Prisma

Prisma turns your database into a GraphQL API and enables the use of GraphQL as a universal query language for all databases. Instead of writing SQL or using a NoSQL API, you can query your database with GraphQL. In this chapter we won't go into details about Prisma, so head over to their website and have a look what features are available.

> **NOTICE**
> In this article, you'll learn how to integrate `Prisma` into the Nest framework. We assume that you are already familiar with the GraphQL concepts and the `@nestjs/graphql` module.

Dependencies

Firstly, we need to install the required packages:

```
$ npm install --save prisma-binding
```

Setup Prisma

While working with Prisma you can either host your own instance or use the Prisma Cloud. In this introduction we are going to use the demo server provided by Prisma.

1. Install the Prisma CLI `npm install -g prisma`
2. Create a new service `prisma init`, choose the demo server and follow the instructions
3. Deploy your service `prisma deploy`

If you find yourself in trouble jump over to their Quick Start section for further details. Eventually you should see two new files in your project directory, `prisma.yaml` configuration file:

```
endpoint: https://us1.prisma.sh/nest-f6ec12/prisma/dev
datamodel: datamodel.graphql
```

and automatically created data model, `datamodel.graphql`.

```
type User {
  id: ID! @unique
  name: String!
```

```
}
```

> **NOTICE**
> In the real-world applications you will create more complex data models. For more information about data modeling in Prisma click **here**.

By typing `prisma playground` you can open the Prisma GraphQL playground.

Create the client

There are a couple of ways to integrate a GraphQL API. We are going to use **GraphQL CLI**, a command line tool for common GraphQL development workflows. To install the GraphQL CLI use the following command:

```
$ npm install -g graphql-cli
```

Next, create your `.graphqlconfig` in the root directory of the your Nest application:

```
$ touch .graphqlconfig.yml
```

Put the following content into it:

```
projects:
  database:
    schemaPath: src/prisma/prisma-types.graphql
    extensions:
      endpoints:
        default: https://us1.prisma.sh/nest-f6ec12/prisma/dev
      codegen:
        - generator: prisma-binding
          language: typescript
          output:
            binding: src/prisma/prisma.binding.ts
```

To download your Prisma GraphQL schema to `prisma/prisma-types.graphql` and create your Prisma client under `prisma/prisma.binding.graphql`, run the following commands in your terminal:

```
$ graphql get-schema --project database
$ graphql codegen --project database
```

Integration

Almost done. Now, let's create a module for our Prisma integration.

prisma.service.ts JS

```
import { Injectable } from '@nestjs/common';
import { Prisma } from './prisma.binding';

@Injectable()
export class PrismaService extends Prisma {
  constructor() {
    super({
      endpoint: 'https://us1.prisma.sh/nest-f6ec12/prisma/dev',
      debug: false,
    });
  }
}
```

Once `PrismaService` is ready, we need to create a corresponding module.

prisma.module.ts JS

```
import { Module } from '@nestjs/common';
import { PrismaService } from './prisma.service';

@Module({
  providers: [PrismaService],
  exports: [PrismaService],
})
export class PrismaModule {}
```

> **HINT**
> To create new modules and services in no time we can make use of the **Nest CLI**. To create a `PrismaModule` type `nest g module prisma` and for the service `nest g service prisma`

Usage

To use your new service we are going to import the `PrismaModule` and inject the `PrismaService` into `UsersResolver`.

users.module.ts JS

```
import { Module } from '@nestjs/common';
import { UsersResolver } from './users.resolver';
import { PrismaModule } from '../prisma/prisma.module';

@Module({
  imports: [PrismaModule],
  providers: [UsersResolver],
})
export class UsersModule {}
```

Importing `PrismaModule` makes exported `PrismaService` available in the `UsersModule` context.

users.resolver.ts JS

```
import { Query, Resolver, Args, Info } from '@nestjs/graphql';
import { PrismaService } from '../prisma/prisma.service';
import { User } from '../graphql.schema';

@Resolver()
export class UsersResolver {
  constructor(private readonly prisma: PrismaService) {}

  @Query('users')
  async getUsers(@Args() args, @Info() info): Promise<User[]> {
    return await this.prisma.query.users(args, info);
  }
}
```

Example

A slightly modified example is available **here**.

Support us

Nest is an MIT-licensed open source project. It can grow thanks to the support by these awesome people. If you'd like to

join them, please read more here.

Principal Sponsor

Sponsors / Partners

Designed by Jakub Staroń, hosted by Netlify