

PROGRAMAÇÃO WEBII

1

Cadastro de Produto

POST

O que iremos aprender?

- Já aprendemos como listar todos os produtos de nossa aplicação.
- Agora, vamos aprender como cadastrar o produto utilizando nossa API.
- Para isso, precisaremos deixar o prompt de comando rodando dentro do frontend para exibir o projeto do Angular
- Temos também que ter o prompt de comando no backend que estará servindo de servidor para os nossos dados.
- Abrindo nosso navegador, Produtos, listar produtos nós teremos nosso produto que está cadastrado no backend

Cadastrar Produto

- Para fazer o cadastrar, precisaremos abrir o arquivo produto.service.ts
- Após o método buscarTodos(), vamos criar o método cadastrar(): que será do tipo Observable, que retornará uma interface do produto <IProduto>

```
buscarTodos(): Observable<IProduto[]> {  
    return this.http.get<IProduto[]>(this.URL).pipe(  
        map(retorno => retorno),  
        catchError(erro => this.exibirErro(erro))  
    );  
}  
  
cadastrar(): Observable<IProduto> {  
}
```

Cadastrar Produto

- Para poder cadastrar um produto, preciso receber ele, então dentro do método cadastrar vou informar que receberei um produto do tipo IProduto.

```
cadastrar(produto: IProduto): Observable<IProduto> {  
}
```

Cadastrar Produto

- Agora nós temos que chamar o nosso método http. Vejam que já temos acima, um código com get, parecido com o que nós precisamos. Vamos copiar esse código e colar embaixo:

```
buscarTodos(): Observable<IProduto[]> {  
    return this.http.get<IProduto[]>(this.URL).pipe(  
        map(retorno => retorno),  
        catchError(erro => this.exibirErro(erro))  
    );  
}  
  
cadastrar(produto: IProduto): Observable<IProduto> {  
    return this.http.get<IProduto[]>(this.URL).pipe(  
        map(retorno => retorno),  
        catchError(erro => this.exibirErro(erro))  
    );  
}
```

Cadastrar Produto

- Só que para cadastrar produto, nós não iremos mais utilizar o método get. Nós iremos enviar dados, então iremos utilizar o método post. O método post irá receber apenas um produto, sendo assim, iremos retirar os colchetes do IProduto. Depois temos a nossa URL, mas ele precisa também receber o corpo, que é o nosso produto que ele irá cadastrar, por isso, vamos inserir uma virgula e escrever produto.
- O meu retorno, vou enviar ele caso esteja tudo certo. Caso esteja errado, nós vamos exibir a mensagem de erro para o usuário.

```
cadastar(produto: IProduto): Observable<IProduto> {  
    return this.http.post<IProduto>(this.URL, produto).pipe(  
        map(retorno => retorno),  
        catchError(erro => this.exibirErro(erro))  
    );  
}
```

Cadastrar Produto

- O meu serviço já está pronto. Tudo o que eu precisava fazer aqui já está ok. Só tivemos que criar o método cadastrar.
- Agora vamos para o cadastrar produto. Vamos abrir o arquivo cadastrar-produto.component.html
- Vamos deletar o último card que inserimos para testarmos o two way data binding, pois não iremos mais precisar dele.
- Vamos salvar

```
<div class="card mt-3">  
  <p>Nome: {{ nome }}</p>  
  <p>Validade: {{ validade }}</p>  
  <p>Preço: {{ preco }}</p>  
</div>
```


Cadastrar Produto

- Próximo passo é abrir o arquivo cadastrar-produto.component.ts
- Abaixo do export, temos as variáveis nome, validade e preco. Não vamos mais utilizar e iremos apagar.

```
export class CadastrarProdutoComponent implements OnInit {  
  nome: string = '';  
  validade: string = '';  
  preco: number = 0;  
}
```

```
export class CadastrarProdutoComponent implements OnInit {  
  
  constructor() {}  
  
  ngOnInit(): void {}  
  
  salvarProduto(): void {  
    console.log('Nome: ', this.nome);  
    console.log('Validade: ', this.validade);  
    console.log('Preço: ', this.preco);  
    alert('Salvo com sucesso!');  
  }  
}
```

Cadastrar Produto

- Vamos criar uma variável denominada produto, que será do tipo IProduto. Ao digitar IProduto, selecione para realizar a importação dela.
- Verifique que já foi importado a nossa interface:

```
import { IProduto } from '../../../model/IProduto.model';  
import { Component, OnInit } from '@angular/core';  
  
@Component({  
  selector: 'app-cadastrar-produto',  
  templateUrl: './cadastrar-produto.component.html',  
  styleUrls: ['./cadastrar-produto.component.css'],  
})  
export class CadastrarProdutoComponent implements OnInit {  
  
  produto: IProduto
```

