Common errors

During your development with NestJS, you may encounter various errors as you learn the framework.

"Cannot resolve dependency" error

> info **Hint** Check out the NestJS Devtools which can help you resolve the "Cannot resolve dependency" error effortlessly.

Probably the most common error message is about Nest not being able to resolve dependencies of a provider. The error message usually looks something like this:

```
Nest can't resolve dependencies of the <provider> (?). Please make sure
that the argument <unknown_token> at index [<index>] is available in the
<module> context.

Potential solutions:
- Is <module> a valid NestJS module?
- If <unknown_token> is a provider, is it part of the current <module>?
- If <unknown_token> is exported from a separate @Module, is that module
imported within <module>?
  @Module({
    imports: [ /* the Module containing <unknown_token> */ ]
  })
```

The most common culprit of the error, is not having the `<provider>` in the module's `providers` array. Please make sure that the provider is indeed in the `providers` array and following standard NestJS provider practices.

There are a few gotchas, that are common. One is putting a provider in an `imports` array. If this is the case, the error will have the provider's name where `<module>` should be.

If you run across this error while developing, take a look at the module mentioned in the error message and look at its `providers`. For each provider in the `providers` array, make sure the module has access to all of the dependencies. Often times, `providers` are duplicated in a "Feature Module" and a "Root Module" which means Nest will try to instantiate the provider twice. More than likely, the module containing the `<provider>` being duplicated should be added in the "Root Module"'s `imports` array instead.

If the `<unknown_token>` above is the string `dependency`, you might have a circular file import. This is different from the circular dependency below because instead of having providers depend on each other in their constructors, it just means that two files end up importing each other. A common case would be a module file declaring a token and importing a provider, and the provider import the token constant from the module file. If you are using barrel files, ensure that your barrel imports do not end up creating these circular imports as well.

If the `<unknown_token>` above is the string `Object`, it means that you're injecting using an type/interface without a proper provider's token. To fix that, make sure you're importing the class reference or use a custom token with `@Inject()` decorator. Read the custom providers page.

Also, make sure you didn't end up injecting the provider on itself because self-injections are not allowed in NestJS. When this happens, `<unknown_token>` will likely be equal to `<provider>`.

If you are in a **monorepo setup**, you may face the same error as above but for core provider called `ModuleRef` as a `<unknown_token>`:

```
Nest can't resolve dependencies of the <provider> (?).
Please make sure that the argument ModuleRef at index [<index>] is
available in the <module> context.
...
```

This likely happens when your project end up loading two Node modules of the package `@nestjs/core`, like this:

```
.
├── package.json
├── apps
│   └── api
│       └── node_modules
│           └── @nestjs/bull
│               └── node_modules
│                   └── @nestjs/core
└── node_modules
    ├── (other packages)
    └── @nestjs/core
```

Solutions:

- For **Yarn** Workspaces, use the nohoist feature to prevent hoisting the package `@nestjs/core`.
- For **pnpm** Workspaces, set `@nestjs/core` as a peerDependencies in your other module and `"dependenciesMeta": {{ '{' }}"other-module-name": {{ '{' }}"injected": true{{ '}}' }}` in the app package.json where the module is imported. see: dependenciesmetainjected

"Circular dependency" error

Occasionally you'll find it difficult to avoid circular dependencies in your application. You'll need to take some steps to help Nest resolve these. Errors that arise from circular dependencies look like this:

```
Nest cannot create the <module> instance.
The module at index [<index>] of the <module> "imports" array is
undefined.

Potential causes:
- A circular dependency between modules. Use forwardRef() to avoid it.
Read more: https://docs.nestjs.com/fundamentals/circular-dependency
- The module at index [<index>] is of type "undefined". Check your import
```

```
  statements and the type of the module.

  Scope [<module_import_chain>]
  # example chain AppModule -> FooModule
```

Circular dependencies can arise from both providers depending on each other, or typescript files depending on each other for constants, such as exporting constants from a module file and importing them in a service file. In the latter case, it is advised to create a separate file for your constants. In the former case, please follow the guide on circular dependencies and make sure that both the modules **and** the providers are marked with `forwardRef`.

Debugging dependency errors

Along with just manually verifying your dependencies are correct, as of Nest 8.1.0 you can set the `NEST_DEBUG` environment variable to a string that resolves as truthy, and get extra logging information while Nest is resolving all of the dependencies for the application.

In the above image, the string in yellow is the host class of the dependency being injected, the string in blue is the name of the injected dependency, or its injection token, and the string in purple is the module in which the dependency is being searched for. Using this, you can usually trace back the dependency resolution for what's happening and why you're getting dependency injection problems.

"File change detected" loops endlessly

Windows users who are using TypeScript version 4.9 and up may encounter this problem. This happens when you're trying to run your application in watch mode, e.g `npm run start:dev` and see an endless loop of the log messages:

```
XX:XX:XX AM - File change detected. Starting incremental compilation...
XX:XX:XX AM - Found 0 errors. Watching for file changes.
```

When you're using the NestJS CLI to start your application in watch mode it is done by calling `tsc --watch`, and as of version 4.9 of TypeScript, a new strategy for detecting file changes is used which is likely to be the cause of this problem. In order to fix this problem, you need to add a setting to your tsconfig.json file after the `"compilerOptions"` option as follows:

```
  "watchOptions": {
    "watchFile": "fixedPollingInterval"
  }
```

This tells TypeScript to use the polling method for checking for file changes instead of file system events (the new default method), which can cause issues on some machines. You can read more about the `"watchFile"` option in TypeScript documentation.