

Atualizar Cliente - *update*

Nessa etapa, precisamos adicionar dois métodos. Para isso, iremos inserir os valores no formulário de edição do cliente para a função de editar de acordo com ID do cliente.

1. Acesse o **cliente.service.ts** e atualize os dados da API com os valores do formulário abaixo do **cadastrarCliente()**:

```
getCliente(id:any):Observable<Cliente[]>{  
  const urlListarUm = `${this.url}?id=${id}`;  
  return this._httpClient.get<Cliente[]>(urlListarUm);  
}  
  
atualizarCliente(id: any, cliente: Cliente):Observable<Cliente[]>{  
  const urlAtualizar = `${this.url}/${id}`;  
  return this._httpClient.put<Cliente[]>(urlAtualizar, cliente);  
}
```

Explicando o código

```
getCliente(id:any):Observable<Cliente[]>{
```

O **getCliente()** terá a função de retornar um único registro da API.

Precisamos de um **id** do cliente, pois precisamos saber quem deve ser carregado para ser exibido no formulário de cadastro. Esse id é tipo **any** (qualquer) para evitar problemas caso o **id** receba um valor do tipo **number** ou **string**. Nesse caso, ele vai aceitar os dois tipos.

```
const urlListarUm = `${this.url}?id=${id}`;
```

O **const** (constante) serve para garantir que o valor da **urlListarUm** não possa ser modificado.

A segunda parte da linha, **`\${this.url}?id=\${id}`** é uma concatenação (junção de textos ou variáveis). É necessário adicionar um **id** na url para buscar somente um exemplo (no Postman, foi realizado esse mesmo processo).

O **this.url** é igual ao link **http://localhost:3000/clientes**.

A **id** é o valor de id existente, 1, por exemplo.

Com o **?**, podemos criar um parâmetro com o nome de **id**, que é uma variável na url do nosso site.

Vamos precisar capturar o valor desse parâmetro posteriormente. Nossa **urlListarUm** vai ficar **http://localhost:3000/clientes?id=1**.

```
return this._httpClient.get<Cliente[]>(urlListarUm);
```

Precisamos novamente utilizar o método **get()** convertido em **<Cliente[]>**, igual nosso **Observable** do **_httpClient**, porém agora com o endereço e id específico do registro. No caso, vamos colocar como parâmetro o **urlListarUm**.

```
atualizarCliente(id: any, cliente: Cliente):Observable<Cliente[]>{
```

Em **atualizarCliente()**, precisamos de dois parâmetros: um **id** do cliente que será retornado do formulário e um objeto do tipo **Cliente** que será salvo no **Model Cliente** quando apertar o botão **Atualizar** no HTML do componente.

```
const urlAtualizar = `${this.url}/${id}`;  
return this._httpClient.put<Cliente[]>(urlAtualizar, cliente);
```

Precisamos modificar o valor da nossa URL, acrescentando um valor id dos registro da nossa API. Em **`\${this.url}/\${id}`**; não vamos utilizar o **?id**, mas sim digitar um valor diretamente, pois vamos precisar utilizar um comando do Angular para capturar o seu valor e precisamos saber o nome do parâmetro.

No caso do atualizar, vai ser igual ao teste no Postman, sendo necessário somente colocar um número de **id**.

Então, no final da url, o **urlAtualizar** vai ficar **http://localhost:3000/1**.

```
return this._httpClient.put<Cliente[]>(urlAtualizar, cliente);
```

Essa linha chama a dependência do **HttpClient** e utiliza o método **put()** convertido em **<Cliente[]>**, igual ao **Observable**. Como parâmetro, ele utiliza a **urlAtualizar**. Acrescentamos um número de id na URL mais um objeto do tipo **Cliente** que precisar ser enviado para a API.

