

NestJS - Introdução e Desenvolvimento de uma API

Criação de Projeto, Controller, Service e Repository

O que é NestJS?

Definição:

- Framework para construção de APIs robustas e escaláveis.
- Baseado no Node.js e TypeScript.
- Segue princípios de arquitetura limpa (clean architecture) e modular.

O que é NestJS?

Benefícios:

- Suporte nativo ao TypeScript.
- Modularidade e injeção de dependências.
- Documentação integrada com Swagger.

Módulos, serviços, controladores e repositórios

Modules, services, controllers, repositories

Módulos (Modules)

O que são?

Módulos agrupam funcionalidades relacionadas, organizando controladores, serviços, repositórios e outras dependências. Eles servem como contêineres que podem importar/exportar recursos entre si.

Exemplo de Definição

```
1 @Module({
2   imports: [], // Importa outros módulos
3   controllers: [ProductController], // Inclui controladores
4   providers: [ProductService, ProductRepository], // Inclui serviços e repositórios
5   exports: [ProductService], // Exporta recursos para outros módulos
6 })
7 export class ProductModule {}
```

Criação de Projeto

Instalando dependências:

- npm install @nestjs/swagger swagger-ui-express class-validator class-transformer

Controladores (Controllers)

O que são?

Os controladores recebem requisições HTTP e definem endpoints da API. Eles são responsáveis por invocar os serviços para processar essas requisições.

Controladores (Controllers)

Exemplo:

```
1 @Controller('products') // Rota base: /products
2 export class ProductController {
3   constructor(private readonly productService:
4     ProductService) {}
5
6   @Get() // Método GET: /products
7   findAll() {
8     return this.productService.findAll();
9   }
10
11   @Post() // Método POST: /products
12   create(@Body() product: any) {
13     return this.productService.create(product);
14   }
15 }
```

Serviços (Services)

O que são?

Os serviços implementam a lógica de negócio e gerenciam operações complexas. Eles interagem com repositórios para acessar ou manipular dados.

Função Principal:

- Servir como intermediário entre os controladores e os repositórios.
- Não lidam diretamente com a persistência dos dados.

Serviços (Services)

Exemplo:

```
1 @Injectable()
2 export class ProductService {
3   constructor(private readonly productRepository:
ProductRepository) {}
4
5   findAll() {
6     return this.productRepository.findAll(); // Busca todos os
produtos
7   }
8
9   create(product: any) {
10    return this.productRepository.create(product); // Cria um
novo produto
11  }
12 }
```

Repositórios (Repositories)

O que são?

Repositórios são responsáveis pelo acesso direto ao banco de dados.

Eles encapsulam as operações de CRUD e isolam a lógica de persistência.

Por que usar?

- Separa a lógica de banco de dados da lógica de negócio.
- Facilita a manutenção e a troca de tecnologias de persistência.

Repositórios (Repositories)

Exemplo com Sequelize:

```
1 @Injectable()
2 export class ProductRepository {
3   constructor(@InjectModel(Product) private readonly
productModel: typeof Product) {}
4
5   findAll() {
6     return this.productModel.findAll(); // Busca todos os
registros
7   }
8
9   create(product: any) {
10    return this.productModel.create(product); // Cria um novo
registro
11  }
12 }
```

Como se relacionam?

Controller → Service → Repository

- O **Controller** recebe uma requisição HTTP e delega a lógica de negócio para o **Service**.
- O **Service** utiliza o **Repository** para realizar operações no banco de dados.

Módulos conectam todas as partes

- **Módulos** agrupam os **Controllers**, **Services** e **Repositories**, garantindo que os recursos necessários estejam disponíveis.

Exemplo prático

Controller: Define rotas e endpoints.

```
1 @Get('/:id')
2 findOne(@Param('id') id: string) {
3   return this.productService.findOne(id);
4 }
```

Service: Implementa a lógica de busca.

```
1 findOne(id: string) {
2   return this.productRepository.findOne(id);
3 }
```

Repository: Realiza a operação no banco de dados.

```
1 findOne(id: string) {
2   return this.productModel.findOne({ where: { id } });
3 }
```

Fluxo completo

1. O cliente faz uma requisição para um endpoint (ex.: GET /products).
2. O **Controller** correspondente chama o **Service** para processar a requisição.
3. O **Service** utiliza o **Repository** para acessar ou manipular dados no banco.
4. O **Repository** interage diretamente com o banco de dados e retorna os dados para o **Service**.
5. O **Service** processa os dados (se necessário) e os retorna para o **Controller**.
6. O **Controller** responde ao cliente com os dados ou uma mensagem apropriada.

