05

Faça como eu fiz: Serviços

Vamos criar o arquivo produtos.service.ts para o nosso serviço, dentro dele exportamos uma classe ProdutosService com o seguinte código:

```
export class ProdutosService {

}
```

COPIAR CÓDIGO

Precisamos de um array que conterá a lista de produtos, para isso, criamos uma propriedade produtos.

```
private readonly produtos: Produto[] =
[
    new Produto("LIV01", "Livro TDD e B
    new Produto("LIV02", "Livro Inician
    new Produto("LIV03", "Inteligência
];
```

COPIAR CÓDIGO

Para usar a classe Produto em nosso serviço, precisamos importá-la.

```
import { Produto } from "./produto.mode
```

COPIAR CÓDIGO

Agora vamos criar o método obterTodos que irá simplesmente retornar o array de produtos.

```
obterTodos(): Produto[] {
    return this.produtos;
}
```

COPIAR CÓDIGO

Também vamos precisar do método obterUm que será responsável por retornar um produto específico.

```
obterUm(id: number): Produto {
    return this.produtos[0];
}
```

COPIAR CÓDIGO

Agora que já retornamos produtos, nosso serviço precisa de um método para criar produto.

```
criar(produto: Produto) {
    this.produtos.push(produto);
}
```

COPIAR CÓDIGO

Também precisaremos de um método para alterar um produto.

```
alterar(produto: Produto): Produto {
    return produto;
}
```

COPIAR CÓDIGO

Para finalizar, ainda falta o método para apagar um produto.

```
apagar(id: number) {
    this.produtos.pop();
}
```

COPIAR CÓDIGO

Legal, agora temos um serviço dedicado a cuidar dos dados, mas ele não está sendo utilizado, então, vamos refatorar o nosso controlador para que use o serviço que acabamos de criar.

O Nest utiliza o conceito de providers (provedores em português), que são classes que podem ser injetadas automaticamente em outras. Vamos aplicar este conceito para que o Nest injete o serviço em nosso controlador.

Para transformar uma classe em um provider, basta anotá-la com o decorator `@Injectable()`, esse decorator deve ser importado do pacote `@nestjs/common` com o seguinte código:

```
import { Injectable } from "@nestjs/com
```

```
@Injectable()
export class ProdutosService {
    // código omitido
}
```

COPIAR CÓDIGO

Feito isso, podemos iniciar a refatoração do controlador de produtos. Vamos começar criando um construtor que recebe por parâmetro o ProdutosServico. O Nest irá automaticamente instanciar um objeto ProdutosService e passá-la por parâmetro para o nosso construtor, é a magia da injeção de dependência acontecendo.

```
export class ProdutosController {

    constructor(private produtosService

    }

    // código omitido
}
```

COPIAR CÓDIGO

Podemos excluir o array de produtos.

```
produtos: Produto[] = [
    new Produto("LIV01", "Livro TDD e B
    new Produto("LIV02", "Livro Inician
    new Produto("LIV03", "Inteligência
];
```

COPIAR CÓDIGO

Em todos os métodos, vamos simplesmente chamar o método equivalente do nosso serviço.

```
@Get()
obterTodos(): Produto[] {
    return this.produtosService.obterTo
}

@Get(':id')
obterUm(@Param() params): Produto {
    return this.produtosService.obterUm
}

@Post()
criar(@Body() produto: Produto) {
```

```
    this.produtosService.criar(produto)
}

@Put()
alterar(@Body() produto: Produto): Prod
    return this.produtosService.alterar
}

@Delete(':id')
apagar(@Param() params) {
    this.produtosService.apagar(params.
}
```

COPIAR CÓDIGO

Pronto, agora só testar. Opsss tem um erro em nosso terminal:

```
[Nest] 11412   - 2020-09-02 22:22:59


Potential solutions:
- If ProdutosService is a provider, is
- If ProdutosService is exported from a
  @Module({
    imports: [ /* the Module containing
```

```
})
+16ms
```

COPIAR CÓDIGO

Repare que interessante, o Nest está nos dando a dica, faltou importar o nosso ProdutoService no módulo principal, então vamos alterar o arquivo app.module.ts e adicionar o ProdutosService no array de providers.

- Antes: providers: `[AppService],`
- Depois: providers: `[AppService, ProdutosService],`

Boa notícia, nenhum erro exibido em nosso terminal.