Nestjs and Mysql in 5 Minutes

A quick tutorial for building scalable Node.js applications

Shai Ben Shimol Jan 26, 2019 · 4 min read

In this tutorial I will show you how easy is to setup and execute a Netsjs project using the following tech stack:

- **Nestjs** (Modules, Controllers, Repositories , TypeORM and Entities)
- **Node.js & NPM:** (https://nodejs.org/en/download/)
- **Mysql 5.7:** (https://dev.mysql.com/downloads/mysql/5.7.html)

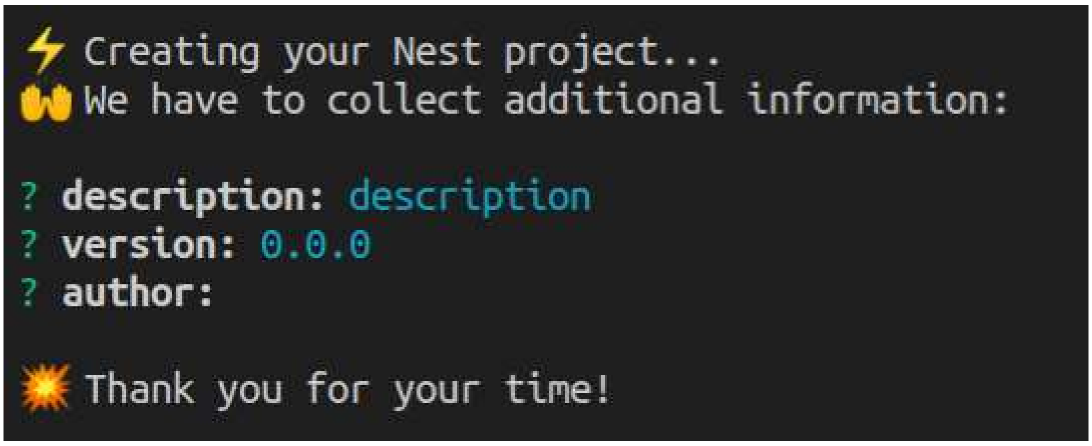
Why Nest js? · Issue #693 · nestjs/nest

Whats Different about nest to another frameworks? Why we have to use nest.js? Please say your opinion.

github.com

Install and create Nestjs project

```
npm install -g typescript
npm install -g @nestjs/cli
```

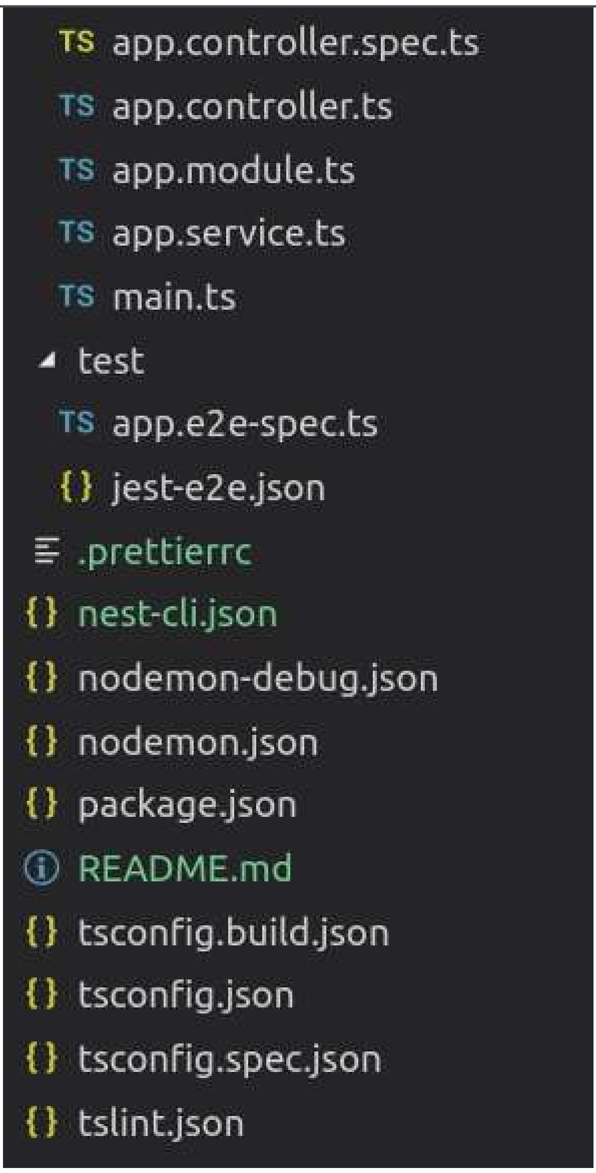


```
⚡ Creating your Nest project...
🙌 We have to collect additional information:

? description: description
? version: 0.0.0
? author:

💥 Thank you for your time!
```

```
nest new my-nest-project

c:\\> cd my-nest-project
```