Cadastrar Produto

- Após o IProduto, vamos digitar = e abrir chaves. Porque quando eu crio uma interface eu preciso passar os parâmetros que são obrigatórios.
- Precisamos passar o id? Não, porque estamos cadastrando um produto. O id virá da nossa API.
- Vamos abrir o arquivo de interface do produto: IProduto.model.ts e vamos informar que o id é opcional. Quando colocamos uma interrogação dentro da nossa interface, dentro das nossas classes, após o atributo, entende-se que é opcional, ou seja, ele não precisa existir.
- Vamos salvar
- Fechar o arquivo Iproduto.model.ts

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

Cadastrar Produto

Vamos voltar para o arquivo `cadastrar-produto.component.ts` e inserir os atributos: `nome: ''` (porque é string), `validade: new Date()` e `precoProduto: 0` e um ponto e vírgula após as chaves:

```
produto: IProduto = {  
  nome: '',  
  validade: new Date(),  
  precoProduto: 0  
};
```

Cadastrar Produto

■ Nosso html agora vai começar a dar problema, por que? Porque no html estávamos procurando uma variável denominada nome e agora precisamos colocar que está dentro produto.nome e veja que já sumiu o erro. Agora precisamos alterar todas as variáveis e incluir produto.

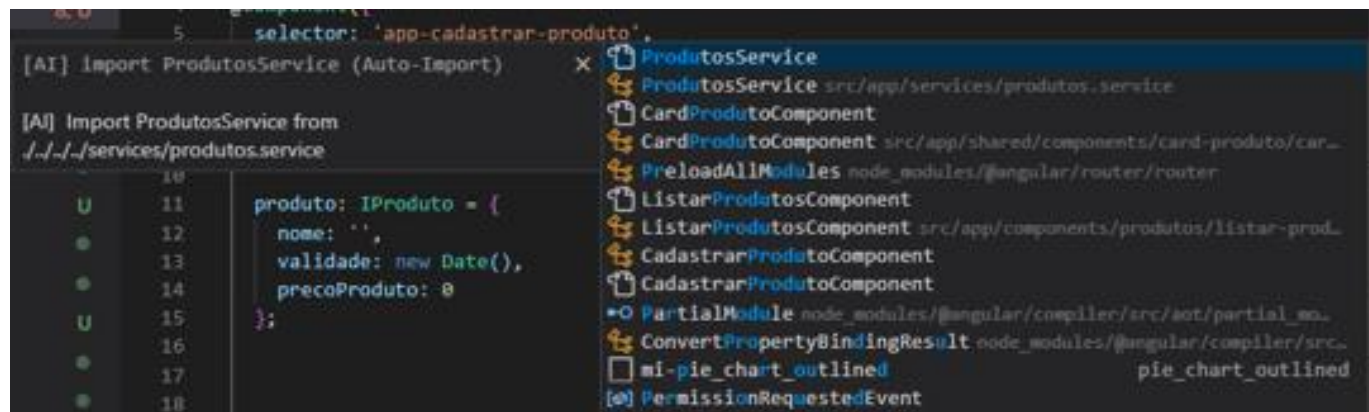
```
<input
  type="text"
  class="form-control"
  placeholder="Digite o nome do Produto"
  name="nomeProduto"
  [(ngModel)]="produto.nome"
```

```
<input
  type="date"
  class="form-control"
  name="validade"
  [(ngModel)]="produto.validade"
```

```
<input
  type="number"
  class="form-control"
  placeholder="0.00"
  name="preco"
  [(ngModel)]="produto.precoProduto"
/>
```

Cadastrar Produto

- Fazendo isso, tiramos o erro do nosso html e já podemos retornar para o TypeScript.
- Aqui nós temos o nosso método construtor, nós precisamos chamar o serviço que será responsável por salvar o produto na API, enviar nosso produto para a API. Para isso vamos digitar, dentro do construtor:
 - ▶ `private produtosService:ProdutosService` (no momento que colocarmos Produtos com letra maiúscula, ele irá trazer para fazer o auto import, vamos clicar e será criado no início da aplicação a importação dele



The screenshot shows a code editor with a TypeScript file. On the left, there's a snippet of code defining an interface `IProduto` with properties `nome`, `validade`, and `precoProduto`. On the right, an autocomplete dropdown is open, showing a list of imports. The first option is `ProdutosService` from `src/app/services/produtos.service`, which is highlighted. Other options include `CardProdutoComponent`, `PreloadAllModules`, `ListarProdutosComponent`, `CadastrarProdutoComponent`, `PartialModule`, `ConvertPropertyBindingResult`, `mi-pie-chart-outlined`, and `PermissionRequestedEvent`.