2. Verifique se o **cliente.service.ts** está com o código destacado abaixo.

```

TS cliente.service.ts U ●
src > app > TS 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    getCliente(id:any):Observable<Cliente[]>{
24      const urlListarUm = `${this.url}?id=${id}`;
25      return this._httpClient.get<Cliente[]>(urlListarUm);
26    }
27
28    atualizarCliente(id: any, cliente: Cliente):Observable<Cliente[]>{
29      const urlAtualizar = `${this.url}/${id}`;
30      return this._httpClient.put<Cliente[]>(urlAtualizar, cliente);
31    }
32
33  }
34

```

3. Acesse o **atualizar-cliente.component.ts** e adicione as importações abaixo.

```

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

```

Explicando o código

OnInit: Responsável por executar o que será executado quando iniciamos um componente.

ActivatedRoute: Responsável por conseguir capturar o valor de parâmetro da URL.

Dentro das aspas do **AtualizarClienteComponent{}**, adicione o código abaixo.

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

constructor(private _clienteService: ClienteService, private _router: Router,
private _activatedRoute: ActivatedRoute){
    this._activatedRoute.params.subscribe(params => this.clienteId =
params['id']);
}
ngOnInit(): void {
    this.listarCliente();
}

listarCliente():void{
    this._clienteService.getClientes(this.clienteId).subscribe(
        (res: any) => {
            this.cliente = new Cliente(
                res[0].id,
                res[0].nome,
                res[0].endereco
            );
        }
    )
}

atualizar(id: number){
    this._clienteService.atualizarCliente(id, this.cliente).subscribe(
        cliente => {this.cliente = new Cliente(0, "", "")},
        err => {alert("Erro ao atualizar")}
    );

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

Explicando o código

```
public clienteId: number = 0;
```

Atributo responsável por capturar o **id** do cliente registrado na API.

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

Atributo responsável por armazenar um objeto **Cliente**.

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

No construtor, foi adicionado uma injeção de dependência, **_activatedRoute** para capturar o valor da URL quando apertarmos o botão **Editar** no componente do HTML.

```
this._activatedRoute.params.subscribe(params => this.clienteId  
= params['id']);
```

O **_activatedRoute** serve para utilizar o atributo **params** da nossa dependência. Já o **this.clienteId** será usado para salvar o parâmetro da URL. O **params['id']** terá o nome do parâmetro capturado pelo método **getCliente()** do **ClienteService** na variável **urlListarUM**.

```
ngOnInit(): void {  
  this.listarCliente();  
}
```

Essa parte do código irá executar o **listarCliente()** assim que o **atualizar.component.ts** carregar.

```
listarCliente():void{  
  this._clienteService.getCliente(this.clienteId).subscribe(  
    (res: any) => {  
      this.cliente = new Cliente(  
        res[0].id,  
        res[0].nome,  
        res[0].endereco  
      );  
    }  
  );  
}
```

O **listarCliente()** está no singular com o propósito de exibir somente um resultado.

Utilizando o **_clienteService.getCliente()** do **ClienteService** e também dentro dele, vamos utilizar o **clienteId** no qual será armazenado o parâmetro **ID** capturado na URL.

Precisamos do **Subscribe** para utilizar os valores da API.

```
atualizar(id: number){}
```

Esse é o método que vai enviar o **id** e o objeto do tipo **cliente** para a API.

```
this._clienteService.atualizarCliente(id,this.cliente).subscribe(  
  cliente => {this.cliente = new Cliente(0,"","")},
```

Nessa parte do código, acessamos o **_clienteService**, que é nossa dependência e chamamos o método **atualizarCliente()**. Como parâmetro, enviamos o **id** e o atributo **this.cliente** que criamos para armazenar os valores do registro da API.

O código **cliente => {this.cliente = new Cliente(0,"","")}**, irá armazenar o objeto salvo no **this.cliente** para o **ClienteService**.

```
err => {alert("Erro ao atualizar")}
```

Se não funcionar, a API ou o **ClienteService** está com problemas e retorna um **alert**.

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

Essa linha usa a dependência de **Router**, acessando o método **navigate()**. Quando for acionado o **atualizar()**, no botão do HTML do componente, seremos redirecionados para o link **Listar** da nossas rotas.