Open "my-nest-project" in vscode with code . command

Project Structure



```
    TS app.controller.spec.ts
    TS app.controller.ts
    TS app.module.ts
    TS app.service.ts
    TS main.ts
  ◢ test
    TS app.e2e-spec.ts
    {} jest-e2e.json
  .prettierrc
{} nest-cli.json
{} nodemon-debug.json
{} nodemon.json
{} package.json
ⓘ README.md
{} tsconfig.build.json
{} tsconfig.json
{} tsconfig.spec.json
{} tslint.json
```

- node_modules: include the packages modules
- src: include the app source files
- test: end-to-end test app
- nest-cli.json: The root level of an Netsjs workspace provides workspace-wide and project-specific configuration defaults for build and development tools provided by the Nestjs. Path values given in the configuration are relative to the root workspace folder.
- packge.json: lists the packages your project depends on.

Every Nestjs app has at least one @Module() class — root module. The root @Module() for an app is so named because it can include child @Module() in A hierarchy of any depth.

The most important properties are as follows:

- **imports:** other modules whose exported classes are needed by component templates declared in *this Module*.
- **controllers:** the set of controllers which have to be created.
- **providers**: creators of services that this NgModule contributes to the global collection of services; they become accessible in all parts of the app.
- **exports:** the subset of declarations that should be visible and usable in the *component templates* of other Modules.

Services

Service is a layer category encompassing any value, function, or feature that an app needs. A service is typically a class with a narrow, well-defined purpose. It should do something specific and do it well.

Controllers

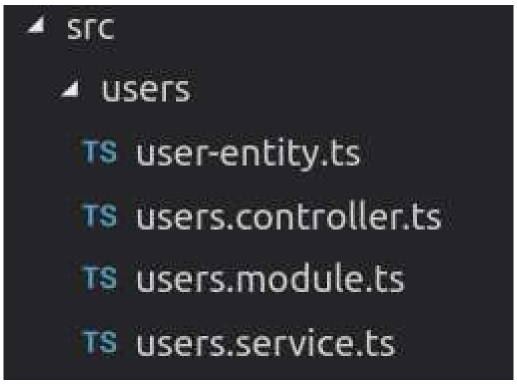
A controller is a class that handles HTTP requests. The public methods of the Controller are called action methods or simply actions. When the Nestjs Framework receives a request, it routes the request to an action.

To determine which action to invoke, the framework uses a routing table.

Let's Get Started

Create users module, service and controller.

```
nest g module users
nest g service users
```



src
users
user-entity.ts
users.controller.ts
users.module.ts
users.service.ts

In this section we'll not use spec file.

Installing Mysql And typeORM

```
npm install --save @nestjs/typeorm typeorm mysql
```

Time to write some code!

- Open user.entity.ts file and Type

```
import { Entity, Column, PrimaryGeneratedColumn } from 'typeorm';

@Entity()
export class User {

    @PrimaryGeneratedColumn()
    id: number;

    @Column({ length: 25 })
    fullName:string;

    @Column('date')
    birthday:Date;

    @Column()
    isActive:boolean;
}
```

user.entity.ts hosted with ♡ by GitHub view raw

```
import { Injectable, Inject } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from './user-entity';

@Injectable()
export class UsersService {

    constructor(@InjectRepository(User) private usersRepository: Repository<User>) { }

    async getUsers(user: User): Promise<User[]> {
        return await this.usersRepository.find();
    }

    async getUser(_id: number): Promise<User[]> {
        return await this.usersRepository.find({
            select: ["fullName", "birthday", "isActive"],
            where: [{ "id": _id }]
        });
    }

    async updateUser(user: User) {
        this.usersRepository.save(user)
    }

    async deleteUser(user: User) {
        this.usersRepository.delete(user);
    }
}
```

users.service.ts hosted with ♡ by GitHub view raw

- Open users.controller.ts and type

```
import { Controller, Post, Body, Get, Put, Delete,Param} from '@nestjs/common';
import { UsersService } from './users.service';
import { User } from './user.entity';

@Controller('users')
export class UsersController {

    constructor(private service: UsersService) { }
```

```
        return this.service.getUser(params.id);
    }

    @Post()
    create(@Body() user: User) {
        return this.service.createUser(user);
    }

    @Put()
    update(@Body() user: User) {
        return this.service.updateUser(user);
    }

    @Delete(':id')
    deleteUser(@Param() params) {
        return this.service.deleteUser(params.id);
    }
}
```

users.controller.ts hosted with ♡ by GitHub view raw

- Create ormconfig.json file in the root project with the following attributes

```
{
  "type": "mysql",
  "host": "localhost",
  "port": 3306,
  "username": "root",
  "password": "root",
  "database": "my_nestjs_project",
  "entities": ["src/**/**.entity{.ts,.js}"],
  "synchronize": true
}
```

