

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет по
лабораторной работе № 1

Выполнил:
Борисов М. Е.

Группа:
К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

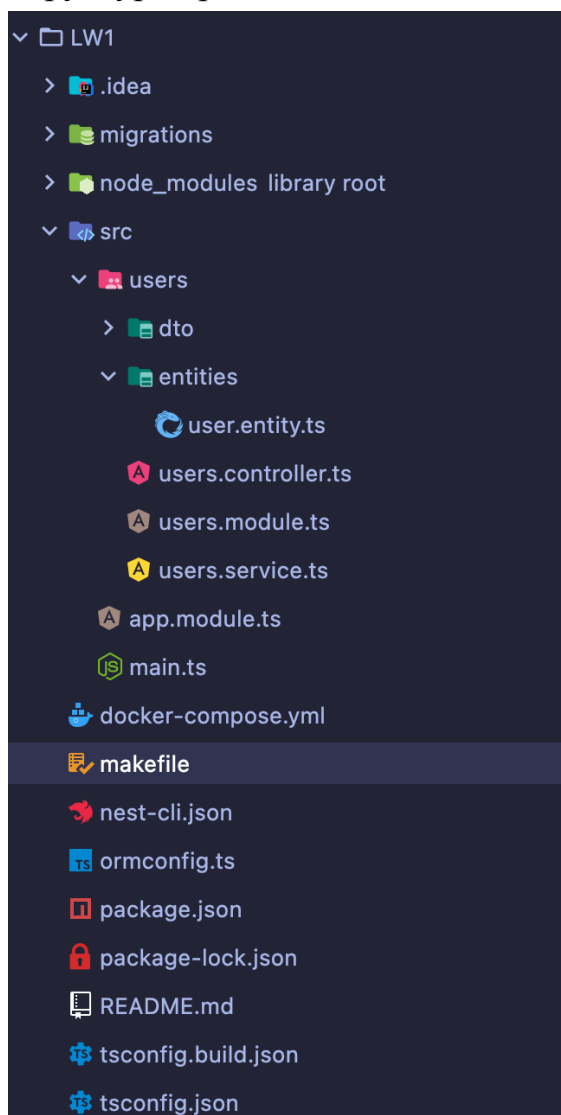
Задача

Написать свой boilerplate на express + sequelize + typescript. Должно быть явное разделение на:

1. Модели
2. Контроллеры
3. Роуты
4. Сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

Структура проекта



Зависимости проекта

```
package.json x
1 {
2   "name": "lw1",
3   "version": "0.0.1",
4   "description": "",
5   "author": "",
6   "private": true,
7   "license": "UNLICENSED",
8   "scripts": {"prebuild": "rimraf dist"...},
24  "dependencies": {"@nestjs/common": "^8.0.0"...},
39  "devDependencies": {
40    "@nestjs/cli": "^8.0.0",
41    "@nestjs/schematics": "^8.0.0",
42    "@nestjs/testing": "^8.0.0",
43    "@types/jest": "27.5.0",
44    "@types/node": "^16.0.0",
45    "@types/supertest": "^2.0.11",
46    "@typescript-eslint/eslint-plugin": "^5.0.0",
47    "@typescript-eslint/parser": "^5.0.0",
48    "eslint": "^8.0.1",
49    "eslint-config-prettier": "^8.3.0",
50    "eslint-plugin-prettier": "^4.0.0",
51    "jest": "28.0.3",
52    "prettier": "^2.3.2",
53    "source-map-support": "^0.5.20",
54    "supertest": "^6.1.3",
55    "ts-jest": "28.0.1",
56    "ts-loader": "^9.2.3",
57    "ts-node": "^10.0.0",
58    "tsconfig-paths": "4.0.0",
59    "typescript": "^4.3.5"
60  },
61  "jest": {
62    "moduleFileExtensions": [
63      "js",
64      "json",
65      "ts"
66    ],
67    "rootDir": "src",
68    "testRegex": ".*\\.spec\\.ts$",
69    "transform": {
70      "^.+\\.?(t|j)s$": "ts-jest"
71    },
72    "collectCoverageFrom": [
73      "**/*.?(t|j)s"
74    ],
75    "coverageDirectory": "../coverage",
```

