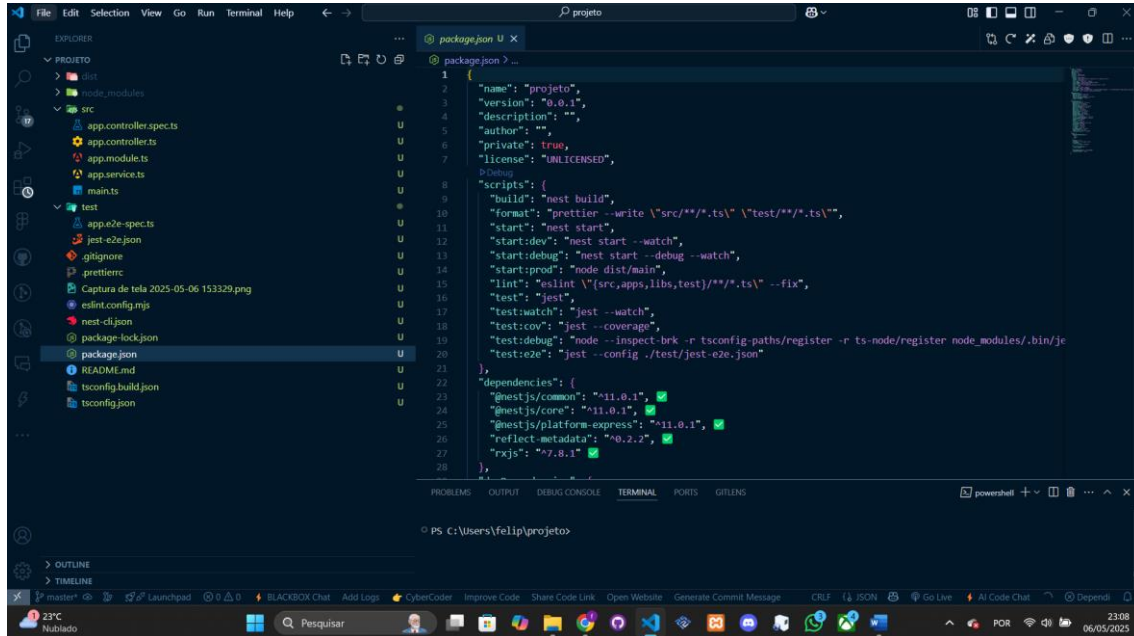


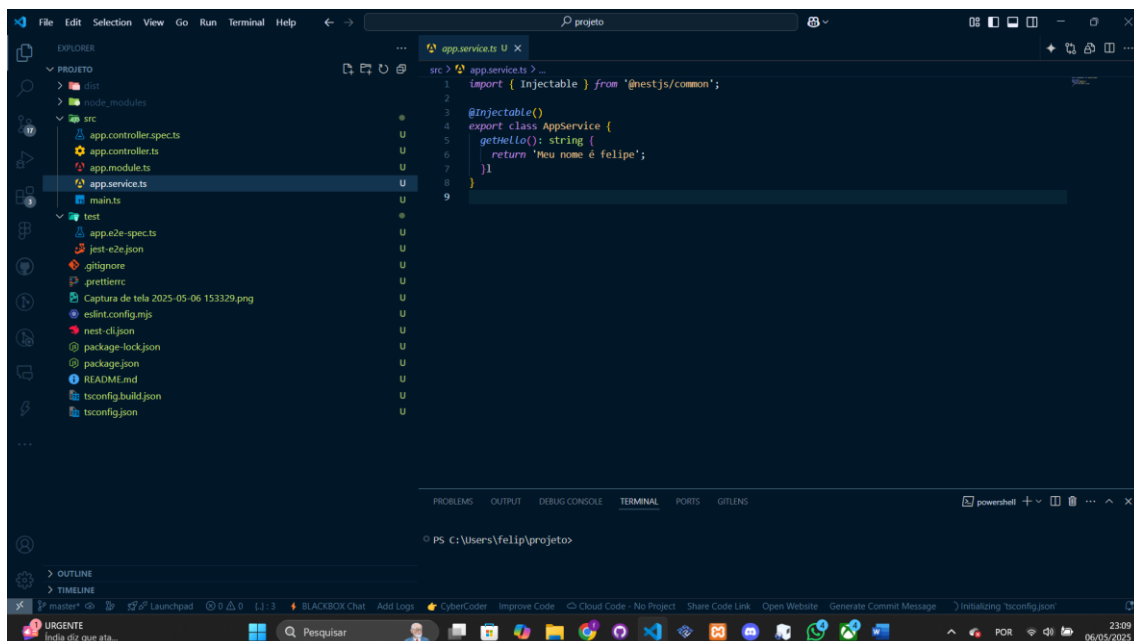
package.json: contém todas as dependências necessárias para a implementação das APIs, além de conter os conjuntos de scripts de execução do programa.