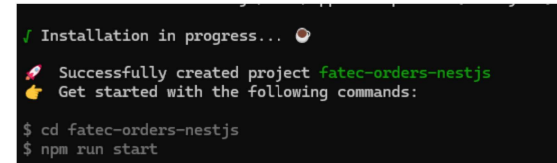
Configuração inicial de Projeto

Configuração do projeto, e comandos importantes

Criação de Projeto

Comando para criar o projeto:

- `npx @nestjs/cli new fatec-orders-nestjs`
- Which package manager would you ❤️ to use? npm



```
✓ Installation in progress... 🔄  
🔧 Successfully created project fatec-orders-nestjs  
👉 Get started with the following commands:  
  
$ cd fatec-orders-nestjs  
$ npm run start
```

Criação de Projeto

Dependências adicionais:

- `@nestjs/swagger`: Documentação com Swagger.
- `class-validator` e `class-transformer`: Validação e transformação de dados.

Criação de Projeto

Instalando dependências:

- `npm install @nestjs/swagger swagger-ui-express class-validator class-transformer`

Execução do Projeto

- npm run start:dev

Para resolver o problema de caracter eol abaixo, adicione "endOfLine": "crlf" no arquivo .prettierrc:

```
main.ts 8, U x
src > main.ts > ...
1 import { NestFactory } from '@nestjs/core'; Delete `cr`
2 import { AppModule } from './app.module'; Delete `cr`
3 Delete `cr`
4 async function bootstrap() { Delete `cr`
5   const app = await NestFactory.create(AppModule); Delete `cr`
6   await app.listen(3000); Delete `cr`
7 } Delete `cr`
8 bootstrap(); Delete `cr`
9
```

Execução do Projeto

No arquivo que inicia a aplicação "main.ts", alterar a porta em que o serviço api é executado, para a porta 80:

```
main.ts U x
src > main.ts > bootstrap
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3
4 async function bootstrap() {
5   const app = await NestFactory.create(AppModule);
6   await app.listen(80);
7 }
8 bootstrap();
```

Execução do Projeto

```
.prettierrc U x main.ts 8, U
.prettierrc > ...
1 {
2   "singleQuote": true,
3   "trailingComma": "all",
4   "endOfLine": "crlf"
5 }
6
```

- Reabra o projeto, e a nova configuração já deverá estar sendo aplicada.

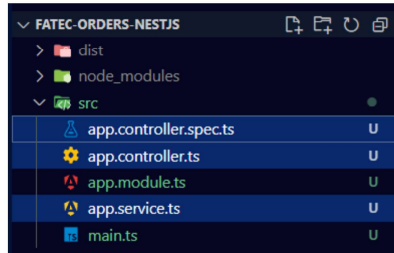
Execução do Projeto

O serviço passará a responder então por <http://localhost>, ou <http://localhost:80>

```
GET http://localhost/ Send 200 OK 3.13 ms 12 B
Body Auth Query Header Docs Raw Header 7 Cookie Timeline
Hello world!
```

Organização do Projeto

- Remover os arquivos: **app.service.ts**, **app.controller.ts** e **app.controller.spec.ts**:



- Assim que for feita a remoção, o **app.module.ts** ficará com erro. Nesse caso, remover as referências que não existem mais.

Organização do Projeto

- O app.module.ts ficará assim:

```
app.module.ts U X
src > app.module.ts > ...
1  import { Module } from '@nestjs/common';
2
3  @Module({
4    imports: [],
5    controllers: [],
6    providers: [],
7  })
8  export class AppModule {}
9
```

Estrutura do Projeto

```
src/
├── app.module.ts
├── main.ts
├── products/
│   ├── products.controller.ts
│   ├── products.service.ts
│   └── dto/
│       ├── create-product.dto.ts
│       └── update-product.dto.ts
```

Implementação do Projeto

Implementar o módulo products:

- nest g module products

Observe que o módulo criado “ProductsModule” foi adicionado em app.module.ts

```
app.module.ts U X
src > app.module.ts > AppModule
1  import { Module } from '@nestjs/common';
2  import { ProductsModule } from './products/products.module';
3
4  @Module({
5    imports: [ProductsModule],
6    controllers: [],
7    providers: [],
8  })
9  export class AppModule {}
10
```

Implementação do Projeto

Implementar o controlador “list-product”:

- nest g controller products/controllers/list-product --flat

Observe que o controlador criado “ListProductController” foi adicionado ao módulo de produtos (products.module.ts)



```
products.module.ts
src > products > products.module.ts > ProductsModule
1 import { Module } from '@nestjs/common';
2 import { ListProductController } from './controllers/list-product.controller';
3
4 @Module({
5   controllers: [ListProductController],
6 })
7 export class ProductsModule {}
8
```

Implementação do Projeto

Implementar o serviço “list-product”:

- nest g service products/services/list-product --flat

Observe que o serviço criado “ListProductService” foi adicionado ao módulo de produtos (products.module.ts)