В качестве ORM была выбрана TypeORM

Модель пользователя

```
user.entity.ts x
1  import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm'
2
3  @Entity()
4  export class User {
5      @PrimaryGeneratedColumn()
6      id!: number
7
8      @Column()
9      firstName!: string
10
11     @Column()
12     lastName!: string
13
14     @Column( options: {unique: true})
15     email!: string
16
17     @Column()
18     password!: string
19 }
20
```

Конфиг TypeORM

```
ormconfig.ts x
1  import 'dotenv/config'
2
3  export = {
4      host: process.env.DB_HOST,
5      type: `postgres`,
6      port: 5432,
7      username: process.env.POSTGRES_USER,
8      password: process.env.POSTGRES_PASSWORD,
9      database: process.env.POSTGRES_USER,
10     entities: [`dist/src/**/*.entity.js`],
11     migrations: [`dist/migrations/*.ts`],
12     cli: {
13         migrationsDir: `migrations`,
14     },
15     synchronize: false,
16 }
```

Модель

```
users.service.ts x
1 import { Injectable } from '@nestjs/common'
2 import { CreateUserDto } from '../dto/create-user.dto'
3 import { UpdateUserDto } from '../dto/update-user.dto'
4 import { Repository } from 'typeorm'
5 import { User } from '../entities/user.entity'
6 import { InjectRepository } from '@nestjs/typeorm'
7
8 @Injectable()
9 export class UsersService {
10   constructor(
11     @InjectRepository(User)
12     private usersRepository: Repository<User>,
13   ) {}
14   async create(createUserDto: CreateUserDto) {
15     return this.usersRepository.findOneBy(
16       await this.usersRepository.save(createUserDto),
17     )
18   }
19
20   findAll() {
21     return this.usersRepository.find()
22   }
23
24   findOneById(id: number) {
25     return this.usersRepository.findOneBy( where: { id })
26   }
27
28   findOneByEmail(email: string) {
29     return this.usersRepository.findOneBy( where: { email })
30   }
31
32   async update(id: number, updateUserDto: UpdateUserDto) {
33     return this.usersRepository.findOneBy(
34       await this.usersRepository.save( entity: { id, ...updateUserDto }),
35     )
36   }
37
38   remove(id: number) {
39     return this.usersRepository.delete(id)
40   }
41 }
42
```

Контроллер

```
    Param,
    Delete,
  } from '@nestjs/common'
  import { UsersService } from '../users.service'
  import { CreateUserDto } from '../dto/create-user.dto'
  import { UpdateUserDto } from '../dto/update-user.dto'
  import { ApiTags } from '@nestjs/swagger'

  @ApiTags( tags: `users`)
  @Controller( prefix: 'users')
  export class UsersController {
    constructor(private readonly usersService: UsersService) {}

    @Post()
    create(@Body() createUserDto: CreateUserDto) {
      return this.usersService.create(createUserDto)
    }

    @Get()
    findAll() {
      return this.usersService.findAll()
    }

    @Get( path: `:${id}`)
    findOneById(@Param( property: 'id') id: string) {
      return this.usersService.findOneById(+id)
    }

    @Get( path: `/email/:email`)
    findOneByEmail(@Param( property: 'email') email: string) {
      return this.usersService.findOneByEmail(email)
    }

    @Patch( path: `:${id}`)
    update(@Param( property: 'id') id: string, @Body() updateUserDto: UpdateUserDto) {
      return this.usersService.update(+id, updateUserDto)
    }

    @Delete( path: `:${id}`)
    remove(@Param( property: 'id') id: string) {
      return this.usersService.remove(+id)
    }
  }
}
```

Так как работаем с NestJS нет необходимости отдельно описывать роуты, с

помощью декораторов @Get, @Post и @Controllers можно указывать префиксы для роута.

Проверим сервис

Создадим пользователя

POST create

POST login

GET get by email

GET list

+

...

Booking / users / create

POST

{{host}}/users

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

1

2

3

4

5

6

Body

Cookies

Headers (7)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

1

2

3

4

5

6

7

Выведем список всех пользователей

GET

{{host}}/users

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

Query Params

KEY	VAL
Key	Val

Body

Cookies

Headers (7)

Test Results (1/1)

Pretty

Raw

Preview

Visualize

JSON

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

Вывод

В ходе лабораторной работы был написан свой boilerplate на NestJS, TypeORM и typescript. Проверена работа CRUD-методов в postman.