Module

August 17, 2026

Module

Kind: Function

Source: `packages/common/decorators/modules/module.decorator.ts`

Part of: Common

Decorator that marks a class as a module.

Modules are used by Nest to organize the application structure into scopes.
Controllers
and Providers are scoped by the module they are declared in. Modules and their classes (Controllers and Providers) form a graph that determines how Nest performs Dependency Injection (DI).

`Module()` is a class decorator that marks a class as a NestJS module and attaches module metadata to it. Modules define application boundaries by grouping controllers, providers, imports, and exports into dependency-injection scopes.

Signature

```
function Module(metadata: ModuleMetadata): ClassDecorator
```

Parameters

Name	Type
`metadata`	`ModuleMetadata`

Returns: `ClassDecorator`

Diagram

```
graph LR
  AppModule["@Module() AppModule"] --> Controllers["Controllers"]
  AppModule --> Providers["Providers"]
  AppModule --> Imports["Imported Modules"]
  AppModule --> Exports["Exported Providers"]

  Controllers --> Scope["Module DI Scope"]
  Providers --> Scope
  Imports --> Scope
  Scope --> Nest["Nest Dependency Injection Container"]
```

Usage

```
import { Module } from '@nestjs/common';
import { UsersController } from './users.controller';
import { UsersService } from './users.service';
import { DatabaseModule } from '../database/database.module';

@Module({
  imports: [DatabaseModule],
  controllers: [UsersController],
  providers: [UsersService],
  exports: [UsersService],
})
export class UsersModule {}
```

AI Coding Instructions

- Apply `@Module()` only to classes intended to act as NestJS module boundaries.
- Register controllers in `controllers` and injectable services, repositories, and factories in `providers`.
- Add dependent modules to `imports`; do not add their providers directly unless they are explicitly exported.
- Use `exports` to make selected providers available to modules that import this module.
- Avoid circular module imports; use NestJS `forwardRef()` only when a circular dependency cannot be redesigned.

Used by

181 references from 181 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (181)

- `ROUTES` — `packages/core/router/router-module.ts`:9
- `AppModule` — `integration/cors/src/app.module.ts`:4
- `AppModule` — `integration/discovery/src/app.module.ts`:6
- `MyWebhookModule` — `integration/discovery/src/my-webhook/my-webhook.module.ts`:5
- `AppModule` — `integration/graphql-code-first/src/app.module.ts`:7
- `RecipesModule` — `integration/graphql-code-first/src/recipes/recipes.module.ts`:8
- `AppModule` — `integration/graphql-schema-first/src/app.module.ts`:7
- `AsyncClassApplicationModule` — `integration/graphql-schema-first/src/async-options-class.module.ts`:15

…and 173 more.