

PROGRAMAÇÃO WEBII

1

Listar Produtos, Interface Produtos

Services, Conexão HTTP, Observable e
Listagem de Produtos - GET

O que iremos aprender?

- Vamos aprender como criar services e assim interagir com o nosso backend.
- Vamos fazer os CRUD's – que seria criar produtos, editar os produtos, listar os produtos que temos lá no nosso backend que foi criado no começo de nossas aulas.

2

Services

Conceito e exemplo prático

Relembrando...

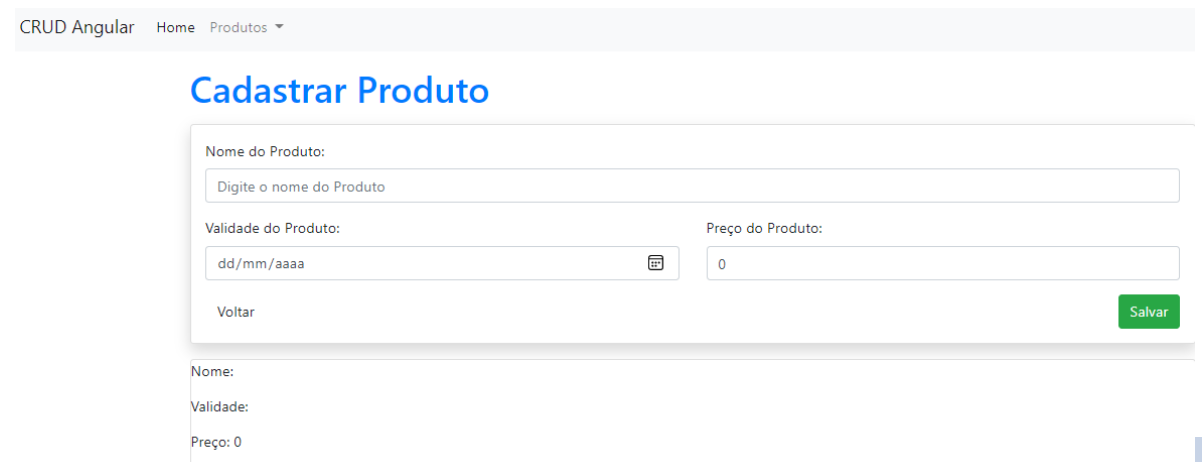
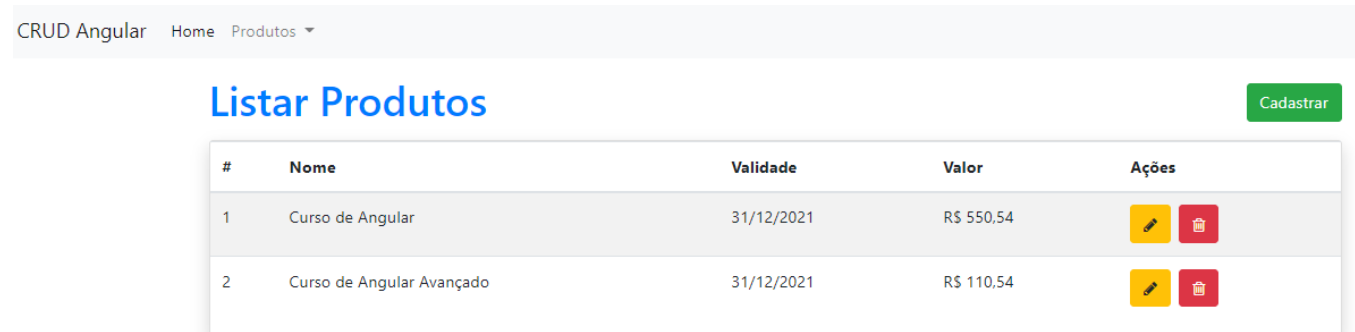
- Considere sua aplicação angular como um edifício. Um edifício pode ter N vários apartamentos nele. Um apartamento é considerado como um **módulo**. Um apartamento pode ter um N número de salas que correspondem aos blocos de construção de um aplicativo angular chamado **componentes**.
- Agora, cada apartamento (**Módulo**) terá salas (**Componentes**), elevadores (**Serviços**) para permitir maior movimentação dentro e fora dos apartamentos, fios (Tubos) para mover informações e torná-las úteis nos apartamentos.
- Você também terá lugares como piscina, quadra de tênis que estão sendo compartilhados por todos os moradores do prédio. Portanto, eles podem ser considerados **componentes no SharedModule**.

Por que utilizamos serviços no Angular?

- A utilização de Serviços tem o propósito de organizar o projeto de software Angular, isolando lógica de negócio e separando-a dos Controllers.
- Não é possível afirmar que seja obrigatório utilizar serviços, mas é muito desejável..
- Na prática não há diferença para o usuário porque, provavelmente, utilizar serviços não afetará diretamente o comportamento da interface. Assim, os benefícios ficam por conta da melhor organização do projeto e da Engenharia de Software para o projeto que está sendo desenvolvido.
- Um serviço é uma classe que pode ser utilizada por outros componentes do projeto.

Componentes que criamos

Componentes:



O que teremos de serviço?