The screenshot shows the Visual Studio Code interface with the 'package.json' file open in the editor. The file is located in the 'projeto' directory. The Explorer sidebar on the left shows the project structure, including 'src' and 'test' folders. The main editor displays the following JSON content:

```
1 {
2   "name": "projeto",
3   "version": "0.0.1",
4   "description": "",
5   "author": "",
6   "private": true,
7   "license": "UNLICENSED",
8   "scripts": {
9     "build": "nest build",
10    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
11    "start": "nest start",
12    "start:dev": "nest start --watch",
13    "start:debug": "nest start --debug --watch",
14    "start:prod": "node dist/main",
15    "lint": "eslint \"src/**/*.ts\" \"test/**/*.ts\" --fix",
16    "test": "jest",
17    "test:watch": "jest --watch",
18    "test:cov": "jest --coverage",
19    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-node/register node_modules/.bin/jest --config ./test/jest-e2e.json",
20    "test:e2e": "jest --config ./test/jest-e2e.json"
21  },
22  "dependencies": {
23    "@nestjs/common": "^11.0.1",
24    "@nestjs/core": "^11.0.1",
25    "@nestjs/platform-express": "^11.0.1",
26    "reflect-metadata": "^0.2.2",
27    "rxjs": "^7.8.1"
28  }
29 }
```

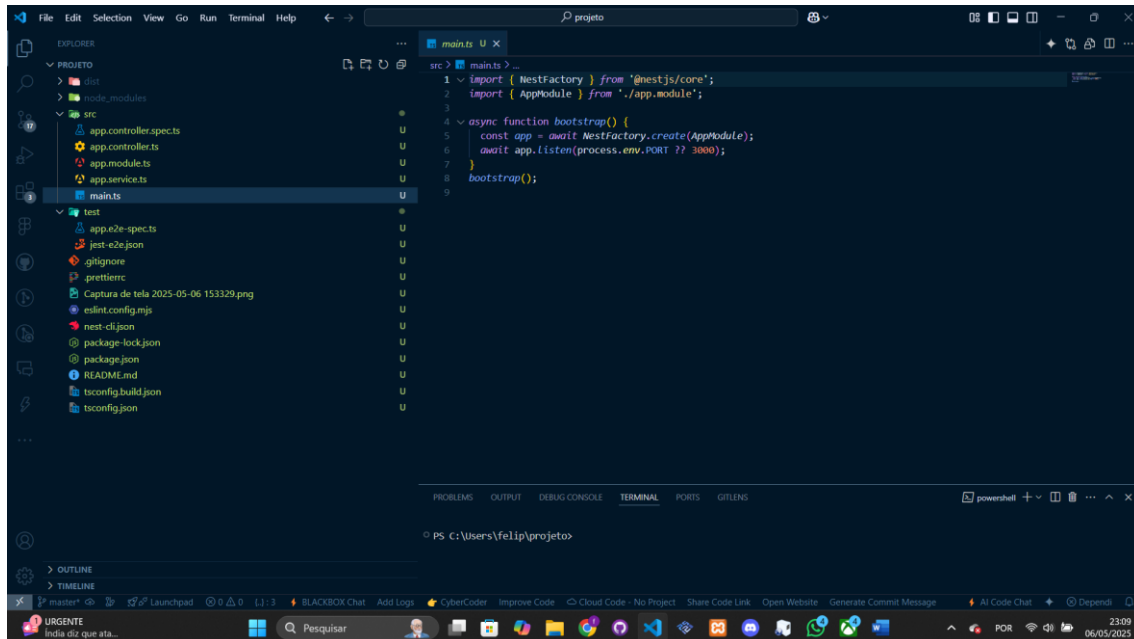
app.servise.ts: é onde são adicionadas as regras de negócio, acesso às informações do banco de dados e conversão de dados; toda parte de obtenção, tratamento e processamento de informações é feita neste arquivo.



