Cadastro de Clientes - *create*

1. Acesse o **cliente.service.ts,** adicione as linhas abaixo do **getClientes()**.

```
cadastrarCliente(cliente: Cliente):Observable<Cliente[]>{
   return this._httpClient.post<Cliente[]>(this.url, cliente);
 }
```

cliente.service.ts U

src > app > cliente.service.ts > ...

```
1   import { HttpClient } from '@angular/common/http';
2   import { Injectable } from '@angular/core';
3   import { Observable } from 'rxjs';
4   import { Cliente } from './models/Cliente.model';
5
6   @Injectable({
7     providedIn: 'root'
8   })
9   export class ClienteService {
10
11    url: string = "http://localhost:3000/clientes";
12
13    constructor(private _httpClient:HttpClient) { }
14
15    getClientes(): Observable<Cliente[]>{
16      return this._httpClient.get<Cliente[]>(this.url);
17    }
18
19    cadastrarCliente(cliente: Cliente):Observable<Cliente[]>{
20      return this._httpClient.post<Cliente[]>(this.url, cliente);
21    }
22
23  }
```

Explicando o código

```
cadastrarCliente(cliente: Cliente):Observable<Cliente[]>{
   return this._httpClient.post<Cliente[]>(this.url, cliente);
 }
```

O **cadastrarCliente()** é o nome do método que terá a função de ativar o POST do HTTP, da API para fazer o cadastro.

Continuamos mantendo o tipo do método como **Observable** convertido para o Tipo do **Modelo Cliente.**

Precisamos colocar o parâmetro cliente do tipo **Cliente**, pois precisamos saber quais são os dados que devemos enviar para a API, que no caso esses dados estarão salvas dentro de um objeto do tipo Cliente.

```
   return this._httpClient.post<Cliente[]>(this.url, cliente);
```

Estamos utilizando a variável **httpCliente** novamente, agora para acessar o método post que já está pronto. Precisamos fazer um *casting* (conversão de tipo) igual ao **Observable <Cliente[]>**.

Precisamos colocar dois parâmetros no método post:

- O endereço da API que está no atributo **this.url;**
- E o cliente que será o objeto enviado para API, que está vindo do parâmetro **cadastrarCliente()** que criamos.

2. Acesse o arquivo **cadastro-create.component.ts** e faça as importações. Elas serão as mesmas do **lista-clientes**.

```
import { Component } from '@angular/core';
import { Cliente } from '../models/Cliente.model';
import { ClienteService } from '../cliente.service';
import { Router } from '@angular/router';
```

3. Adicione dentro das chaves de **CadastrarClienteComponent{}** o código a seguir.

```
public cliente: Cliente = new Cliente(0,"","");

  constructor(private _clienteService:ClienteService, private
_router: Router){}

  cadastrar():void{
    this._clienteService.cadastrarCliente(this.cliente).subscribe(
      cliente => {
        this.cliente = new Cliente(0,"","");
        alert("Cadastro Efetuado com sucesso");
      },
      err => {
        alert("Erro ao Cadastrar");
      }
    );

    this._router.navigate(["/listar"]);
  }
```

Explicando o código

```
public cliente: Cliente = new Cliente(0,"","");
```

O atributo cliente foi criado para salvar os valor que virá do formulário em HTML.

O **new Cliente (0,"","")** possui um valor 0 e em branco aspas quando é do tipo string, pois como informado antes, quando precisamos criar um objeto do tipo cliente, obrigatóriamente precisamos inicializar os valores do objeto Cliente, mas esses valores serão substituídos, no caso os valores 0 e em branco, foi adicionado somente para o nosso algoritmo funcionar.

```
  constructor(private _clienteService:ClienteService, private
_router: Router){}
```

Esse construtor tem a injeção de dependência do ClienteService para fazer a comunicação com a API de Clientes. E do Router vai servir para conseguirmos redirecionar a página dentro da função em Typescript.

```
  cadastrar():void{
```

Método qua vamos chamar quando o usuário apertar o botão cadastrar no HTML do componente.

```
this._clienteService.cadastrarCliente(this.cliente)
```

Estamos utilizando o objeto de serviço **_clienteService**, para acessar o nosso serviço de comunicação com a API. Vamos chamar o método **cadastrarCliente()** que criamos no **ClienteService**. E ele terá como parâmetro o atributo cliente do nosso componente, por isso **this.cliente**.

```
cliente => {
  this.cliente = new Cliente(0,"","");
  alert("Cadastro Efetuado com sucesso");
```

Cliente é uma variável que existe somente dentro da função cadastrar(), que receberá uma função de salvar um objeto.

this.cliente vai receber um objeto cliente mas será substituído pelos valores que estão preenchidos no formulário de cadastro.

