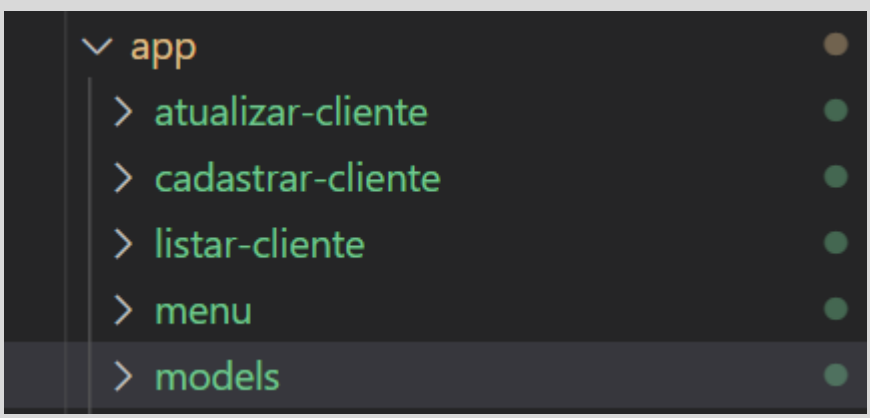
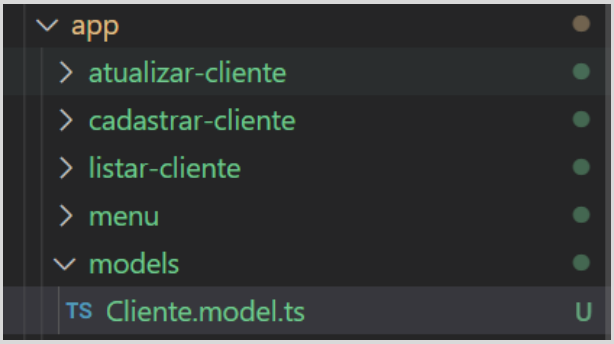
Consulta de dados de Cliente – *read*

Agora, estudaremos um exemplo de como criar uma requisição para consulta de dados (*READ*), que será utilizado para listar o clientes.

1. Vamos criar uma pasta com o nome de **models** na pasta **app** do projeto.



2. Dentro da pasta **models,** crie um arquivo **Cliente.model.ts**.



No arquivo **Cliente.model.ts**, será criado uma classe Cliente, que vai representar um modelo de cliente que será salvo e consumido pela API, as informações serão utilizadas por um objeto do tipo Cliente.

3. No **Cliente.model.ts**, adicione os comandos.

```
export class Cliente{
    id:number;
    nome:string;
    endereco:string;

    constructor(id:number, nome:string, endereco:string){
        this.id = id;
        this.nome = nome;
        this.endereco = endereco;
    }
}
```

Temos três atributos que serão os mesmos da API: **id**, **nome** e **endereço**. O construtor será responsável por inicializar um objeto do tipo Clientes.

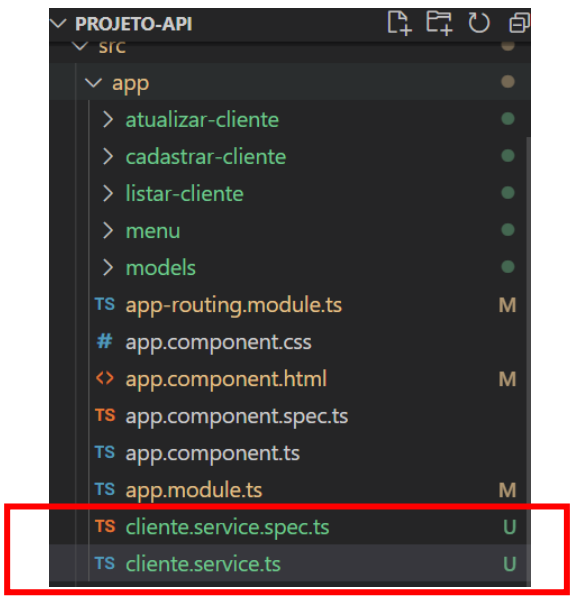
4. Vamos criar um **arquivo service**, que será responsável por se comunicar com a API. O arquivo se chamará cliente, acesse o **Terminal** do VS Code (Command Prompt), digite:

```
ng generate service cliente
```

Dica!

