CLI Plugin

> warning **Warning** This chapter applies only to the code first approach.

TypeScript's metadata reflection system has several limitations which make it impossible to, for instance, determine what properties a class consists of or recognize whether a given property is optional or required. However, some of these constraints can be addressed at compilation time. Nest provides a plugin that enhances the TypeScript compilation process to reduce the amount of boilerplate code required.

> info **Hint** This plugin is **opt-in**. If you prefer, you can declare all decorators manually, or only specific decorators where you need them.

Overview

The GraphQL plugin will automatically:

- annotate all input object, object type and args classes properties with `@Field` unless `@HideField` is used
- set the `nullable` property depending on the question mark (e.g. `name?: string` will set `nullable: true`)
- set the `type` property depending on the type (supports arrays as well)
- generate descriptions for properties based on comments (if `introspectComments` set to `true`)

Please, note that your filenames **must have** one of the following suffixes in order to be analyzed by the plugin: `['.input.ts', '.args.ts', '.entity.ts', '.model.ts']` (e.g., `author.entity.ts`). If you are using a different suffix, you can adjust the plugin's behavior by specifying the `typeFileNameSuffix` option (see below).

With what we've learned so far, you have to duplicate a lot of code to let the package know how your type should be declared in GraphQL. For example, you could define a simple `Author` class as follows:

```
@@filename(authors/models/author.model)
@ObjectType()
export class Author {
  @Field(type => ID)
  id: number;

  @Field({ nullable: true })
  firstName?: string;

  @Field({ nullable: true })
  lastName?: string;

  @Field(type => [Post])
  posts: Post[];
}
```

While not a significant issue with medium-sized projects, it becomes verbose & hard to maintain once you have a large set of classes.

By enabling the GraphQL plugin, the above class definition can be declared simply:

```
@@filename(authors/models/author.model)
@ObjectType()
export class Author {
  @Field(type => ID)
  id: number;
  firstName?: string;
  lastName?: string;
  posts: Post[];
}
```

The plugin adds appropriate decorators on-the-fly based on the **Abstract Syntax Tree**. Thus, you won't have to struggle with `@Field` decorators scattered throughout the code.

> info **Hint** The plugin will automatically generate any missing GraphQL properties, but if you need to override them, simply set them explicitly via `@Field()`.

Comments introspection

With the comments introspection feature enabled, CLI plugin will generate descriptions for fields based on comments.

For example, given an example `roles` property:

```
/**
 * A list of user's roles
 */
@Field(() => [String], {
  description: `A list of user's roles`
})
roles: string[];
```

You must duplicate description values. With `introspectComments` enabled, the CLI plugin can extract these comments and automatically provide descriptions for properties. Now, the above field can be declared simply as follows:

```
/**
 * A list of user's roles
 */
roles: string[];
```

Using the CLI plugin

To enable the plugin, open `nest-cli.json` (if you use Nest CLI) and add the following `plugins` configuration:

```
{
  "collection": "@nestjs/schematics",
  "sourceRoot": "src",
  "compilerOptions": {
    "plugins": ["@nestjs/graphql"]
  }
}
```

You can use the `options` property to customize the behavior of the plugin.

```
"plugins": [
  {
    "name": "@nestjs/graphql",
    "options": {
      "typeFileNameSuffix": [".input.ts", ".args.ts"],
      "introspectComments": true
    }
  }
]
```

The `options` property has to fulfill the following interface:

```
export interface PluginOptions {
  typeFileNameSuffix?: string[];
  introspectComments?: boolean;
}
```

Option	Default	Description
`typeFileNameSuffix`	`['.input.ts', '.args.ts', '.entity.ts', '.model.ts']`	GraphQL types files suffix
`introspectComments`	`false`	If set to true, plugin will generate descriptions for properties based on comments

If you don't use the CLI but instead have a custom `webpack` configuration, you can use this plugin in combination with `ts-loader`:

```
getCustomTransformers: (program: any) => ({
  before: [require('@nestjs/graphql/plugin').before({}, program)]
}),
```

SWC builder

For standard setups (non-monorepo), to use CLI Plugins with the SWC builder, you need to enable type checking, as described here.

```
$ nest start -b swc --type-check
```

For monorepo setups, follow the instructions here.

```
$ npx ts-node src/generate-metadata.ts
# OR npx ts-node apps/{YOUR_APP}/src/generate-metadata.ts
```

Now, the serialized metadata file must be loaded by the `GraphQLModule` method, as shown below:

```
import metadata from './metadata'; // <-- file auto-generated by the
"PluginMetadataGenerator"

GraphQLModule.forRoot<...>({
  ..., // other options
  metadata,
}),
```

Integration with `ts-jest` (e2e tests)

When running e2e tests with this plugin enabled, you may run into issues with compiling schema. For example, one of the most common errors is:

```
Object type <name> must define one or more fields.
```

This happens because `jest` configuration does not import `@nestjs/graphql/plugin` plugin anywhere.

To fix this, create the following file in your e2e tests directory:

```
const transformer = require('@nestjs/graphql/plugin');

module.exports.name = 'nestjs-graphql-transformer';
// you should change the version number anytime you change the
configuration below - otherwise, jest will not detect changes
module.exports.version = 1;

module.exports.factory = (cs) => {
  return transformer.before(
    {
      // @nestjs/graphql/plugin options (can be empty)
    },
    cs.program, // "cs.tsCompiler.program" for older versions of Jest (<=
v27)
  );
};
```

With this in place, import AST transformer within your `jest` configuration file. By default (in the starter application), e2e tests configuration file is located under the `test` folder and is named `jest-e2e.json`.

```
{
  ... // other configuration
  "globals": {
    "ts-jest": {
      "astTransformers": {
        "before": ["<path to the file created above>"]
      }
    }
  }
}
```

If you use `jest@^29`, then use the snippet below, as the previous approach got deprecated.

```
{
  ... // other configuration
  "transform": {
    "^.+\\.(t|j)s$": [
      "ts-jest",
      {
        "astTransformers": {
          "before": ["<path to the file created above>"]
        }
      }
    ]
  }
}
```