

# Tópicos Especiais

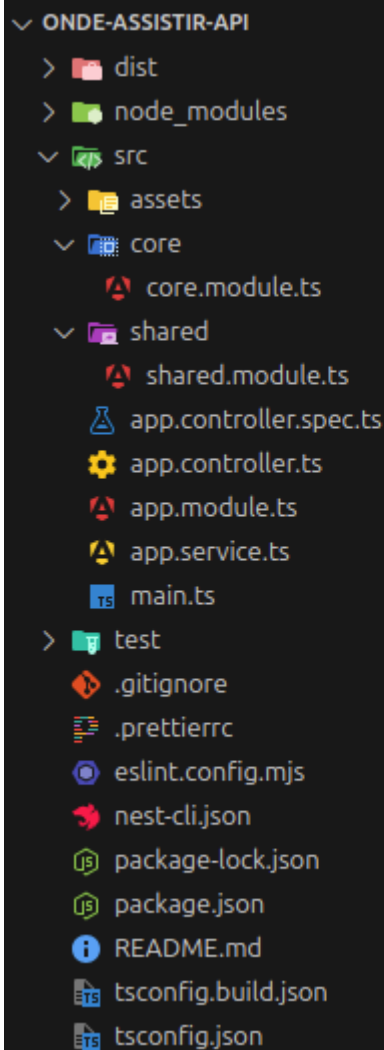
## Atividade 05 - Criação e estruturação de um projeto Modular no NestJS

Prof. Dr. Roni Fabio Banaszewski  
Matheus Rocha Camargo - 2132214

### 1 - Print do terminal mostrando a versão do Node.js.

```
matheus@matheus-Aspire-A515-54G:~/UTFPR/2025-1/topicos-especiais/onde-assistir-api$ node -v  
v22.13.1
```

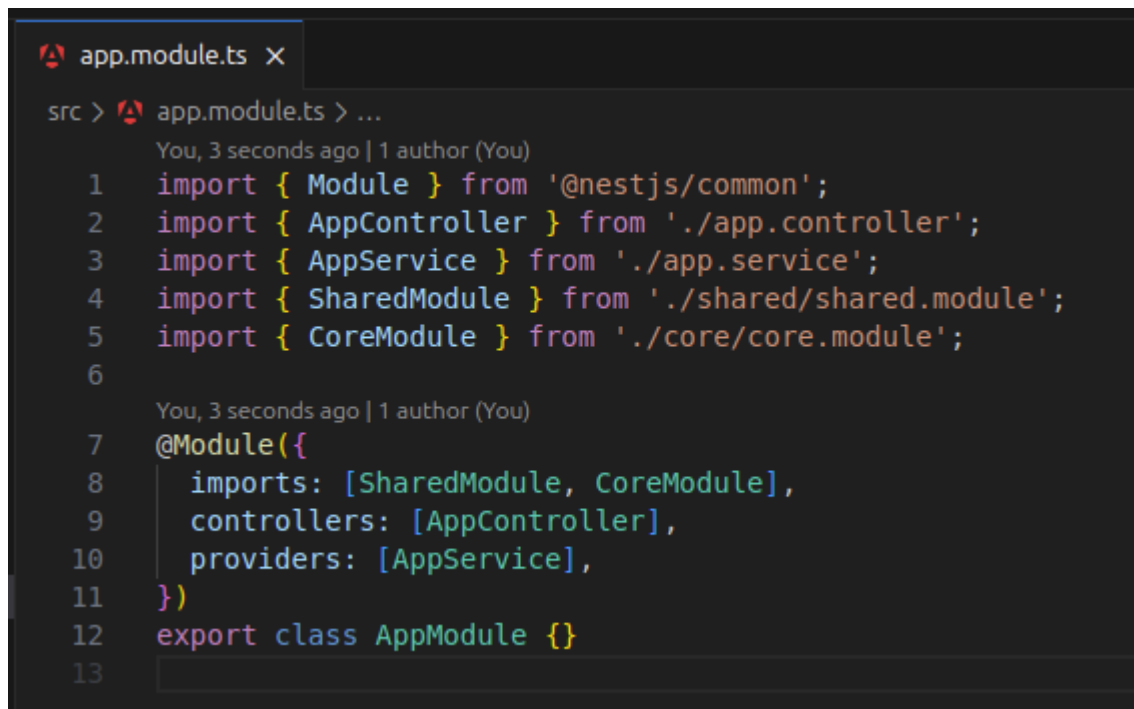
### 2 - Print da estrutura de pastas em `src`, incluindo `app.module.ts`, `core/core.module.ts` e `shared/shared.module.ts`.