O **alert** será acionado caso o cadastro funcione.

```
err => {
  alert("Erro ao Cadastrar");
```

Caso a variável cliente não funcione será acionado a variável **err,** com o alert de erro. Se aparecer esse erro, é devido a ter problemas com a API ou com o ClienteService

```
this._router.navigate(["/listar"]);
```

Estamos utilizando o **_router** que criamos no construtor, para usar o método navigate, quando o usuário conseguir fazer o cadastro, ele será redirecionado para o link de rota **Lista**.

Agora vamos para o **cadastrar-cliente.component.ts.**

TS cadastrar-cliente.component.ts U

src > app > cadastrar-cliente > TS cadastrar-cliente.component.ts > ...

```ts
import { Component } from '@angular/core';
import { Cliente } from '../models/Cliente.model';
import { ClienteService } from '../cliente.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-cadastrar-cliente',
  templateUrl: './cadastrar-cliente.component.html',
  styleUrls: ['./cadastrar-cliente.component.css']
})
export class CadastrarClienteComponent {

  public cliente: Cliente = new Cliente(0,"","");

  constructor(private _clienteService:ClienteService, private _router: Router){}

  cadastrar():void{
    this._clienteService.cadastrarCliente(this.cliente).subscribe(
      cliente => {
        this.cliente = new Cliente(0,"","");
        alert("Cadastro Efetuado com sucesso");
      },
      err => {
        alert("Erro ao Cadastrar");
      }
    );

    this._router.navigate(["/listar"]);
  }
}
```

4. No arquivo **cadastrar-cliente.component.html**, substitua o código pelo que está abaixo.

```
<h2>Cadastrar Novo Cliente</h2>
<form class="row g-3">
    <div class="">
      <label for="nome" >Nome</label>
      <input type="text" class="form-control" id="nome" name="nome"
[(ngModel)]="cliente.nome">
    </div>
    <div class="">
      <label for="endereco">Endereço</label>
      <input type="text" class="form-control" id="endereco"
name="endereco" [(ngModel)]="cliente.endereco">
    </div>
    <div class="col-auto">
      <button type="submit" class="btn btn-primary mb-3"
(click)="cadastrar()">Cadastrar</button>
    </div>
  </form>
```

Explicando o código

```
[(ngModel)]="cliente.nome">
    </div>
    <div class="">
      <label for="endereco">Endereço</label>
      <input type="text" class="form-control" id="endereco"
name="endereco" [(ngModel)]="cliente.endereco">
```

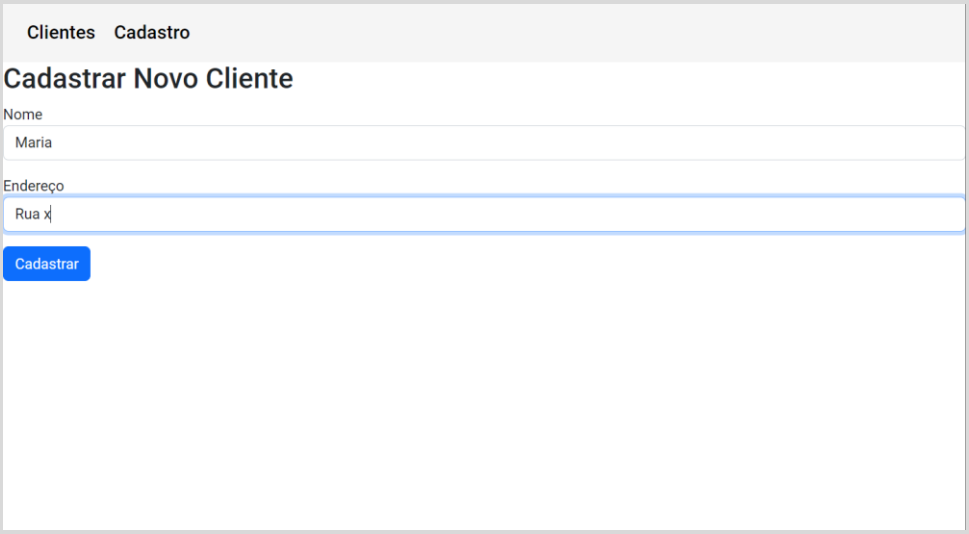
Em **[(ngModel)]**, estamos passando os dados desse formulário para os atributos criados na **model** Cliente e instanciados no **cadastrar-cliente.component.ts**, como o nome cliente.

Então, podemos acessar cada um dos atributos objeto cliente, e o que será digitado no input, **[(ngModel)]="cliente.id"**, será salvo no atributo **id** do **Model Cliente**, e a mesma coisa acontece com **cliente.nome** e **cliente.endereco**.

```
      <button type="submit" class="btn btn-primary mb-3"
(click)="cadastrar()">Cadastrar</button>
```

Já a função **click** realizará algum evento por meio da interação do clientes, que no caso é o método **cadastrar()**, ativará a função de cadastro.

5. Salve o código (**ctrl + s**) e veja o resultado no navegador. Clique no menu **Cadastro** e preencha o nome e endereço e pressione **Cadastrar**.



Ao testar no browser, seremos redirecionados para a página de lista, e verifique se apareceu o cliente cadastrado.