4. Acesse o **atualizar-cliente.component.html** e adicione as importações abaixo.

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

Explicando o código

```
<form class="row g-3" *ngIf="cliente">
```

Se por acaso acessarmos a tela **Atualizar** e o atributo **cliente** não for instanciado, os clientes que estão nos **[(ngModel)]** não irão aparecer caso não existam. Caso eles existam, todos os dados irão aparecer.

```
<button type="submit" class="btn btn-primary mb-3"
(click)="atualizar(cliente.id)">Atualizar</button>
```

Clicando no botão **Atualizar**, será chamado o método **atualizar()**, criado no **atualizar-cliente.component.ts**. O **cliente.id** vai ter o valor de ID do registro escolhido.

5. Acesse o **listar-cliente.component.html** e adicione as importações abaixo.

```
<table class="table">
  <thead>
    <tr>
      <th scope="col">Id</th>
      <th scope="col">Nome</th>
      <th scope="col">Endereço</th>
      <th scope="col">Editar</th>
      <th scope="col">Excluir</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let cliente of clientes">
      <th scope="row">{{cliente.id}}</th>
      <td>{{cliente.nome}}</td>
      <td>{{cliente.endereco}}</td>
      <td><a mat-raised-button color="primary"
routerLink=" ../editar/{{cliente.id}}">Editar</a></td>
      <td><a mat-raised-button color="warn">Excluir</a></td>
    </tr>
  </tbody>
</table>
```

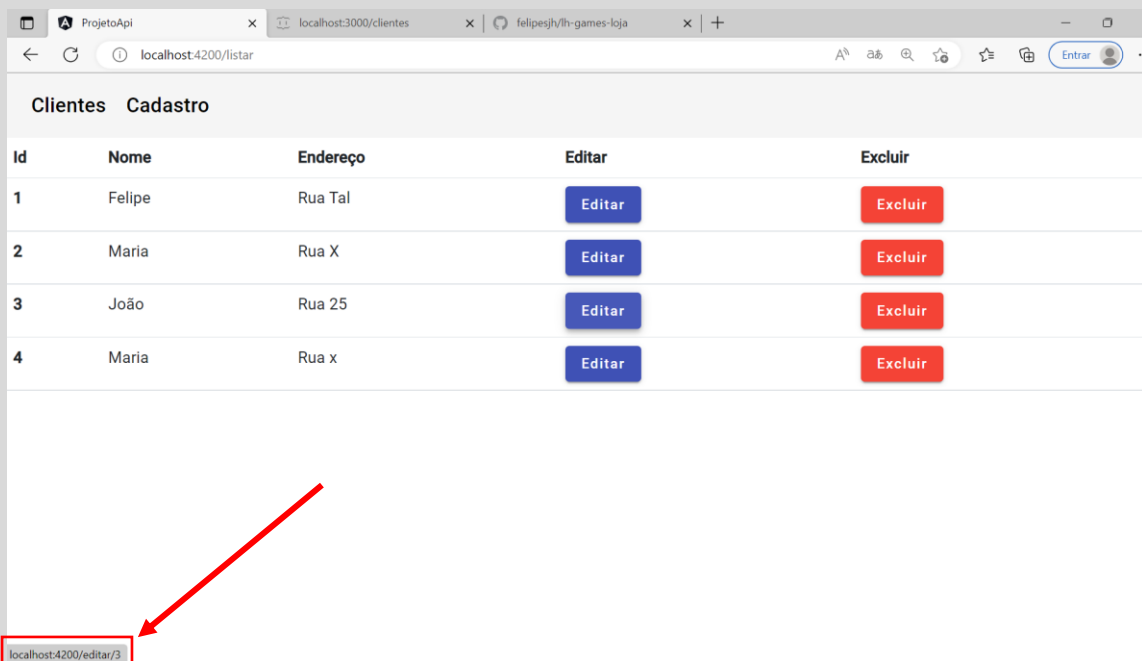
Explicando o código

```
<td><a mat-raised-button color="primary"
routerLink=" ../editar/{{cliente.id}}">Editar</a></td>
```