The screenshot shows the Visual Studio Code interface with the 'app.servise.ts' file open in the editor. The file is located in the 'src' directory. The Explorer sidebar on the left shows the project structure, including 'src' and 'test' folders. The main editor displays the following TypeScript code:

```
1 import { Injectable } from '@nestjs/common';
2
3 @Injectable()
4 export class AppService {
5   getHello(): string {
6     return 'Neu nome é felipe';
7   }
8 }
9
```

main.ts: é o arquivo de execução do serviço, nele também é definido em qual porta será executado o serviço e todas as requisições irão passar por ele.

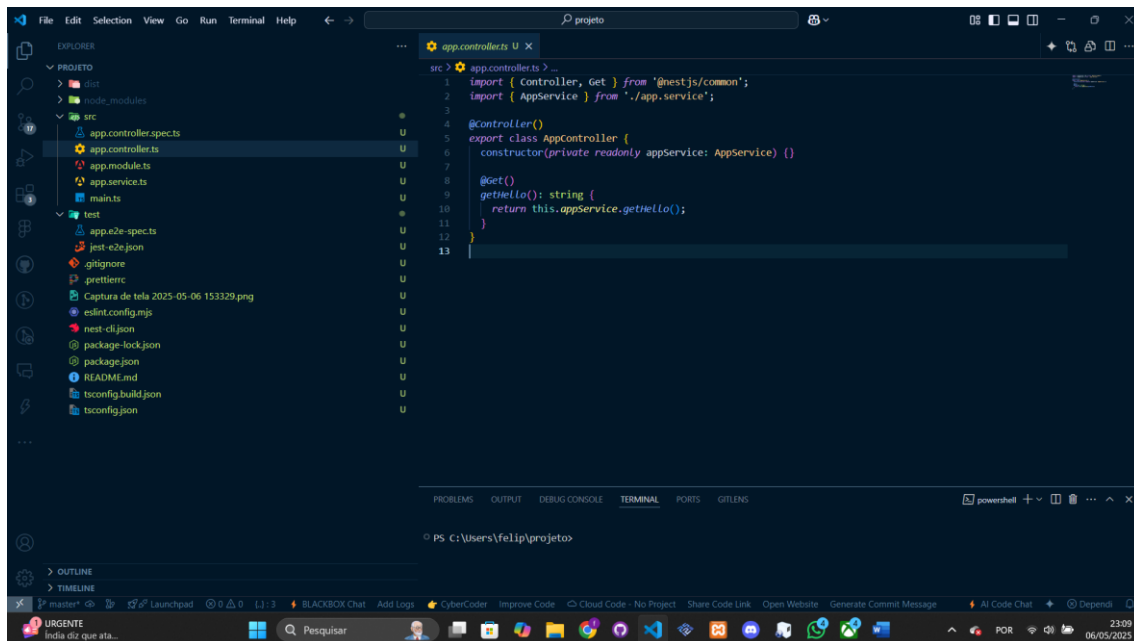


The screenshot shows the Visual Studio Code editor with a project named 'projeto'. The Explorer sidebar on the left shows the file structure, with 'main.ts' selected under the 'src' directory. The main editor window displays the code for 'main.ts'.

```
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(process.env.PORT ?? 3000);
7 }
8 bootstrap();
9
```