The screenshot shows a file explorer view of a project named 'ONDE-ASSISTIR-API'. The structure is as follows:

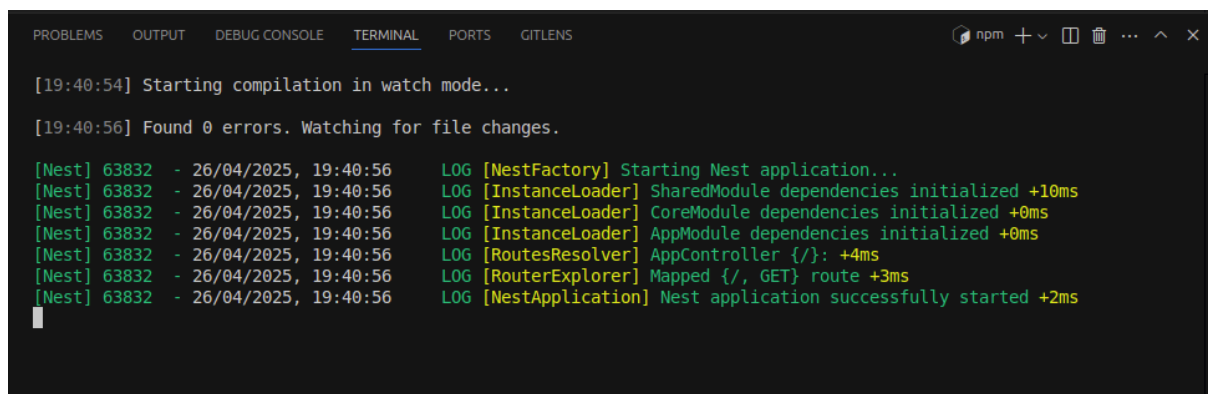
- ONDE-ASSISTIR-API
  - dist
  - node\_modules
  - src
    - assets
    - core
      - core.module.ts
    - shared
      - shared.module.ts
    - app.controller.spec.ts
    - app.controller.ts
    - app.module.ts
    - app.service.ts
    - main.ts
  - test
  - .gitignore
  - .prettierrc
  - eslint.config.mjs
  - nest-cli.json
  - package-lock.json
  - package.json
  - README.md
  - tsconfig.build.json
  - tsconfig.json

### 3 - Print da importação de `CoreModule` e `SharedModule` dentro de `AppModule`.



```
app.module.ts x
src > app.module.ts > ...
You, 3 seconds ago | 1 author (You)
1 import { Module } from '@nestjs/common';
2 import { AppController } from './app.controller';
3 import { AppService } from './app.service';
4 import { SharedModule } from '../shared/shared.module';
5 import { CoreModule } from '../core/core.module';
6
You, 3 seconds ago | 1 author (You)
7 @Module({
8   imports: [SharedModule, CoreModule],
9   controllers: [AppController],
10  providers: [AppService],
11 })
12 export class AppModule {}
13
```

### 4 - Print do terminal com o servidor NestJS iniciado, indicando que o projeto está rodando corretamente.




```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
[19:40:54] Starting compilation in watch mode...
[19:40:56] Found 0 errors. Watching for file changes.
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [NestFactory] Starting Nest application...
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [InstanceLoader] SharedModule dependencies initialized +10ms
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [InstanceLoader] CoreModule dependencies initialized +0ms
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [RoutesResolver] AppController {/}: +4ms
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [RouterExplorer] Mapped {/, GET} route +3ms
[Nest] 63832 - 26/04/2025, 19:40:56 LOG [NestApplication] Nest application successfully started +2ms
```

## 5- Print ou descrição de uma sugestão feita pelo Copilot:



matheusrc-dev

o que esses módulos abaixo fazem?

 **GitHub.copilot-chat.github.copilot.editor** used /explain (rerun without)

Os módulos mencionados no código ( `SharedModule` e `CoreModule` ) são partes de um projeto desenvolvido com o framework **NestJS**, que organiza a aplicação em módulos para facilitar a modularidade e a reutilização de código. Vou explicar o propósito geral de cada um:

### 1. SharedModule

- O **SharedModule** geralmente é usado para agrupar **recursos compartilhados** que podem ser reutilizados em diferentes partes da aplicação.
- Exemplos de itens que podem estar no `SharedModule` :
  - Serviços genéricos (ex.: um serviço de logging ou de manipulação de datas).
  - Pipes, filtros ou interceptors customizados.
  - Componentes que são usados em vários módulos (ex.: utilitários ou helpers).
- Normalmente, o `SharedModule` é exportado para que outros módulos possam utilizá-lo.

Exemplo de definição de um `SharedModule` :

```
@Module({
  providers: [LoggerService],
  exports: [LoggerService], // Disponível para outros módulos
})
export class SharedModule {}
```

### 2. CoreModule

- O **CoreModule** geralmente contém **recursos centrais** ou **serviços globais** que são usados em toda a aplicação.
- Ele é projetado para ser importado **apenas uma vez** (geralmente no `AppModule` ) e pode conter:
  - Configurações globais (ex.: variáveis de ambiente, configuração de banco de dados).
  - Serviços que devem ser únicos na aplicação (ex.: autenticação, cache, ou serviços de configuração).
- Para evitar múltiplas importações, o `CoreModule` geralmente usa o padrão **singleton**.

Exemplo de definição de um `CoreModule` :

```
@Module({
  providers: [ConfigService],
  exports: [ConfigService], // Disponível para outros módulos
})
export class CoreModule {
  static forRoot(): DynamicModule {
    return {
      module: CoreModule,
      providers: [ConfigService],
    };
  }
}
```

### Resumo

- **SharedModule**: Recursos reutilizáveis e compartilhados entre módulos.
- **CoreModule**: Recursos centrais e globais da aplicação, geralmente importados apenas uma vez.