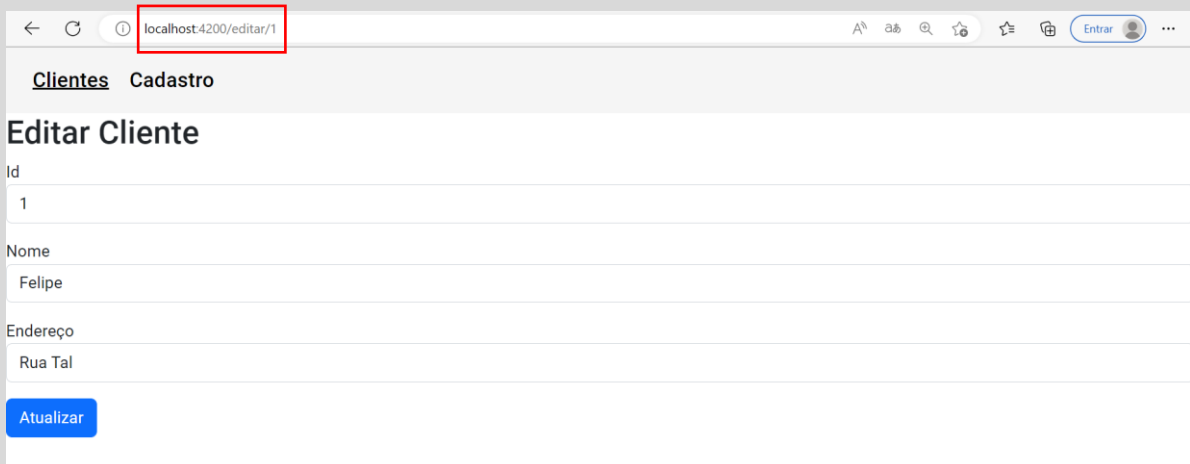
Como já estamos exibindo os valores da API, podemos aproveitar e utilizar o ID do registro que está salvo em **cliente.id**. Iremos exibi-lo na URL, que em nosso exemplo, ficará **../editar/1**.

Testando a aplicação

1. Passe o mouse em cima do botão **Editar** e verifique se os números são exibidos na barra de status no inferior da tela de acordo com os IDs dos registros.



2. Clique no botão **Editar** de qualquer registro e verifique **se** o id está aparecendo na URL, como é mostrado na tela abaixo.



← ↻ ⓘ localhost:4200/editar/1

Clientes Cadastro

Editar Cliente

Id
1

Nome
Felipe

Endereço
Rua Tal

Atualizar

3. Com os registros preenchidos no formulário, clique em **Atualizar**.



Clientes Cadastro

Editar Cliente

Id
1

Nome
Felipe Santos

Endereço
Rua X

Atualizar

Importante

O campo **ID** está bloqueado pois não deverá ser modificado.



4. Confira se os dados da página **Listar** foram alterados para os valores do registro de acordo com o que você preencheu.

Clientes Cadastro				
Id	Nome	Endereço	Editar	Excluir
1	Felipe Santos	Rua X	<button>Editar</button>	<button>Excluir</button>
2	Maria	Rua X	<button>Editar</button>	<button>Excluir</button>
3	João	Rua 25	<button>Editar</button>	<button>Excluir</button>
4	Maria	Rua x	<button>Editar</button>	<button>Excluir</button>

5. Em **localhost:3000**, confira se os valores também foram modificados. Realize a mesma checagem no arquivo **db.json**.

```

[
  {
    "id": 1,
    "nome": "Felipe Santos",
    "endereço": "Rua X"
  },
  {
    "id": 2,
    "nome": "Maria",
    "endereço": "Rua X"
  },
  {
    "id": 3,
    "nome": "João",
    "endereço": "Rua 25"
  },
  {
    "id": 4,
    "nome": "Maria",
    "endereço": "Rua x"
  }
]

```

```

() db.json U x
() db.json > [ ] clientes
1
2 {
3   "clientes": [
4     {
5       "id": 1,
6       "nome": "Felipe Santos",
7       "endereço": "Rua x"
8     },
9     {
10      "id": 2,
11      "nome": "Maria",
12      "endereço": "Rua x"
13    },
14    {
15      "id": 3,
16      "nome": "João",
17      "endereço": "Rua 25"
18    },
19    {
20      "id": 4,
21      "nome": "Maria",
22      "endereço": "Rua x"
23    }
24  ]
25 }

```