O comando ng generate service cliente pode ser digitado também como: **ng g s cliente**, como na imagem abaixo.

Serão criados dois arquivos, **cliente.service.spec.ts** e **cliente.service.ts**. Utilizaremos somente o arquivo **cliente.service.ts.**



5. No arquivo **cliente.service.ts,** adicione essas importações na parte de cima do arquivo.

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

HttpCliente: Responsável por buscar dados de uma fonte externa, que no caso será o db.json
Injectable: Responsável pelo funcionamento do Service.
Observable: Trabalha junto com HttpCliente, ele recebe e lida com os dados de forma assíncrona.
Cliente: O Modelo criado para guardar e enviar os dados da API.

6. Digite o código a seguir, logo abaixo da linha **export class ClienteService{}.**

```
url: string = "http://localhost:3000/clientes";

  constructor(private _httpClient:HttpClient) { }

  getClientes(): Observable<Cliente[]>{
    return this._httpClient.get<Cliente[]>(this.url);
  }
```

Com essas alterações, o código ficará assim:

```
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 }
```

Explicando o código

```
url: string = "http://localhost:3000/clientes";
```

O atributo url vai receber o link da URL da API, no nosso caso **http:localhost:3000/clientes.**

Se sua API estivesse utilizando outra porta, como a 3001, a porta do link também precisa ser alterada, como no exemplo abaixo:
http:localhost:3001/clientes

```
constructor(private _httpClient:HttpClient) { }
```

Aqui é realizada uma injeção de dependência *(Dependence Injectable*) da Classe HttpClient, que tem uma variável **_httpClient** do tipo **HttpClient**.

A injeção de dependência é uma técnica na qual um objeto fornece as dependências de outro objeto. Uma dependência é um objeto que pode ser usado, ou seja, poderemos usar todas as funções que existem no objeto httpClient, pois dependemos dela para fazer o CRUD com a API.

```
getClientes(): Observable<Cliente[]>{
  return this._httpClient.get<Cliente[]>(this.url);
}
```

O método **getClientes()** foi criado para ter a função de listar os clientes da API.

O tipo **Observable** vai verificar os dados da API, enquanto o **<Cliente[]>** converte os dados da API para um vetor de objetos do tipo Cliente.

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

O **this._httpclient**, é a variável criada no construtor. Ele possui um método pronto chamado **get()**. Em HTTP, o **GET** é responsável por recuperar os dados da API.

Precisamos fazer a conversão do **get,** utilizando **<Cliente[]>,** pois ele precisa ser igual à conversão do **Observable**.

This.url dentro do **get<Cliente[]>()** é o caminho que foi definido no atributo url **`http://localhost:3000/clientes`**

7. Acesse **listar-cliente.componente.ts** e adicione as importações na parte de cima do arquivo:

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

Explicando o código

```
import { Router } from '@angular/router';
```

O **Router** é uma classe que terá a responsabilidade de fazer o redirecionamento de navegação do conteúdo do projeto.

8. Dentro das chaves de **export class ListarClienteComponent {},** adicione o código abaixo.

```
public clientes: Cliente[] = [];

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

  ngOnInit(): void {
    this.listarClientes();
  }

  listarClientes():void{
    this._clienteService.getClientes().subscribe(
      retornaCliente =>{
        this.clientes = retornaCliente.map(
          item => {
            return new Cliente(
              item.id,
              item.nome,
              item.endereco
            );
          }
        )
      }
    )
  }
```

Explicando o código

```
public clientes: Cliente[] = [];
```

Atributo criado para receber todos os valores da API cliente. Isso ocorre por estamos utilizando um vetor **[]** para atribuir vários valores da API.