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

Cadastrar Produto

- Já podemos voltar para o método salvarProduto() e vamos apagar o que temos lá:

```
salvarProduto(): void {  
  
}  
}
```

Cadastrar Produto

Aqui vamos digitar:

- ▶ `this.produtosService.cadastrar()` e iremos passar o nosso produto dentro do parenteses?:
- ▶ `this.produtosService.cadastrar(this.produto).subscribe()` -> estamos nos inscrevendo para receber um retorno, estamos mandando ele enviar nosso pedido para o http.
- ▶ Dentro dos parenteses do `subscribe`, vamos pegar nosso retorno e iremos trabalhar com ele.
 - ▶ `this.produtosService.cadastrar(this.produto).subscribe(retorno => {`

```
salvarProduto(): void {  
    this.produtosService.cadastrar(this.produto).subscribe(retorno => {  
  
    });  
}
```

Não esquece do ponto e vírgula no final.

Cadastrar Produto

■ E agora podemos fazer a programação aqui:

- ▶ `this.produto = retorno;` -> retorno é o que iremos receber da nossa API
- ▶ Quando enviarmos o nosso produto para a API, estamos enviando apenas nome, validade e preço

```
produto: IProduto = {  
  nome: '',  
  validade: new Date(),  
  precoProduto: 0  
};
```

- ▶ Quando recebermos o retorno, teremos também o ID que esse produto recebeu dentro do banco de dados e estamos guardando dentro da nossa variável produto já com a ID agora.

Cadastrar Produto

- Continuando...
- Vamos chamar o produtosService
 - ▷ `this.produtosService.exibirMensagem();`
- Agora vamos montar uma mensagem para ser exibida. Vamos escrever:
 - ▷ 'Sistema', vamos montar um template string (lembrando, preciso utilizar crase)
 - ▷ ``${this.produto.nome} foi cadastrado com sucesso. ID ${this.produto.id},`
- Agora vamos colocar o tipo de erro:
 - ▷ 'toast-success'

Cadastrar Produto

- Ficará assim:
- Veja que já sumiu todos os erros que tínhamos aqui e agora já podemos acionar o nosso produto, já conseguiremos salvar o nosso produto.

```
salvarProduto(): void {  
  this.produtosService.cadastrar(this.produto).subscribe(retorno => {  
    this.produto = retorno;  
    this.produtosService.exibirMensagem(  
      'Sistema',  
      `${this.produto.nome} foi cadastrado com sucesso. ID: ${this.produto.id}`,  
      'toast-success'  
    );  
  });  
}
```

Cadastrar Produto

- Porém, antes é interessante que quando clicarmos no botão SALVAR e tenha sucesso, nos naveguemos o nosso usuário de volta para a lista de produtos.
- Para isso, no construtor, teremos que colocar uma virgula após ProdutosService e vamos digitar:
 - ▶ private router: Router (no momento que escrever a palavra Router já teremos o auto import e só precisaremos clicar nele

```
constructor(private produtosService: ProdutosService, private router: Router) {}
```

@angular/router' class Rout x Router node_modules/@angular/router/router

```
constructor(private produtosService: ProdutosService, private router: Router) {}
```

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

Cadastrar Produto

- O Router é o responsável por fazer o roteamento da nossa aplicação.
- Após o método salvarProduto, vamos chamar ele:
 - ▶ `this.router.navigate();`
- Dentro dos parênteses, precisamos passar um parâmetro, um array , onde o primeiro parâmetro que vamos passar é o caminho `'/produtos'`

```
▼ salvarProduto(): void {  
  ▼ this.produtosService.cadastrar(this.produto).subscribe(retorno => {  
    ▼ this.produto = retorno;  
    this.produtosService.exibirMensagem(  
      'Sistema',  
      `${this.produto.nome} foi cadastrado com sucesso. ID: ${this.produto.id}`,  
      'toast-success'  
    );  
    this.router.navigate(['/produtos']);  
  });  
}
```

Cadastrar Produto

- Conseguimos fazer a navegação e vamos ver se a nossa aplicação está funcionando.
- Vamos na página de Listar Produtos e clicar no botão Cadastrar. Inclua um curso novo e clique no botão Salvar

Cadastrar Produto

Nome do Produto:

Validade do Produto:

Preço do Produto:

Voltar


Salvar

Cadastrar Produto

Será exibida a mensagem que foi cadastrado com sucesso e será exibido o curso na tela:

Listar Produtos

[Cadastrar](#)

#	Nome	Validade	Valor	Ações
1	Curso de Angular	31/12/2021	R\$ 550,54	 
2	Curso de Ionic	31/12/2021	R\$ 253,12	 