Serviços:

Listar Produtos:

- ▶ Buscar todos
- ▶ Buscar por id

Cadastrar

Atualizar

Excluir




Exibir erro

Exibir Mensagem

CRUD Angular Home Produtos ▾

Listar Produtos

Cadastrar

#	Nome	Validade	Valor	Ações
1	Curso de Angular	31/12/2021	R\$ 550,54	 
2	Curso de Angular Avançado	31/12/2021	R\$ 110,54	 

CRUD Angular Home Produtos ▾

Cadastrar Produto

Nome do Produto:

Validade do Produto:

Preço do Produto:

[Voltar](#)

[Salvar](#)

Nome:

Validade:

Preço: 0

Criando serviços

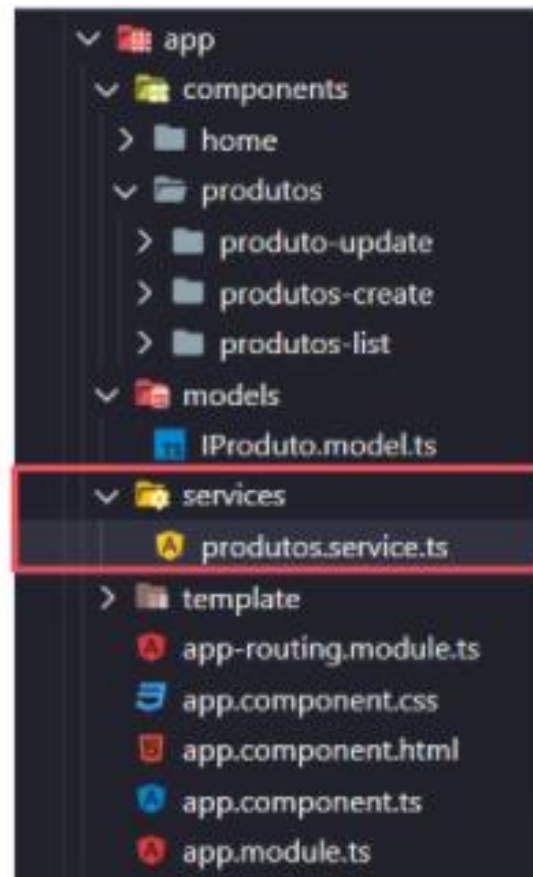
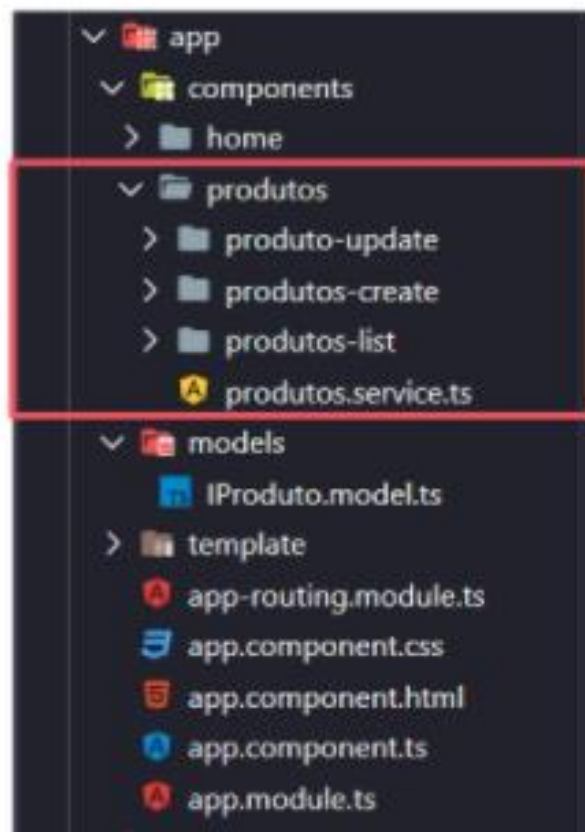
- Como criar um serviço no Angular?

```
ng generate service <diretório>/<nome>
```

Organização dos serviços



Geralmente em cursos e tutoriais os serviços são criados junto com a pasta que agrupa os componentes de CRUD.



Eu organizo em uma pasta exclusiva, pois acredito que facilita a separação das responsabilidades.



Escolha a metodologia de organização que mais se identificar!

Criando serviços

- Nós temos a pasta produtos, que está agrupando nossos componentes do produto, nossas views do produto e o serviço estaria aqui.
- Nós, gostamos de organizar em uma pasta exclusiva.
- Quando coloco em uma pasta exclusiva eu separo melhor essa responsabilidade, deixo de uma forma mais organizada, eu encontro mais fácil os meus serviços e eu não misturo a parte de view da minha parte que estará conversando com o backend, que estará tratando os dados da minha aplicação. Aqui faremos dessa maneira, mas vocês podem escolher a metodologia que achar mais interessante.

Criando serviços

- O service não é obrigatório no Angular, mas é uma boa prática. O Angular não nos obriga a utilizar service.
- Quando nós trabalhamos com componentes, temos que pensar que os nossos componentes tem que ser o mais simples possível. Por isso, teremos o serviço para fazer todo o tratamento de dados, seja buscando a informação no backend, buscando um arquivo que você tem dentro do seu projeto: você poderia ter um arquivo de carga de dados dentro do seu próprio projeto, você poderia estar buscando APIs de outras empresas, de outros lugares e todo mundo estará sendo controlado. Toda a gestão de dados que você vai ter fará utilizando um serviço.

