	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. Para o httpClient funcionar corretamente é necessário declará-lo na lista de providers no arquivo **app.config.ts**:

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZonelessChangeDetection(),
    provideRouter(routes),
    provideClientHydration(withEventReplay()),
    provideHttpClient(),

  ]
};
```

2. 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

3. Acesse **src/app/pages/cliente/cadastro-cliente/cadastro-cliente.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.

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

```
import { Component } from '@angular/core';
import { FormBuilder, FormControl, FormGroup, FormsModule,
ReactiveFormsModule, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { ClienteService } from
'../../../shared/services/cliente/cliente-service';
import { Cliente } from '../../../shared/models/cliente';
import Swal from 'sweetalert2';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatRadioModule } from '@angular/material/radio';
import { CommonModule } from '@angular/common';
import {MatInputModule} from '@angular/material/input';

@Component({
  selector: 'app-cadastro-cliente',
  imports: [MatInputModule, MatFormFieldModule, MatRadioModule,
CommonModule, FormsModule, ReactiveFormsModule],
  templateUrl: './cadastro-cliente.html',
  styleUrl: './cadastro-cliente.scss'
})
export class CadastroCliente {
  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.

5. Acesse **src/pages/cliente/cadastro-cliente/cadastro-cliente.scss** e copie esse css:

```scss
.example-form {
    min-width: 150px;
    max-width: 500px;
    width: 100%;
  }

  .example-full-width {
    width: 100%;
  }

  .d-flex{
    display: flex;
    justify-content: center;
    align-items: center;
    margin-top: 50px;
  }
```

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

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

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

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

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

8. Acesse **src/pages/cliente/listagem-cliente/listagem-cliente.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):

```html
<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.

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

```
import { AfterViewInit, Component, ViewChild, inject, signal } from
'@angular/core';
import { CommonModule } from '@angular/common';
import { RouterLink } from '@angular/router';

import { MatTableDataSource, MatTableModule } from
'@angular/material/table';
import { MatPaginator, MatPaginatorModule, PageEvent } from
'@angular/material/paginator';
import { MatIconModule } from '@angular/material/icon';
import { MatButtonModule } from '@angular/material/button';

import Swal from 'sweetalert2';
import { ClienteService } from
'../../../shared/services/cliente/cliente-service';
import { Cliente } from '../../../shared/models/cliente';

// Se sua API ainda não retorna este formato, ajuste o service para
retornar { items, total }.
export interface PageResult<T> {
  items: T[];
  total: number;
}

@Component({
  selector: 'app-listagem-cliente',
  standalone: true,
  templateUrl: './listagem-cliente.html',
  styleUrls: ['./listagem-cliente.scss'],
  imports: [
    CommonModule,
    RouterLink,
    MatTableModule,
    MatPaginatorModule,
    MatIconModule,
    MatButtonModule,
  ],
})
export class ListagemCliente implements AfterViewInit {
  private clienteService = inject(ClienteService);

  // colunas exibidas na tabela
  displayedColumns: string[] = ['id', 'nome', 'cpf', 'email', 'status',
'funcoes'];
```

```
  // datasource do Material (poderia ser Cliente[]; usando
MatTableDataSource por compatibilidade com seu template)
  dataSource = new MatTableDataSource<Cliente>([]);

  // paginação (server-side)
  totalClientes = 0;
  pageIndex = 0; // zero-based para o paginator
  pageSize = 5;

  loading = signal(false);

  @ViewChild(MatPaginator) paginator!: MatPaginator;

  ngAfterViewInit(): void {
    this.listarClientes(this.pageIndex + 1, this.pageSize);
  }

  listarClientes(page: number, pageSize: number): void {
    this.loading.set(true);

    // Ajuste seu service para retornar Observable<PageResult<Cliente>>
    this.clienteService.listar_paginado(page, pageSize).subscribe({
      next: (res: PageResult<Cliente> | Cliente[]) => {
        // fallback: se a API ainda retornar apenas array, tenta
deduzir total
        if (Array.isArray(res)) {
          this.dataSource.data = res;
          // Caso não saiba o total, use um valor aproximado (ex.: page
* pageSize + 1)
          // Recomendo fortemente evoluir sua API para retornar {
items, total }
          this.totalClientes = this.pageIndex * this.pageSize +
res.length;
        } else {
          this.dataSource.data = res.items;
          this.totalClientes = res.total;
        }
        this.loading.set(false);
      },
      error: (err) => {
        console.error(err);
        this.loading.set(false);
        Swal.fire({
          icon: 'error',
          title: 'Erro',
          text: 'Não foi possível carregar a lista de clientes.',
        });
      },
    });
  }

  onPageChange(event: PageEvent): void {
    this.pageIndex = event.pageIndex;         // zero-based
    this.pageSize = event.pageSize;
    this.listarClientes(this.pageIndex + 1, this.pageSize); // sua API
usa 1-based
  }

  deletarCliente(id: number): void {
    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,
            });
            // Recarrega a página atual (mantendo página/size)
            this.listarClientes(this.pageIndex + 1, this.pageSize);
          },
          error: (error) => {
            console.error(error);
            Swal.fire({
              icon: 'error',
              title: 'Oops...',
              text: 'Erro ao deletar cliente!',
            });
          },
        });
      }
    });
  }

  // opcional: performance ao renderizar linhas
  trackById = (_: number, item: Cliente) => item.id;
}
```

- **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.

10. 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-routes.ts,** ele deverá ficar mais ou menos assim:

```
import { Routes } from '@angular/router';
import { CadastroCliente } from
'./pages/cliente/cadastro-cliente/cadastro-cliente';
import { ListagemCliente } from
'./pages/cliente/listagem-cliente/listagem-cliente';

export const routes: Routes = [{
    path: 'cliente',
    children: [
      {
        path: 'novo',
        component: CadastroCliente
      },
      {
        path: 'editar/:id',
        component: CadastroCliente
      },
      {
        path: '',
        component: ListagemCliente,
      },
    ]
  },
  {
    path: '',
    component: ListagemCliente,
  },
];
```

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

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

12. 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.ts** e ele deve ficar mais ou menos assim:

```
import { Component } from '@angular/core';

import { FormBuilder, FormControl, FormGroup, FormsModule,
ReactiveFormsModule, Validators } from '@angular/forms';

import { ActivatedRoute, Router } from '@angular/router';

import { ClienteService } from
'../../../shared/services/cliente/cliente-service';

import { Cliente } from '../../../shared/models/cliente';

import Swal from 'sweetalert2';

import { MatFormFieldModule } from '@angular/material/form-field';

import { MatRadioModule } from '@angular/material/radio';

import { CommonModule } from '@angular/common';

import {MatInputModule} from '@angular/material/input';



@Component({

  selector: 'app-cadastro-cliente',

  imports: [MatInputModule, MatFormFieldModule, MatRadioModule,
CommonModule, FormsModule, ReactiveFormsModule],

  templateUrl: './cadastro-cliente.html',

  styleUrl: './cadastro-cliente.scss'

})

export class CadastroCliente {

  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!',
          });
        }
      });
    }
  }
}
```

- 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.

13. 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.

14. 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