Injectable

August 18, 2026

Injectable

Kind: Function

Source: `packages/common/decorators/core/injectable.decorator.ts`

Part of: Common

Decorator that marks a class as a provider.
Providers can be injected into other classes via constructor parameter injection using Nest's built-in Dependency Injection (DI) system.

When injecting a provider, it must be visible within the module scope (loosely speaking, the containing module) of the class it is being injected into. This can be done by:

- defining the provider in the same module scope
- exporting the provider from one module scope and importing that module into the module scope of the class being injected into
- exporting the provider from a module that is marked as global using the `@Global()` decorator

Providers can also be defined in a more explicit and imperative form using various custom provider techniques that expose more capabilities of the DI system.

`@Injectable()` marks a class as a Nest provider, allowing Nest's dependency injection container to create and inject it into other classes. Providers must be registered in a module and visible in the consuming class's module scope through local registration, module imports/exports, or a global module.

Signature

```
function Injectable(options: InjectableOptions): ClassDecorator
```

Parameters

Name	Type
`options`	`InjectableOptions`

Returns: `ClassDecorator`

Diagram

```
graph LR
  Service["@Injectable()<br/>UsersService"] --> Module["UsersModule providers"]
  Module --> Container["Nest DI Container"]
  Container --> Consumer["UsersController<br/>constructor(usersService)"]
```

Usage

```
import { Controller, Get, Injectable, Module } from '@nestjs/common';

@Injectable()
export class UsersService {
  findAll() {
    return ['Ada', 'Grace'];
  }
}

@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @Get()
  findAll() {
    return this.usersService.findAll();
  }
}

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

AI Coding Instructions

- Apply `@Injectable()` to classes intended to be instantiated and managed by Nest's DI container.
- Register injectable classes in a module's `providers` array before injecting them elsewhere.
- Export providers from their defining module and import that module where the provider is consumed across module boundaries.
- Use constructor injection for dependencies; avoid manually instantiating injectable providers with `new`.
- Prefer custom provider definitions when a dependency requires a token, factory, value, alias, or custom scope.