem-cliente.component.html** e vamos configurar o código de exibição da listagem de cliente. Segue um modelo de html (podem personalizar e exibir a lista de clientes como preferirem):

```
<div class="container mt-5">
    <div class="d-flex justify-content-between">
      <h1>Listagem de clientes</h1>

      <button mat-raised-button color="primary"
routerLink="/cliente/novo">Novo cliente</button>
    </div>

    <table mat-table class="mt-5" [dataSource]="dataSource">
      <!-- Coluna ID -->
      <ng-container matColumnDef="id">
        <th mat-header-cell *matHeaderCellDef> No. </th>
        <td mat-cell *matCellDef="let element"> {{element.id}} </td>
      </ng-container>

      <!-- Coluna Nome -->
      <ng-container matColumnDef="nome">
        <th mat-header-cell *matHeaderCellDef> Nome </th>
        <td mat-cell *matCellDef="let element"> {{element.nome}} </td>
      </ng-container>

      <!-- Coluna CPF -->
      <ng-container matColumnDef="cpf">
        <th mat-header-cell *matHeaderCellDef> CPF </th>
        <td mat-cell *matCellDef="let element"> {{element.cpf}} </td>
      </ng-container>

      <!-- Coluna Email -->
      <ng-container matColumnDef="email">
        <th mat-header-cell *matHeaderCellDef> Email </th>
        <td mat-cell *matCellDef="let element"> {{element.email}} </td>
      </ng-container>

      <!-- Coluna Status -->
      <ng-container matColumnDef="status">
        <th mat-header-cell *matHeaderCellDef> Status </th>
        <td mat-cell *matCellDef="let element"> {{element.ativo ?
'Ativo' : 'Inativo'}} </td>
      </ng-container>
```

```
        <!-- Coluna Funcoes -->
        <ng-container matColumnDef="funcoes">
          <th mat-header-cell *matHeaderCellDef> </th>
          <td mat-cell *matCellDef="let element">
            <mat-icon fontIcon="edit" [routerLink]="'/cliente/editar/' +
element.id"></mat-icon>
            <mat-icon fontIcon="delete"
(click)="deletarCLiente(element.id)"></mat-icon>
          </td>
        </ng-container>

        <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
        <tr mat-row *matRowDef="let row; columns:
displayedColumns;"></tr>
      </table>

      <mat-paginator [pageSizeOptions]="[5, 10, 20]"
        showFirstLastButtons [length]="dataSource.data.length + 1"
(page)="onPageChange($event)">
      </mat-paginator>
    </div>
```

- **<button mat-raised-button color="primary" routerLink="/cliente/novo">**: Isso cria um botão "Novo cliente" usando o Angular Material com um link de rota para a página de criação de um novo cliente. Quando clicado, ele navegará para a rota "/cliente/novo".
- **<table mat-table class="mt-5" [dataSource]="dataSource">**: Isso define uma tabela usando o Angular Material e vincula-a a um dataSource que deve conter os dados que serão exibidos na tabela.
- **<ng-container>:** Aqui, várias colunas da tabela são definidas usando o Angular's matColumnDef. Cada coluna tem um cabeçalho e células de dados correspondentes. Por exemplo, a coluna "Nome" exibirá o nome do cliente e a coluna "CPF" exibirá o CPF do cliente.
- **<tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>**: Isso define a linha de cabeçalho da tabela e usa displayedColumns para determinar quais colunas são exibidas.
- **<tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>**: Isso define as linhas de dados da tabela e usa displayedColumns para determinar quais colunas são exibidas para cada linha.
- **<ng-container matColumnDef="funcoes">**: Isso define uma coluna chamada "Funções" que conterá ícones de edição e exclusão para cada cliente.
- **<td mat-cell *matCellDef="let element">**: Nesta célula, são exibidos ícones para edição e exclusão de clientes. O ícone "edit" tem um atributo [routerLink] que gera um link para a edição do cliente com base no element.id, enquanto o

ícone "delete" tem um evento (click) que chama a função deletarCliente(element.id) quando clicado.