Criando serviços

- Vamos para o nosso projeto para criarmos ele.
- Vamos acessar a pasta do projeto
 - ▷ Ir para o prompt de comando
 - ▷ Ir para a pasta do projeto
 - ▷ `cd\angular\cursoPWEBII\frontend <ENTER>`

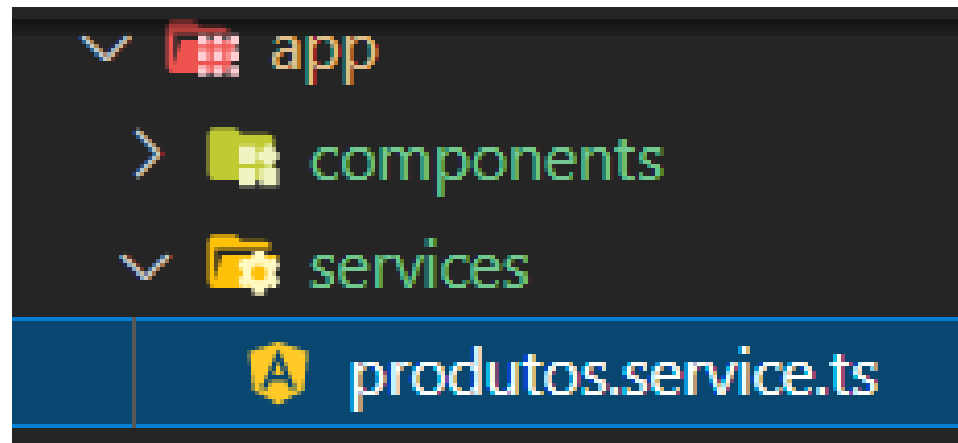
Criando serviços

- No prompt, vamos digitar:

- ▶ `ng g s services/produtos <ENTER>`

```
C:\angular\cursopwebII\frontend>ng g s services/produtos  
CREATE src/app/services/produtos.service.ts (137 bytes)
```

- Dentro de `src/app/services` eu vou ter o meu `produtos.service`



Configurando serviços

- Vamos abrir o arquivo e verificar o que temos nele. Basicamente é bem simples. O serviço, quando criamos ele é basicamente uma classe que vai ter o injectable que vai facilitar a integração do nosso serviço com o resto da nossa aplicação.
- Novamente, um serviço é uma classe. A diferença está na anotação **@Injectable**. É ela que instrui o Angular a considerar a classe ProdutosService como um serviço, pois é, não há uma anotação com nome parecido com @Service() =.
- O objetivo é adicionar funcionalidades nesse serviço:
 - ▶ Listar Produtos;
 - ▶ Cadastrar produtos;
 - ▶ Atualizar Produtos;
 - ▶ Excluir Produtos;
- Agora precisamos começar a elaborar ele.

Configurando serviços

- Vamos começar criando a variável que conterà a URL de onde ele irá buscar as informações do nosso produto.
- Vamos escrever `private` já que essa URL ficará privada dentro da minha aplicação.
- Vamos escrever URL maiúsculo, já que ela será uma constante aqui dentro. Vocês podem ver que já está deixando até marcado . O tipo dela será `string` e será igual a <http://localhost:3000> que é o nosso servidor que estará rodando na porta 3000 e lá dentro vamos mexer com a nossa tabela de produtos, por isso complementamos com `/produtos`.

```
6  export class ProdutosService {  
7      private URL: string = 'http://localhost:3000/produtos';  
8  
9      constructor() {}  
10 }
```


Configurando serviços

- O próximo passo iremos começar a fazer o nosso método. Vou criar o método que será buscarTodos(), porque irei buscar todos os produtos que tenho lá.
- Criamos o nosso método, mas eu não passei o tipo aqui, nós iremos colocar ai conforme formos criando nosso método para ir entendendo o que vai acontecer em cada parte aqui.

```
6  export class ProdutosService {  
7      private URL: string = 'http://localhost:3000/produtos';  
8  
9      constructor() {}  
10  
11     buscarTodos(){  
12  
13     }  
14 }
```

3

Conexões http

Configurando...

Configurando serviços

- Para fazer requisições HTTP, estamos mexendo com protocolos HTTP, precisamos importar o modulo do Angular que é responsável por tratar as requisições HTTP que é chamado de HttpClient.
- Para isso, precisaremos ir no arquivo app.module.ts. Dentro do imports, teremos que fazer o import do HttpClient, mas ele não será importado automaticamente, por isso faremos a importação manual:
- Vamos digitar, após o último import e digitar:
 - `import from '@angular/common/http',` `import from '@angular/common/http';`
- Após o import, vou passar o descontrutor, que serão as minhas chaves. Vou digitar http e já será exibido o HttpClient, mas não queremos o HttpClient simplesmente, queremos o HttpClientModule. Só clicar nele e estará pronto.

Configurando serviços

Ficará assim:

```
import {HttpClientModule} from '@angular/common/http';
```

Após fazer a importação do HttpClientModule no início, já podemos selecionar o HttpClientModule, copiar e colar no imports, conforme abaixo para que ele fique disponível dentro de toda a nossa aplicação

```
32  imports: [  
33      BrowserModule,  
34      AppRoutingModule,  
35      FormsModule,  
36      SharedModule,  
37      HttpClientModule  
38  ],
```

Configurando serviços

- Salve o arquivo
- Agora já podemos rodar nossa aplicação. Lembrando que todas as vezes que alterarmos informações no `app.module`, precisaremos parar a aplicação e executar novamente.
 - ▷ `ng serve -o` <ENTER>

Configurando serviços

■ Para poder utilizar o http, vamos acessar o arquivo produtos.service.ts e dentro do construtor, vamos digitar:

- ▶ private http: HttpClient (Ao começar a digitar HttpC, já irá trazer a opção para você selecionar.
- ▶ HttpClientModule é somente no app.module.ts, aqui é o HttpClient.

```
7  export class ProdutosService {  
8      private URL: string = 'http://localhost:3000/produtos';  
9  
10     constructor(private http: HttpClient) {}  
11  
12     buscarTodos(){  
13
```

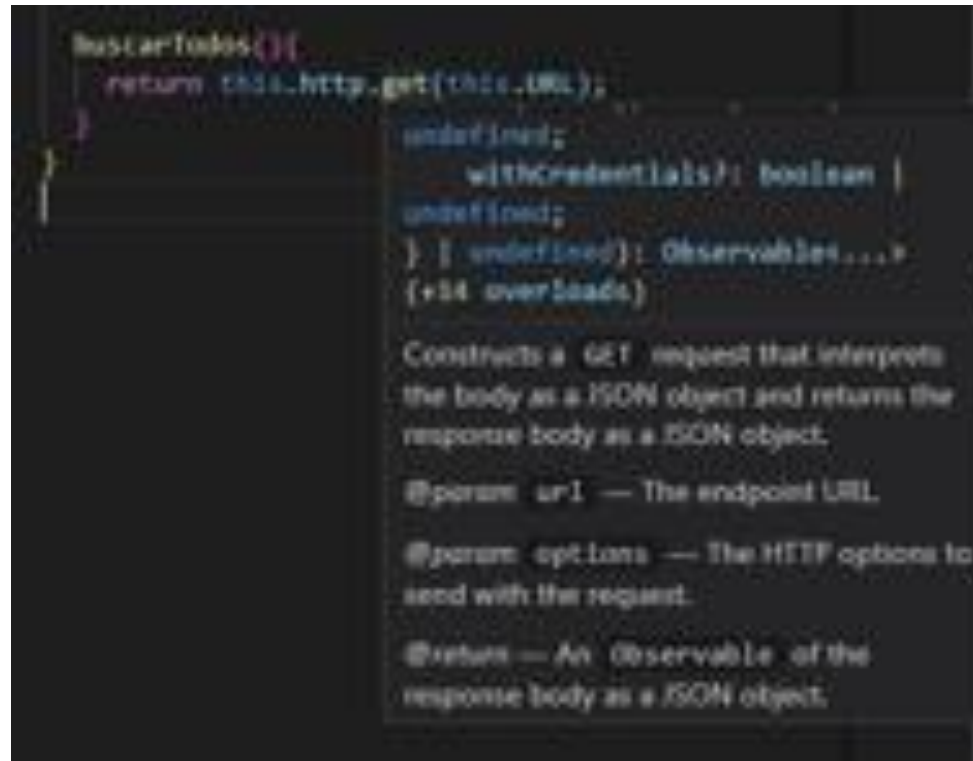
Configurando serviços

- Já fizemos a criação da variável que irá fazer a auto instanciação do meu objeto HttpClient no momento do nosso serviço e agora já podemos ir no método buscarTodos() e vamos codar o que precisamos.
- No método buscarTodos(), vamos falar que ele irá retornar a variável que criamos no construtor e o método que iremos utilizar para buscar os dados dos nossos produtos, que será o método get. Dentro do método get, o que preciso passar de parâmetro? Precisamos enviar a url (this.url)
 - ▶ `return this.http.get(this.URL);`

```
12     buscarTodos(){  
13         |   return this.http.get(this.URL);  
14     }  
15 }
```

Configurando serviços

- Está simples aqui, mas podemos fazer algumas melhorias.
- Quando utilizamos o get do httpClient, vamos ter um Observable. Só passar o mouse em cima do get, descer com o cursor do mouse e será exibido:



4

Observable

Configurando...