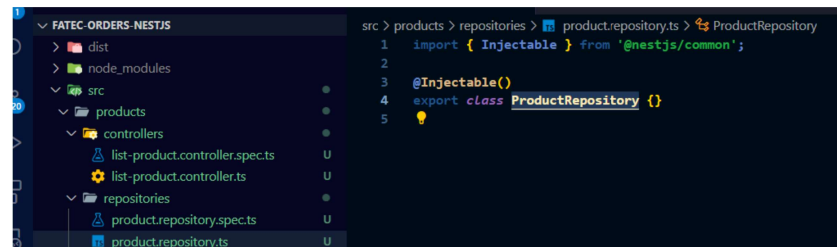
```
products.module.ts
src > products > products.module.ts > ProductsModule
1 import { Module } from '@nestjs/common';
2 import { ListProductController } from './controllers/list-product.controller';
3 import { ListProductService } from './services/list-product.service';
4
5 @Module({
6   controllers: [ListProductController],
7   providers: [ListProductService],
8 })
9 export class ProductsModule {}
10
```

Implementação do Projeto

Implementar o repositório “product”:

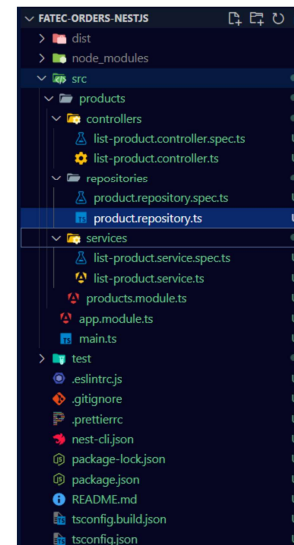
- nest g provider products/repositories/product --flat

Na sequência, altere o nome dos arquivos gerados em /repositories, de “product” para “product.repository”. Altere o nome da classe de “Product” para “ProductRepository”:



```
src > products > repositories > product.repository.ts > ProductRepository
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class ProductRepository {}
5
```

Estrutura do Projeto



```
FATEC-ORDERS-NESTJS
├── dist
├── node_modules
├── src
│   ├── products
│   │   ├── list-product.controller.spec.ts
│   │   ├── list-product.controller.ts
│   │   ├── repositories
│   │   │   ├── product.repository.spec.ts
│   │   │   └── product.repository.ts
│   │   └── services
│   │       ├── list-product.service.spec.ts
│   │       ├── list-product.service.ts
│   │       ├── products.module.ts
│   │       ├── app.module.ts
│   │       ├── main.ts
│   │       └── test
│   ├── .eslintrc.js
│   ├── .gitignore
│   ├── .prettierrc
│   ├── nest-cli.json
│   ├── package-lock.json
│   ├── package.json
│   ├── README.md
│   ├── tsconfig.build.json
│   └── tsconfig.json
```


Lógica

- Implementação do repositório:

```
productrepository.ts U X
src > products > repositories > productrepository.ts > ...
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class ProductRepository {
5   private products = [];
6   constructor() {
7     this.products.push({
8       id: 1,
9       descricao: 'Bolacha', "descricao": Unknown word.
10      marca: 'Trakinas', "Trakinas": Unknown word.
11      valor: 1.99,
12      peso_gramas: 100,
13      sabor: 'morango',
14    });
15  }
16  listProducts() {
17    return this.products;
18  }
19 }
20
```

Lógica

- Implementação do serviço:

```
list-product.service.ts U X
src > products > services > list-product.service.ts > ...
1 import { Injectable } from '@nestjs/common';
2 import { ProductRepository } from '../repositories/product.repository';
3
4 @Injectable()
5 export class ListProductService {
6   constructor(private productRepository: ProductRepository) {}
7   execute() {
8     return this.productRepository.listProducts();
9   }
10 }
```

Lógica

- Implementação do controlador:

```
list-product.controller.ts U X
src > products > controllers > list-product.controller.ts > ...
1 import { Controller, Get } from '@nestjs/common';
2 import { ListProductService } from '../services/list-product.service';
3
4 @Controller('produto')
5 export class ListProductController {
6   constructor(private listProductService: ListProductService) {}
7
8   @Get()
9   handler() {
10     return {
11       produtos: this.listProductService.execute(),
12     };
13   }
14 }
```

CORS

- Habilitação do CORS para o domínio do frontend:

```
main.ts U X
src > main.ts > bootstrap
1 import { NestFactory } from '@nestjs/core';
2 import { AppModule } from './app.module';
3
4 async function bootstrap() {
5   const app = await NestFactory.create(AppModule);
6
7   app.enableCors({
8     origin: 'http://localhost:3000',
9   });
10
11   await app.listen(80);
12 }
13 bootstrap();
```

Próximos Passos

Integração com ORM Sequelize

Configuração do Swagger