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

Essas são outras injeções de dependência:

Uma injeção funciona para o serviço que criamos **Cliente service**, que utilizaremos o **getClientes()** para listar nossos clientes.

A outra injeção serve para fazer redirecionamento de páginas, mas será utilizado somente quando precisarmos excluir um registro de cliente.

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

O **ngOnInit()** tem a função de executar tudo que tiver dentro dele quando o componente for carregado.

Nesse caso, ele vai executar o **listarCliente()** assim que nosso componente for carregado.

```
  listarClientes():void{
    this._clienteService.getClientes().subscribe(
      retornaCliente =>{
        this.clientes = retornaCliente.map(
```

O método **listarClientes()** foi criado para utilização e recebimento de valores do serviço de Cliente(**ClienteService**).

O **this._clienteService** é a variável do construtor. Vamos utilizar o **getClientes()** do **ClienteService** que criamos.

O **subscribe()** é necessário para receber as notificações do **Observable** do **ClienteService**.

O **retornaCliente** é uma variável criada para receber uma função anônima ou **arrow function** (=>), na qual irá receber as listas de valores da API. Já o **map()** tem a função de mapear todos os itens existentes dentro de um vetor, array ou matriz.

Saiba mais

Para conhecer sobre Arrow Function, acesse o site https://www.w3schools.com/js/js_arrow_function.asp

```
          item => {
            return new Cliente(
              item.id,
              item.nome,
              item.endereco
```

A variável **item** tem a função de criar objetos do tipo Cliente de acordo com a quantidade de itens que existirem na API.

Será necessário preencher os parâmetros de **Cliente()**, pois em **model Clientes** foi definido que precisamos obrigatoriamente criar um objeto **Cliente** com 3 parâmetros, colocá-lo dentro em **this.clientes.**

```
export class Cliente{
    id:number;
    nome:string;
    endereco:string;

    constructor(id:number, nome:string, endereco:string){
        this.id = id;
        this.nome = nome;
        this.endereco = endereco;
    }
}
```

Será criado um objeto para cada “item” existente na API e será realizado passando por **item** -> **retornaCliente.map** -> **this.clientes** -> **retornaCliente**.

TS listar-cliente.component.ts U ●

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

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

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

  public clientes: Cliente[] = [];

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

  ngOnInit(): void {
    this.listarClientes();
  }

  listarClientes():void{
    this._clienteService.getClientes().subscribe(
      retornaCliente =>{
        this.clientes = retornaCliente.map(
          item => {
            return new Cliente(
              item.id,
              item.nome,
              item.endereco
            );
          }
        )
      }
    )
  }

}
```

9. Acesse **listar-cliente.componente.html**.

Substitua as linhas de HTML dentro das tags **<TBODY> </TBODY>,** conforme abaixo.

```
<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>
```

Explicando o código

```
<tr *ngFor="let cliente of clientes">
```

***ngFor** é uma diretiva do Angular que tem a função de repetir um conteúdo dentro de um critério.

Let **cliente** é uma variável criada e poderia ter qualquer nome.

Clientes é o atributo que criamos no arquivo **listar-cliente.componente.ts.**

Dessa forma, o cliente que criamos vai acessar o atributo clientes que tem a lista de clientes, linha por linha. O **ngFor** vai repetir a **tag <tr>** até acabar a lista de clientes.

Temos a variável **cliente**, que virou um objeto do tipo Cliente, graças ao **ngFor**. Podemos acessar seus atributos **cliente.id**, **cliente.nome** e **cliente.endereço.**

Para exibir os valores no HTML, precisamos utilizar a Interpolação de Texto (*Text Interpolation*) do Angular, **utilizando chaves de chaves {{}**. Exemplo: **{{cliente.id}}**

Por fim, sua aplicação terá a aparência abaixo.