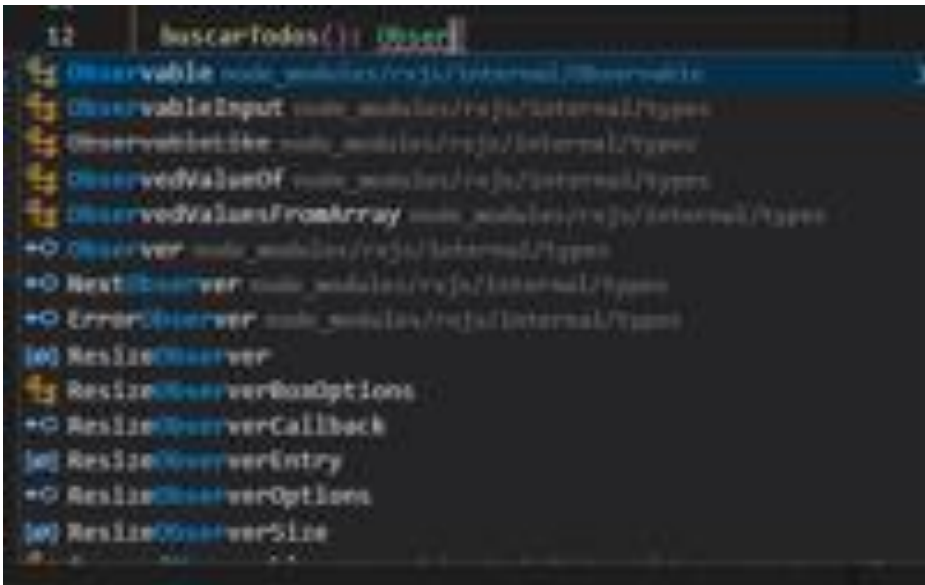
Configurando serviços

- O que é o Observable?
- Acessar o site:
 - ▶ <https://github.com/Reactive-Extensions/RxJs> <ENTER>
- Faz parte da biblioteca RxJs da Microsoft. É uma biblioteca para trabalhar com Javascript para programação reativa. A programação reativa é uma programação orientada a evento. Cada vez que um evento acontecer aqui, ela será acionada. E qual o papel do Observable? Ele ficará observando o nosso método e cada vez que acontecer uma determinada situação ele irá avisar todo mundo que essa situação aconteceu.
- Onde temos isso no nosso dia a dia?
- Quando entramos em uma loja e temos aquela campainha com sensor de presença. Quando alguém passa pela porta da loja, a loja faz um barulho, a campainha faz um barulho para que o dono da loja saiba que alguém entrou dentro da loja.
- O Observable trabalha mais ou menos dessa maneira.

Configurando serviços

- Vamos colocar aqui então, que o nosso método buscarTodos() vai retornar um Observable
- Após o buscarTodos(), vamos digitar dois pontos e começar a digitar Observ. Automaticamente irá importar.

buscarTodos(): Observable



```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4
5 @Injectable({
6   providedIn: 'root',
7 })
8 export class ProdutosService {
9   private URL: string = 'http://localhost:3000/pro
10
11   constructor(private http: HttpClient) {}
12
13   buscarTodos(): Observable{
14     return this.http.get(this.URL);
15   }
16 }
```

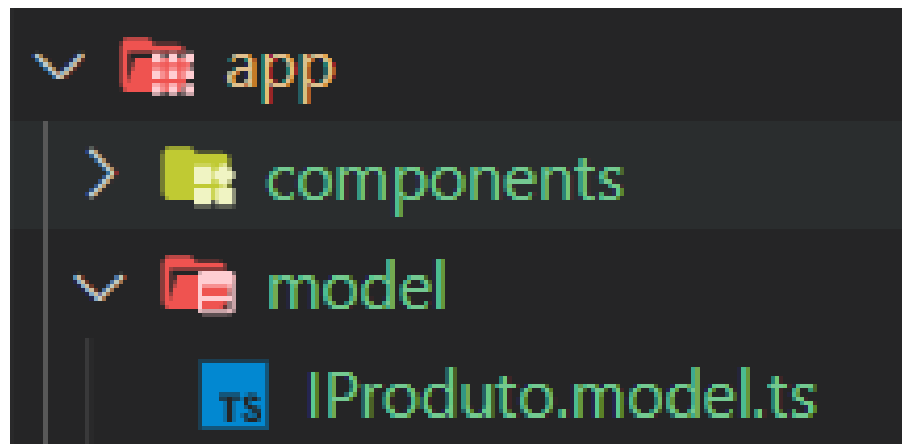
Configurando serviços

■ Só que o Observable fica esperando que tipo de informação será retornada. Então, poderia colocar aqui uma string, poderia colocar que seria um any (que poderia retornar qualquer coisa), mas nós iremos retornar aqui um produto. Vamos deixar como any:

```
buscarTodos(): Observable<any> {  
  | return this.http.get(this.URL);  
}
```

Configurando serviços

- Para facilitar para o typescript, que ele entenda, que reconheça como será a estrutura do objeto que será retornado para nós, nós vamos criar aqui, dentro do nosso app, botão direito, new folder e vamos escrever o nome da pasta: model. Nós vamos criar um modelo, uma interface modelo, para que o typescript saiba como será a estrutura do objeto que nós estaremos recebendo.
- Dentro de model, vamos clicar com o botão direito, new file e vamos criar o arquivo: IProduto.model.ts -> I de Interface,



5

Interface


Configurando...

O que Listar produtos e Cadastrar Produtos tem em comum?

CRUD Angular Home Produtos ▾

Listar Produtos

Cadastrar

#	Nome	Validade	Valor	Ações
1	Curso de Angular	31/12/2021	R\$ 550,54	 
2	Curso de Angular Avançado	31/12/2021	R\$ 110,54	 

CRUD Angular Home Produtos ▾

Cadastrar Produto

Nome do Produto:

Validade do Produto:



Preço do Produto:

[Voltar](#)

[Salvar](#)

Nome:

Validade:

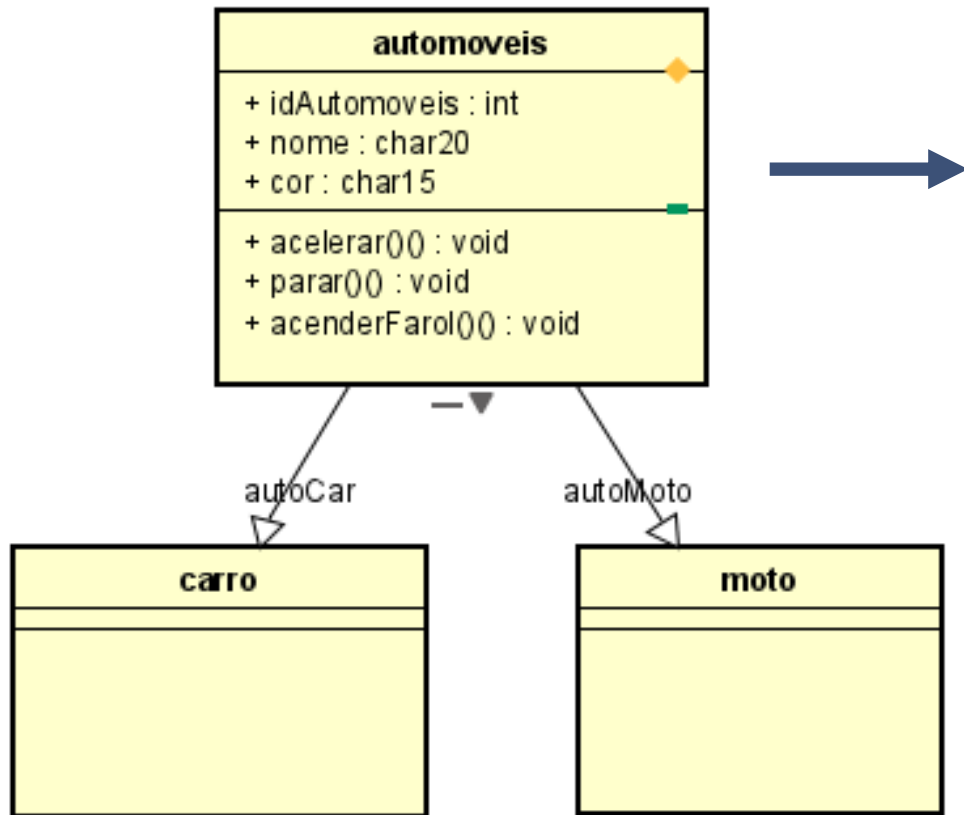
Preço: 0

Criaremos uma interface

Interface

- Uma interface funciona como um contrato entre si e qualquer classe ou estrutura que implemente sua interface.
- Interfaces não possuem a implementação de métodos pois apenas declaram o conjunto de métodos, o comportamento que uma ou um conjunto de classes deve ter.
- Na interface, todos os métodos são portanto abstratos e públicos, já que são apenas declarados na interface sendo obrigatoriamente implementados pelas classes que implementam a interface.
- **Interfaces** são muito importantes pois, nos permitem separar o “**o que**” do “**como**”. A interface não se preocupa com a forma com a qual o método está sendo implementado e sim, que este método estará disponível a todos os objetos que a implementarem.

Interface



Interface

OBS: Uma interface não pode ser herdada por uma classe, mas sim implementada. No entanto, uma interface pode herdar de outra interface, criando uma hierarquia de interfaces.

Configurando serviços

■ Dentro do arquivo IProduto.model.ts, vamos escrever:

▶ `export interface IProduto{ }`

■ O que é uma interface?

▶ Ela trabalha como se fosse um contrato, Uma vez que nós informamos que a variável é do tipo dessa interface, o typescript irá obrigar que nós passemos todos os parâmetros que foram criados para a elaboração do nosso objeto.

```
src > app > model > ts IProduto.model.ts > ...  
1   export interface IProduto{  
2  
3   }
```

Configurando serviços

- Sendo assim, iremos colocar todos os parâmetros dentro da nossa interface.
 - ▶ Na validade já vamos utilizar a interface Date. A interface Date já irá trazer um contrato de como trabalhar com datas e horários. Ficará assim:

```
1  export interface IProduto{  
2      id: number;  
3      nome: string;  
4      validade: Date;  
5      precoProduto: number;  
6  }
```

Configurando serviços

- Criamos a nossa interface. Agora, toda vez que criarmos uma variável e ela for do tipo IProduto, eu vou ser obrigado a criar um objeto que tenha: id, nome, validade e preço.
- Vamos retornar agora no arquivo produtos.service.ts e vamos alterar a variável do Observable de any para Ipro (no momento que começar a escrever, já irá trazer a interface de Iproduto e já realizará a importação).
- Só que não iremos trazer apenas um produto, iremos trazer um array de produtos. Por isso, devemos inserir os colchetes após o Iproduto[]

```
14     buscarTodos(): Observable<IProduto[]> {  
15         | return this.http.get(this.URL);  
16     }
```

Configurando serviços

- Dentro do get, ficou vermelhinho, com erro? Por que?
- Porque está esperando que passemos para ele, que tipo de variável iremos receber aqui, então nós devemos passar que será IProduto antes dos parênteses, mostrando que iremos receber um array.

```
return this.http.get<IProduto[]>(this.URL);
```

```
14     buscarTodos(): Observable<IProduto[]> {  
15         | return this.http.get<IProduto[]>(this.URL);  
16     }
```

Configurando serviços

- Criamos o nosso serviço, já estamos com a nossa interface, qual será o próximo passo agora?
- Vamos no nosso projeto backend
 - Vamos abrir uma nova janela no prompt de comando
 - Vamos até a pasta backend
 - Vamos abrir com o visual code

```
C:\angular>cd\angular\cursopwebII\backend  
C:\angular\cursopwebII\backend>code .
```

Configurando serviços

Dentro do arquivo db.json temos:

- ▶ posts
- ▶ comments
- ▶ profile

Vamos apagar tudo o que temos do nosso objeto, deixando
Somente as chaves {}

```
1  {  
2  
3  }
```

```
1  {  
2    "posts": [  
3      {  
4        "id": 1,  
5        "title": "json-server",  
6        "author": "typicode"  
7      }  
8    ],  
9    "comments": [  
10     {  
11       "id": 1,  
12       "body": "some comment",  
13       "postId": 1  
14     }  
15   ],  
16   "profile": {  
17     "name": "typicode"  
18   }  
19 }
```