- Open users.module.ts file and it looks like

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { UsersService } from './users.service';
import { UsersController } from './users.controller';
import { User } from './user-entity';
```

```
  providers: [UsersService],
  controllers: [UsersController],
})

export class UsersModule { }
```

users.ts hosted with ♡ by GitHub view raw

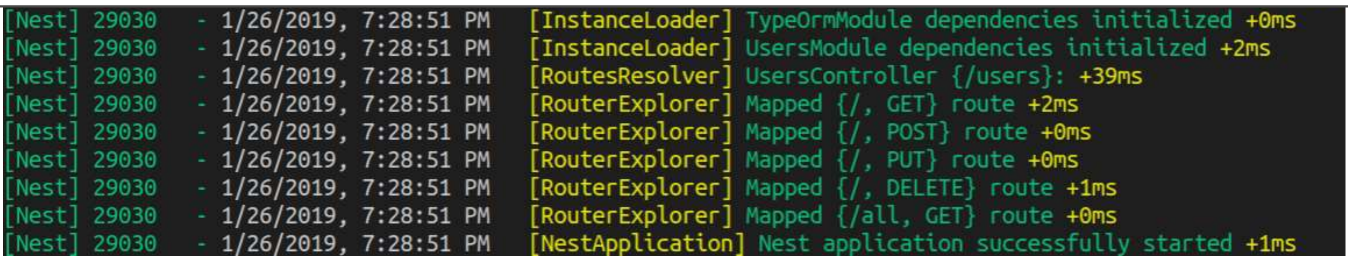
- Now open app.module.ts and import the database config file

```
import { Module } from '@nestjs/common';
import { UsersModule } from './users/users.module';
import { TypeOrmModule } from '@nestjs/typeorm';

@Module({
  imports: [
    TypeOrmModule.forRoot(),
    UsersModule
  ],
})
export class AppModule {}
```

app.module.ts hosted with ♡ by GitHub view raw

Open terminal in vscode and run