- **<mat-paginator [pageSizeOptions]="[5, 10, 20]" showFirstLastButtons [length]="dataSource.data.length + 1" (page)="onPageChange($event)">**: Isso configura um componente de paginação do Angular Material. Ele permite que o usuário navegue pelas diferentes páginas de clientes na tabela e escolha quantos itens por página deseja ver. O número de páginas é determinado pelo comprimento dos dados no dataSource.

- A função **onPageChange($event)** é chamada quando o usuário altera a página na paginação, permitindo que você atualize os dados exibidos com base na página selecionada.

19. Acesse **src/pages/cliente/listagem-cliente/listagem-cliente.component.ts** e vamos configurar a lógica de exibição e deleção de novo cliente. Segue um exemplo de configuração:

```
import { AfterViewInit, Component, ViewChild } from '@angular/core';
import { MatPaginator, PageEvent } from '@angular/material/paginator';
import { MatTableDataSource } from '@angular/material/table';
import { Cliente } from 'src/app/shared/models/Cliente';
import { ClienteService } from
'src/app/shared/services/cliente.service';
import Swal from 'sweetalert2';

@Component({
  selector: 'app-listagem-cliente',
  templateUrl: './listagem-cliente.component.html',
  styleUrls: ['./listagem-cliente.component.scss']
})
export class ListagemClienteComponent implements AfterViewInit {
  displayedColumns: string[] = ['id', 'nome', 'cpf', 'email', 'status',
'funcoes'];
  dataSource = new MatTableDataSource<Cliente>;

  @ViewChild(MatPaginator) paginator!: MatPaginator;

  constructor(private clienteService: ClienteService){
  }

  ngAfterViewInit() {
    this.listarClientes(1, 5)
  }

  listarClientes(page: number, pageSize: number) {
    this.clienteService.listar_paginado(page,
pageSize).subscribe(clientes => {
      this.dataSource.data = clientes;
```

```
      });
    }

    onPageChange(event: PageEvent) {
      const pageIndex = event.pageIndex + 1;
      const pageSize = event.pageSize;
      this.listarClientes(pageIndex, pageSize);
    }

    deletarCliente(id: number){
      Swal.fire({
        title: 'Você tem certeza que deseja deletar?',
        text: "Não tem como reverter essa ação",
        icon: 'warning',
        showCancelButton: true,
        confirmButtonColor: 'red',
        cancelButtonColor: 'grey',
        confirmButtonText: 'Deletar'
      }).then((result) => {
        if (result.isConfirmed) {
          this.clienteService.deletar(id).subscribe({
            next: () => {
              Swal.fire({
                icon: 'success',
                title: 'Sucesso',
                text: 'Cliente deletado com sucesso!',
                showConfirmButton: false,
                timer: 1500
              })
              this.listarClientes(1,5)
            },
            error: (error) => {
              console.error(error)
              Swal.fire({
                icon: 'error',
                title: 'Oops...',
                text: 'Erro ao deletar cliente!',
              })
            }})
        }})}
  }
```

- **displayedColumns:** Um array de strings que define as colunas que serão exibidas na tabela de clientes. Cada string corresponde a um nome de coluna.