Configurando serviços

- Como estamos criando, nosso banco de dados de produtos, vamos escrever entre aspas “produtos”, que será um array e terá um objeto inicial {}, onde iremos passar um id, nome, validade e preço.
- Sendo assim, criamos no db.json nossa tabela de produtos para que possamos interagir com nossos produtos no nosso banco de dados, nosso backend

```
1  {
2    "produtos": [
3      {
4        "id": 1,
5        "nome": "Curso de Angular",
6        "validade": "2021-12-31",
7        "precoProduto": 550.54
8      }
9    ]
10  }
```


Configurando serviços

■ Agora, vamos no prompt de comando do backend e vamos rodar:

▶ npm start <ENTER>

```
C:\angular\cursopwebII\backend>npm start

> backend@1.0.0 start C:\angular\cursopwebII\backend
> json-server --watch db.json

\{^_^}/ hi!

Loading db.json
Done

Resources
http://localhost:3000/produtos

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

Configurando serviços

■ O servidor subiu e podemos ver que temos um recurso que é <http://localhost:3000/produtos>. Esse mesmo recurso que temos aqui é o recurso que colocamos dentro do arquivo produtos.service.ts do nosso frontend, como url:

```
Resources  
http://localhost:3000/produtos
```

```
9   export class ProdutosService {  
10     private URL: string = 'http://localhost:3000/produtos';  
11  
12     constructor(private http: HttpClient) {}  
13  }
```

Configurando serviços

- Para podermos utilizar esses produtos, para ver esses produtos que foram retornados vamos dentro de components, produtos, listar-produtos, listar-produtos.component.ts
- Dentro do construtor, vamos escrever private produtosService: ProdutosService (Já irá exibir ProdutosService para fazer a auto importação).

```
constructor(private produtosService: ProdutosService) {  
  for (let item of this.listaStrings) {  
    console.log(item);  
  }  
}
```

```
1  import { ProdutosService } from '../services/produtos.service';  
2  import { Component, OnInit } from '@angular/core';
```

Configurando serviços

- Agora, já conseguimos utilizar esse produtoService
- Vamos criar um método aqui, para carregar os nossos produtos. Após o ngOnInit {}, digitar o método carregarProdutos(): que deverá retornar uma lista de produtos
 - ▶ carregarProdutos():
- Para isso, vamos apagar a ListaProdutos que criamos após o export:

```
9  export class ListarProdutosComponent implements OnInit {
10
11      listaStrings: string[] = ['Primeiro', 'Segundo', 'Terceiro'];
12      listaNumeros: number[] = [15, 15.18, 100];
13
14      objetoModelo = {
15          nome: 'Fatima',
16          idade: 18,
17          altura: 1.56,
18          graduado: true
19      };
10 }
```

Configurando serviços

Após o export, teremos os comandos abaixo. Nossa lista não será mais do tipo any, mas IProduto. Ao começar a digitar IProd, selecionar para importar automaticamente.

```
9 export class ListarProdutosComponent implements OnInit {  
10  
11   listaProdutos: any[] = [  
12     {nome: 'Curso de Angular', precoProduto: 35.56, validade: '2021-12-31', id: 1},  
13     {nome: 'Curso de Ionic', precoProduto: 50, validade: '2021-12-31', id: 2, promocao:  
14       {id: 3, nome: 'Curso de Ionic Avançado', precoProduto: 50, validade: '2021-12-31'}}  
15   ];  
16  
17
```

```
import { IProduto } from '../model/IProduto';  
import { ProdutosService } from '../service/ProdutosService';  
import { Component, OnInit } from '@angular/core';
```

```
listaProdutos: IProduto[] = [  
  {nome: 'Curso de Angular', precoProduto: 35.56, validade: '2021-12-31', id: 1},  
  {nome: 'Curso de Ionic', precoProduto: 50, validade: '2021-12-31', id: 2, promocao:  
    {id: 3, nome: 'Curso de Ionic Avançado', precoProduto: 50, validade: '2021-12-31'}}  
];
```

Configurando serviços

- Apresentou um erro no campo validade, dizendo que não existe, mas é porque o campo validade está como date no Iproduto.model.
- Mas.. nós vamos começar com o nosso array vazio, então podemos apagar as informações dentro desse array. Ele não dará mais erro aqui, porque o próprio campo vai receber a data corretamente aqui.

12

```
listaProdutos: IProduto[] = [];
```

Configurando serviços

- Agora, vamos apagar tudo o que está no construtor, já que não vamos mais utilizar isso.

```
10 export class ListarProdutosComponent implements OnInit {  
11  
12     listaProdutos: IProduto[] = [];  
13  
14  
15     constructor(private produtosService: ProdutosService) {  
16  
17     }
```

Configurando serviços

- Agora nós temos um `listar-produtos.component.ts`. Eu tenho um `listaProdutos` que irá receber um array de produto. Estou chamando aqui, estou injetando o meu `ProdutoServices` e criei um método `carregarProdutos()`.
- O método `carregarProdutos()` irá retornar para nós um `void`, abra e fecha chaves, ficando assim:

```
19      ngOnInit(): void {  
20      }  
21      carregarProdutos(): void{  
22      }  
23  }
```


Configurando serviços

- Dentro do método, vamos digitar:
 - ▶ `this.produtosService.buscarTodos().subscribe()`
- `subscribe` é o método que vai ser responsável por acionar o evento do Observable. Ele irá acionar o HttpClient para que vá até o backend e busque os dados.
- Ele vai ter um retorno, que iremos inserir dentro do método `subscribe`:

```
carregarProdutos(): void{  
    this.produtosService.buscarTodos().subscribe(retorno =>{  
  
    })  
}
```

Configurando serviços

- E esse retorno, nós iremos trabalhar com ele aqui e iremos falar:
 - ▶ `this.listaProdutos = retorno`

```
carregarProdutos(): void{  
  this.produtosService.buscarTodos().subscribe(retorno =>{  
    this.listaProdutos = retorno;  
  })  
}
```


Functions

```
let media = function(nota1:number, nota2:number, nota3:number): number {  
    return (nota1 + nota2 + nota3) / 3  
}
```

Arrow Functions

```
let media = function(nota1:number, nota2:number, nota3:number): number {  
    return (nota1 + nota2 + nota3) / 3  
}
```

```
let media = (nota1:number, nota2:number, nota3:number) => (nota1 + nota2  
+ nota3) / 3
```



As **arrow functions** permitem ter um retorno implícito, que são valores retornados sem ter que usar a palavra return. Uma expressão *arrow function* possui uma sintaxe mais curta quando comparada a uma expressão de função

Arrow Functions

```
// declaracao da arrow function
const getData = () => ({name: 'Steve', lastName: 'Jobs'})

// chama a funcao
console.log(getData())

// output: {name: "Steve", lastName: "Jobs"}
```

Para declarar um objeto, é preciso usar chaves e dentro definir as propriedades necessárias, que no nosso código são as propriedades `name` e `lastName`. O motor JavaScript confunde as chaves do objeto com as chaves do corpo da função e para desfazer essa confusão, você precisa usar parênteses em torno do objeto, conforme segue acima.

■ Maiores detalhes:

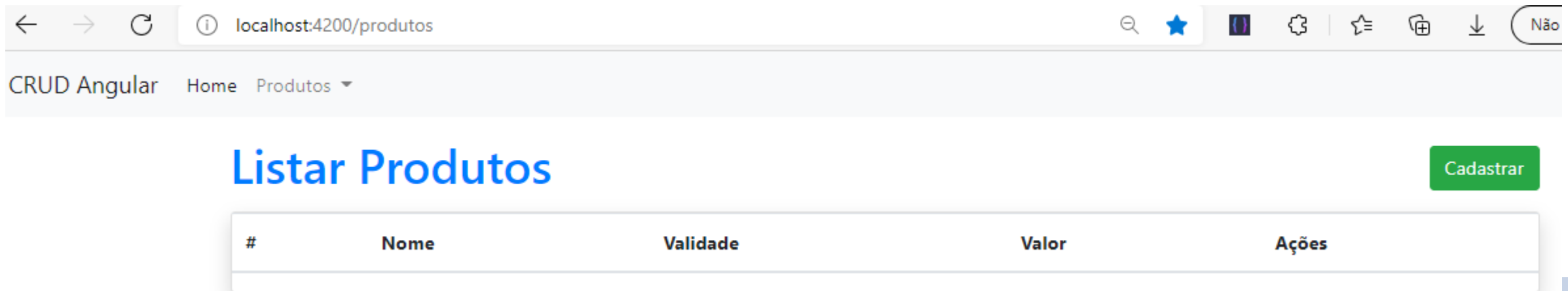
<https://developerplus.com.br/o-que-sao-arrow-functions-e-como-usar/>

Configurando serviços

- Agora está prontinho, o meu `carregarProdutos()` está indo buscar todos os produtos que estão no backend e estou me inscrevendo para receber uma resposta. O `subscribe` vai acionar o `HttpClient` que irá buscar os nossos produtos e o `Observable` vai ser responsável por aguardar essa resposta. Nossa mensagem assíncrona ela vai chegar quando estiver pronta, depois que navegar pela rede, retornar e quando retornar irá injetar esse retorno que estamos recebendo dentro da nossa lista de produtos.

Configurando serviços

- Vamos abrir o nosso navegador e ver o que aconteceu.
- Vamos em produtos / listar produtos
- Não exibiu nada.... Por que?



Configurando serviços

- Não trouxe nada porque eu não chamei o método, não acionamos o método `carregarProdutos()` em nenhum momento.
- É agora que vamos utilizar o `ngOnInit()`. Ele é uma interface que é implementada aqui que vai trabalhar com o ciclo de vida do nosso componente dentro do Angular.
- O `ngOnInit` quer dizer que quando o componente for inicializado é para executar alguma coisa, o que ele irá executar? Vamos chamar o método `carregarProdutos()`. Agora sim, quando o meu componente `listarProdutos` for carregado, quando for inicializado eu vou chamar o método `carregarProdutos()`

```
ngOnInit(): void {  
  | this.carregarProdutos();  
}
```




Configurando serviços

Vamos ver nossa página. Já está atualizada, temos o curso de Angular, validade, valor. Já temos o listar produtos funcionando através do nosso backend.

CRUD Angular Home Produtos ▾

Listar Produtos

Cadastrar

#	Nome	Validade	Valor	Ações
1	Curso de Angular	31/12/2021	R\$ 550,54	 

Estudando um pouco mais...

- <https://br.atsit.in/archives/35140>
- <https://developerplus.com.br/o-que-sao-arrow-functions-e-como-usar/>