```
npm run start
```

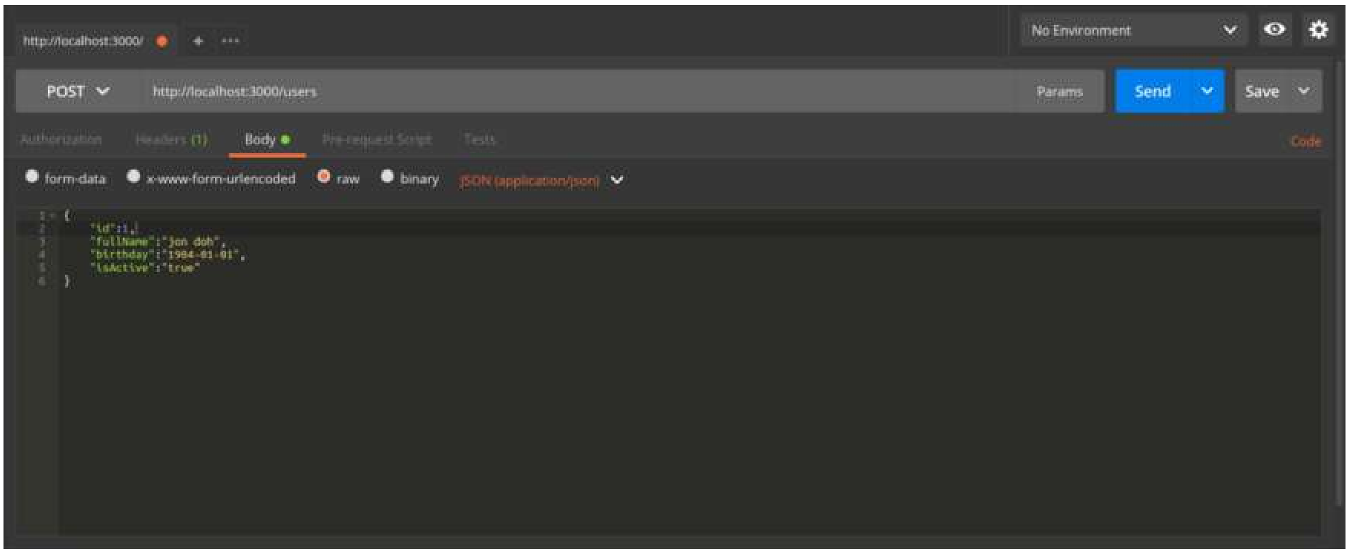


```
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [InstanceLoader] UsersModule dependencies initialized +2ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [RoutesResolver] UsersController {/users}: +39ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [RouterExplorer] Mapped {/, POST} route +0ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [RouterExplorer] Mapped {/, PUT} route +0ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [RouterExplorer] Mapped {/, DELETE} route +1ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [RouterExplorer] Mapped {/all, GET} route +0ms
[Nest] 29030   - 1/26/2019, 7:28:51 PM   [NestApplication] Nest application successfully started +1ms
```

Nest compilation

Testing via POSTMAN

- Create User

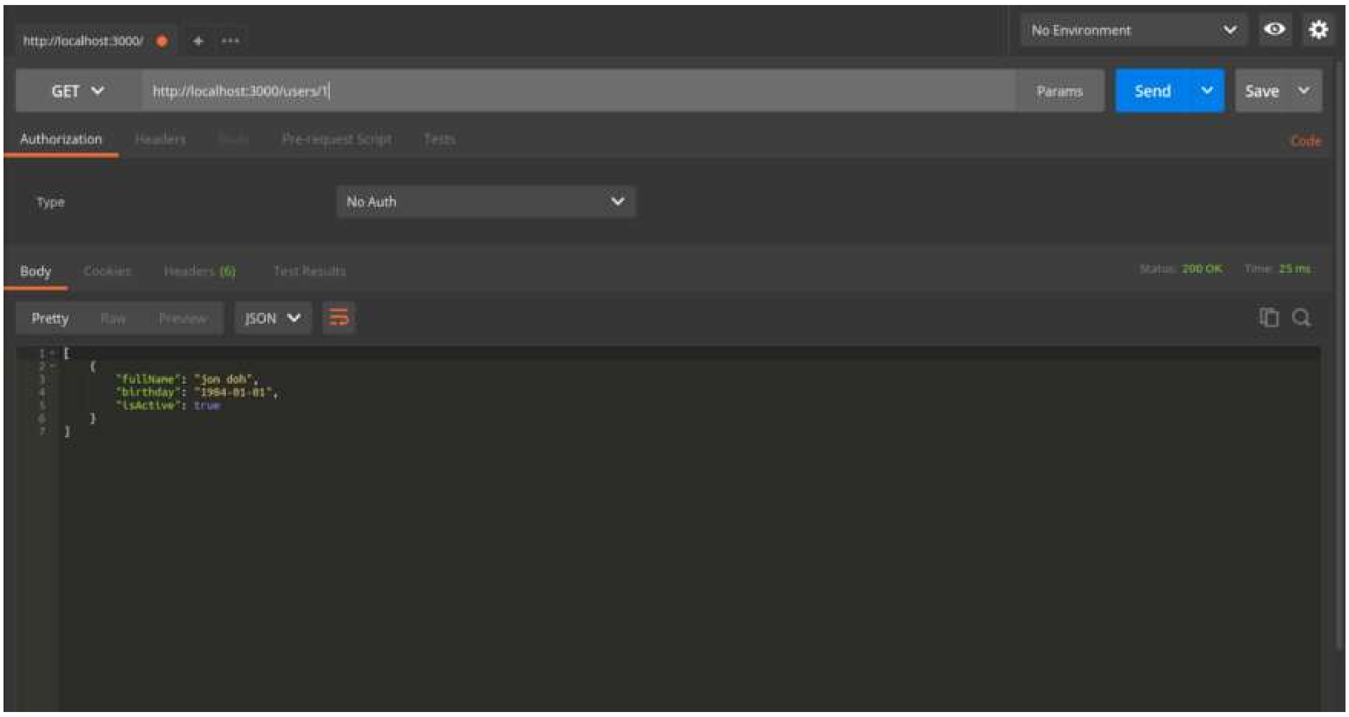


POST http://localhost:3000/users

raw JSON (application/json)

```
{
    "id":1,
    "fullName":"jon doh",
    "birthday":"1984-01-01",
    "isActive":"true"
}
```

- Get User by id



GET http://localhost:3000/users/1

Type: No Auth

Status: 200 OK Time: 25 ms

JSON

```
[
    {
        "fullName": "jon doh",
        "birthday": "1984-01-01",
        "isActive": true
    }
]
```

Conclusion

We made a good progress in this very first part of building Nestjs application. We got most of the architecture of Nestjs decisions.

My ko-fi

🍻Cheers! 🍻

👉 My Others Stories:

Build Effective Web Application

🔥With type-collector package 🔥

medium.com

Docker for Angular 7

Deploy Angular 7 to Docker Hub in 5 Minutes

medium.com

JavaScript Nodejs Nestjs Expressjs Typescript