- **dataSource:** Uma instância de MatTableDataSource<Cliente> que serve como a fonte de dados da tabela. Ela é vinculada à tabela no template.
- **@ViewChild(MatPaginator) paginator!: MatPaginator;**: Usa o decorator @ViewChild para obter uma referência ao componente MatPaginator da biblioteca Angular Material. Isso permitirá o controle da paginação na tabela.
- O método **ngAfterViewInit** é chamado após a inicialização da visualização. Nele, a função listarClientes é chamada para listar os clientes na página inicial (página 1 e tamanho de página 5).
- **listarClientes():** Esta função chama o método listar_paginado do ClienteService para obter a lista de clientes paginada. Em seguida, os dados obtidos são atribuídos à propriedade dataSource para atualizar a tabela.
- **onPageChange():** Esta função é chamada quando o usuário altera a página na paginação. Ela obtém o número da página e o tamanho da página selecionados e chama a função listarClientes para atualizar os dados da tabela de acordo com a página selecionada.
- **deletarCliente():** Esta função é chamada quando o usuário clica no ícone de exclusão na tabela. Ela chama o método deletar do ClienteService para excluir um cliente com o ID especificado. Em caso de sucesso, exibe uma mensagem de sucesso usando Swal.fire e, em seguida, atualiza a lista de clientes. Em caso de erro, exibe uma mensagem de erro.

20. Lembrando que para conseguir visualizar é necessário exportar o arquivo no módulo de clientes (assim ele ficará visível e disponível para os outros componentes). Vá para **src/app/pages/cliente/cliente.module.ts** e nos exports do módulo coloque:

```
    exports: [
        ListagemClienteComponent,
        CadastroClienteComponent
      ]
```

21. Agora você vai perceber que ao clicar no botão de novo cliente, ainda não estamos navegando para a tela de novo cliente. Para conseguirmos precisamos configurar as rotas da nossa aplicação. Vá para **src/app/app-routing.module.ts,** ele deverá ficar mais ou menos assim:

```
    import { NgModule } from '@angular/core';
    import { RouterModule, Routes } from '@angular/router';
    import { CadastroClienteComponent } from './pages/cliente/cadastro-
    cliente/cadastro-cliente.component';
    import { ListagemClienteComponent } from './pages/cliente/listagem-
    cliente/listagem-cliente.component';

    const routes: Routes = [
      {
        path: 'cliente',
        children: [
          {
```

```
        path: 'novo',
        component: CadastroClienteComponent
      },
      {
        path: 'editar/:id',
        component: CadastroClienteComponent
      },
      {
        path: '',
        component: ListagemClienteComponent,
      },
    ]
  },
  {
    path: '',
    component: ListagemClienteComponent,
  },
];

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

22. Agora, para conseguirmos exibir no navegador todos os componentes de cada rota que acessamos precisamos ir para **src/app/app.component.html**, remova tudo que tem no arquivo e coloque:

```
<router-outlet></router-outlet>
```

23. Falta apenas configurarmos a edição de cliente, vocês já devem ter percebido que utilizaremos o mesmo componente tanto para cadastro quanto para edição. Vá para **src/app/cliente/cadastro-cliente.component.ts** e ele deve ficar mais ou menos assim:

```
import { Component, OnInit } from '@angular/core';
import { FormControl, FormGroup, Validators } from '@angular/forms';
import { ActivatedRoute, Router } from '@angular/router';
import { Cliente } from 'src/app/shared/models/Cliente';
import { ClienteService } from
'src/app/shared/services/cliente.service';
import Swal from 'sweetalert2';

@Component({
  selector: 'app-cadastro-cliente',
  templateUrl: './cadastro-cliente.component.html',
```

```
  styleUrls: ['./cadastro-cliente.component.scss']
})
export class CadastroClienteComponent implements OnInit{
  editar;
  formGroup: FormGroup;

  constructor(private clienteService: ClienteService, private router:
Router, private route: ActivatedRoute){
    this.formGroup = new FormGroup({
      id: new FormControl(null),
      nome: new FormControl('', Validators.required),
      cpf: new FormControl('', Validators.required),
      email: new FormControl('', [Validators.required,
Validators.email]),
      observacoes: new FormControl('', Validators.required),
      ativo: new FormControl(true)
    });
    this.editar = false
  }

  ngOnInit(): void {
    if (this.route.snapshot.params["id"]){
      this.editar = true
this.clienteService.pesquisarPorId(this.route.snapshot.params["id"]).su
bscribe(
        cliente => {
          this.formGroup.patchValue(cliente)
        }
      )
    }
  }

  cadastrar() {
    const cliente: Cliente = this.formGroup.value;
    if (this.editar) {
      this.clienteService.atualizar(cliente).subscribe({
        next: () => {
          Swal.fire({
            icon: 'success',
            title: 'Sucesso',
            text: 'Cliente atualizado com sucesso!',
            showConfirmButton: false,
            timer: 1500
          })
          this.router.navigate(['/cliente']);
```