The bottom of the window shows a terminal with the PowerShell prompt 'PS C:\Users\Felip\projeto>'.

app.controller.ts: é onde se define as rotas da aplicação, que contém o endereço do serviço no qual a página web faz a chamada.

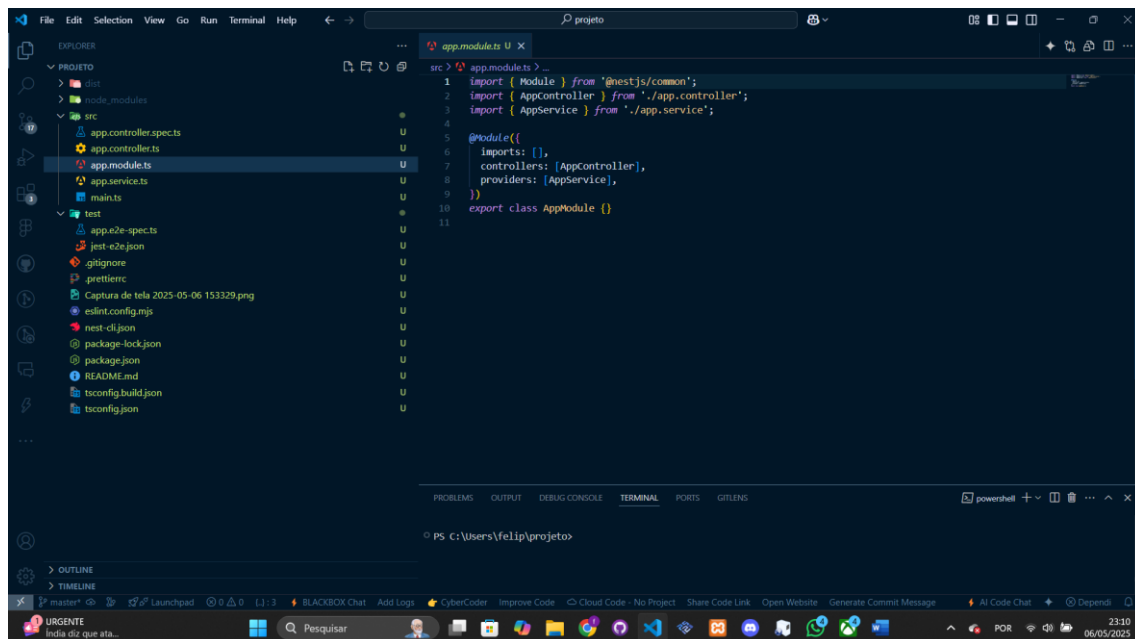


The screenshot shows the Visual Studio Code editor with the same project 'projeto'. The Explorer sidebar shows 'app.controller.ts' selected under the 'src' directory. The main editor window displays the code for 'app.controller.ts'.

```
1 import { Controller, Get } from '@nestjs/common';
2 import { AppService } from './app.service';
3
4 @Controller()
5 export class AppController {
6   constructor(private readonly appService: AppService) {}
7
8   @Get()
9   getHello(): string {
10     return this.appService.getHello();
11   }
12 }
13
```

The bottom of the window shows the same terminal with the PowerShell prompt 'PS C:\Users\Felip\projeto>'.

app.module.ts: é utilizado para declarar todas as classes controller e servisse na qual é utilizado, serve para agrupar as classes e organizar os códigos.



Página no navegador: o resultado do código depois de feito e executado, localizado na porta 3000, mostrando a frase “Meu nome é Ananda”.

