

Instalar Nest.js CLI: Command line interface

```
npm i -g @nestjs/cli
```

Nuevo proyecto: en el path actual

```
nest new project-name
```

Comandos útiles del CLI

```
nest generate <comando>
nest g <comando>
```

Mostrar ayuda: en cualquier comando

```
nest -h
nest g -h
nest g r nombre -h
```

Componentes comunes: Añadir -h para extras

```
# Crear un resolver
nest g r <path/nombre>

# Crear un módulo
nest g mo <path/nombre>

# Crear un servicio
nest g s <path/nombre>

# Crear un recurso completo
nest g resource <nombre>
```

Instalación de GraphQL en Nest:

Siempre es bueno revisar la documentación oficial

```
yarn add
  @nestjs/graphql
  @nestjs/apollo
  graphql
  apollo-server-express
```

Configurar GraphQL Module: app.module.ts

```
@Module({
  imports: [
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      // debug: false,
      // playground: false
      autoSchemaFile: join(process.cwd(), 'src/schema.gql'),
    }),
  ],
})
```

Endpoint por defecto

```
http://localhost:3000/graphql
```

Nota: Tengan presente que sin ninguna definición de schema (**schema.gql**), el servidor de Nest no se levantará.

Apollo Sandbox: (Opcional)

```
yarn add apollo-server-core
```

```
import {
  ApolloServerPluginLandingPageLocalDefault
} from 'apollo-server-core';
@Module({
  imports: [
    GraphQLModule.forRoot<ApolloDriverConfig>({
      ...
      playground: false,
      plugins: [
        ApolloServerPluginLandingPageLocalDefault
      ]
    }),
  ],
})
```

Usos dentro de un “resolver”

```
# Query => @nestjs/graphql
# [Pet] = Arreglo de mascotas
@Query(() => [Pet], { name: 'pets' })
async pets(): Promise<Pet[]> {
  return; // lógica
}

# Mutation, Args => @nestjs/graphql
# Pet = Una sola mascota
@Mutation(() => Pet)
async createPet(
  @Args('createPetInput')
  createPetInput: CreatePetInput
): Promise<Pet> {
  return; // lógica
}

# ResolveField
# El nombre es importante
@ResolveField(() => Owner)
async owner(
  @Parent() pet: Pet
): Promise<Owner> {
  return; // lógica
}
```

Nest + GraphQL

Página de Atajos

Objetos comunes de Nest / GraphQL

Query	Mutation	Args
Parent	Resolver	ResolveField
Int	Float	ID
ObjectType	Field	InputType

Librerías externas útiles:

```
yarn add class-validator class-transformer
```

Algunos decoradores de Class Validator

IsOptional	IsPositive	IsMongoId
IsArray	IsString	IsUUID
IsDecimal	IsDate	IsDateString
IsBoolean	IsEmail	IsUrl

Configuración global de pipes

```
app.useGlobalPipes(  
  new ValidationPipe({  
    whitelist: true,  
    forbidNonWhitelisted: true,  
  })  
);
```

whiteList: Remueve todo lo que no está incluido en los DTOs

forbidNonWhiteListed: Retorna bad request si hay propiedades en el objeto no requeridas

Estructura de módulo recomendado:

```
src  
- common  
- decorators  
- dtos  
- args  
- inputs  
- filters  
- guards  
- interceptors  
- middleware  
- pipes  
- common.resolver.ts  
- common.module.ts  
- common.service.ts
```

Al ser una estructura recomendada no lo hace obligatorio, pero es aconsejable tener organizado nuestro proyecto de forma modular.

GraphQL core building blocks:

SDL: Schema Definition Language. Es un lenguaje agnóstico muy simple, que permite escribir la definición de cómo funcionará nuestro GraphQL endpoint, también conocido como “GraphQL schema language”

Query: Usado para leer o cargar valores, un query es similar a una petición GET, pero nos permite solicitar la información necesaria desde el frontend.

Mutations: Son queries usados para modificar la data almacenada y retornar valores.

Arguments: Información adicional que se puede proveer en los queries. Esta información puede estar presente en varios niveles del query para aplicar filtros o condiciones especiales.

Inputs: En una mutación, es la información que llamaríamos “body” en una petición REST tradicional.

Directives: Usadas para evitar la interpolación a la hora de construir un query string o para anotar que una propiedad esta obsoleta y no debe de usarse. Las directivas comunes son:

```
skip include deprecated
```

Fragments: Unidades reutilizables para construir grupos de campos y así evitar re-escribir todo varias veces.

Variables: Usualmente cuando escribimos queries, no estarán “hard-coded”, y será necesario enviar variables que cambian de forma dinámica para obtener los resultados de los queries (mutaciones son queries también)

Enumeration Types: también son conocidos como “Enums”, es un tipo especial que restringe las opciones a un set de valores predefinidos.



Nest + GraphQL

Página de Atajos



Scalar Types: Son básicamente las hojas del árbol, o las hojas del query, es la última unidad que nos dice data se requiere retornar. Los tipos básicos son:

Int	32-bit int
Float	Número con decimales (double-precision)
String	UTF-8 character set sequence
Boolean	true or false
ID	Representa un identificador único.



Resolvers: Proveen las instrucciones para transformar las instrucciones provenientes del cliente en data que GraphQL puede utilizar. Los resolvers son similares a los controladores tradicionales de un REST endpoint con Nest, pero son técnicamente “providers”.

Algunos ejemplos útiles:

Asignar nombres de propiedades

```
query Query {  
  rnd1: randomNumber  
  rnd2: randomNumber  
  rnd3: randomNumber  
  rnd4: randomNumber  
}
```

Enviar argumentos:

Operación

```
query ($todoId: Int!) {  
  todo(id: $todoId) {  
    id  
    description  
    done  
  }  
}
```

Variables

```
{  
  "todoId": 1  
}
```

Fragments:

Operación

```
{  
  todos {  
    ...todoFields  
  }  
}  
  
fragment todoFields on Todo {  
  id  
  description  
  done  
}
```

Comparación lado a lado

```
query ($todoId1: Int!, $todoId2: Int!)  
{  
  todo1: todo(id: $todoId1) {  
    id  
    description  
  }  
  todo2: todo(id: $todoId2) {  
    id  
    description  
  }  
}
```

Variables (atención que la “,” si es necesaria aquí)

```
{  
  "todoId1": 1,  
  "todoId2": 2,  
}
```

Usando directivas

Operación

```
query (  
  $includeIf: Boolean!,  
  $skipIf: Boolean!  
) {  
  completedTodos @include(if: $includeIf)  
  pendingTodos @skip(if: $skipIf)  
}
```

Variables

```
{  
  "includeIf": true,  
  "skipIf": true,  
}
```