```
          },
          error: (error) => {
            console.error(error);
            Swal.fire({
              icon: 'error',
              title: 'Oops...',
              text: 'Erro ao atualizar cliente!',
            });
          }
        });
      } else {
        // Modo de criação
        this.clienteService.inserir(cliente).subscribe({
          next: () => {
            Swal.fire({
              icon: 'success',
              title: 'Sucesso',
              text: 'Cliente cadastrado com sucesso!',
              showConfirmButton: false,
              timer: 1500
            })
            this.router.navigate(['/cliente']);
          },
          error: (error) => {
            console.error(error);
            Swal.fire({
              icon: 'error',
              title: 'Oops...',
              text: 'Erro ao cadastrar cliente!',
            });
          }
        });
      }
    }
  }
```

- No início do código, você adicionou uma propriedade editar que será usada para determinar se o componente está no modo de edição ou criação. Inicialmente, this.editar é definido como false, o que significa que o componente está no modo de criação por padrão.
- No método ngOnInit(), você adicionou uma lógica para verificar se a rota atual possui um parâmetro id. Se houver um parâmetro id, isso indica que o componente deve entrar no modo de edição. Portanto, this.editar é definido como true, e você faz uma chamada ao serviço para obter os dados do cliente com o id especificado.

- Usando this.formGroup.patchValue(cliente), você preenche o formulário com os dados do cliente existente, garantindo que os campos do formulário sejam populados com os valores corretos para edição.

- Método cadastrar(): No método cadastrar(), você adicionou uma lógica condicional que verifica se o componente está no modo de edição (this.editar). Se estiver no modo de edição, ele chama o método atualizar do serviço ClienteService, passando os dados do cliente atualizados. Caso contrário, se estiver no modo de criação, ele chama o método inserir do serviço ClienteService, passando os dados do novo cliente.

- Esta abordagem permite que o mesmo formulário seja usado tanto para criar quanto para atualizar clientes, com base no contexto em que o componente é acessado.

24. Vamos relembrar? Configuramos todo o CRUD de clientes, e fizemos as configurações necessárias para exibir na tela. Que tal você criar uma navbar para o nosso sistema? Para melhorar a navegação dele? E criar uma tela de home para ele, com um textinho inicial, uma imagem, o que a sua imaginação permitir. Além disso, personalize o sistema para deixá-lo ainda mais bonito.

25. Segue o link do github do código desenvolvido nesse tutorial: https://github.com/Nicoly-Almeida/sistema_bancario/

Matérias úteis:

- Matéria que também fala sobre uma maneira de organização de pastas de um projeto Angular, é um padrão diferente do que estamos utilizando nesse projeto, mas também é um padrão bastante aceito: https://belmirofss.medium.com/minha-nova-estrutura-de-pastas-para-angular-escal%C3%A1vel-limpa-e-f%C3%A1cil-93b6ffb203d9

- Matéria que explica a diferença entre interface e type em Typescript: https://viniciusestevam.medium.com/principais-diferen%C3%A7as-entre-types-e-interfaces-em-typescript-a00c945e5357#:~:text=interface%20%3A%20definir%20estruturas%20de%20objetos,%2C%20type%2Dguards%20%2C%20etc.

- Matéria sobre Environments no Angular: https://dev.to/felipemsfg/angular-environment-mbp

- Matéria sobre Observables: https://dev.to/felipedsc/observables-como-funcionam-15eb