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

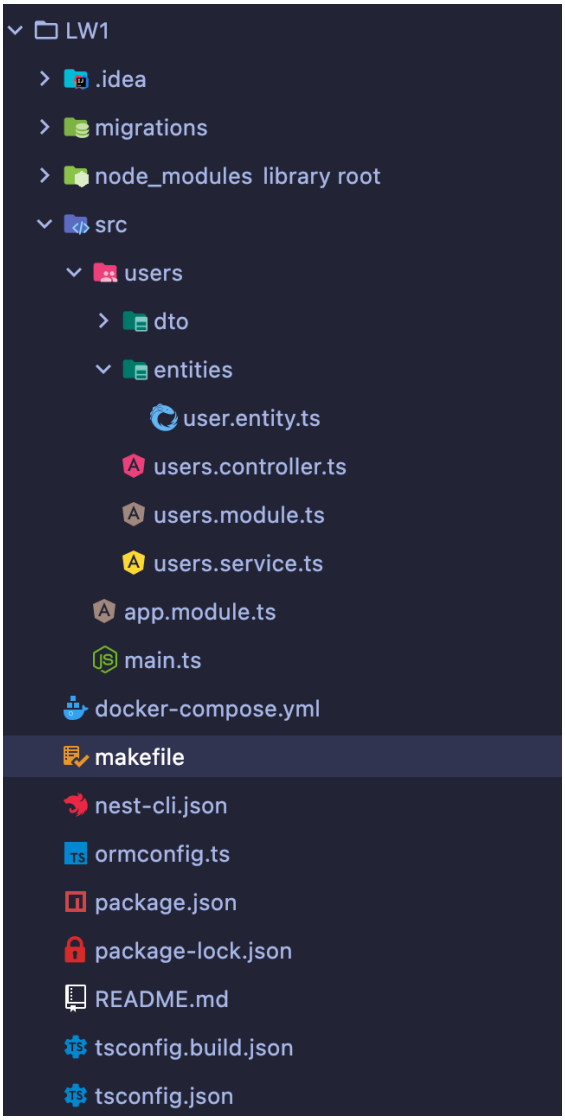
2. Контроллеры

3. Роуты

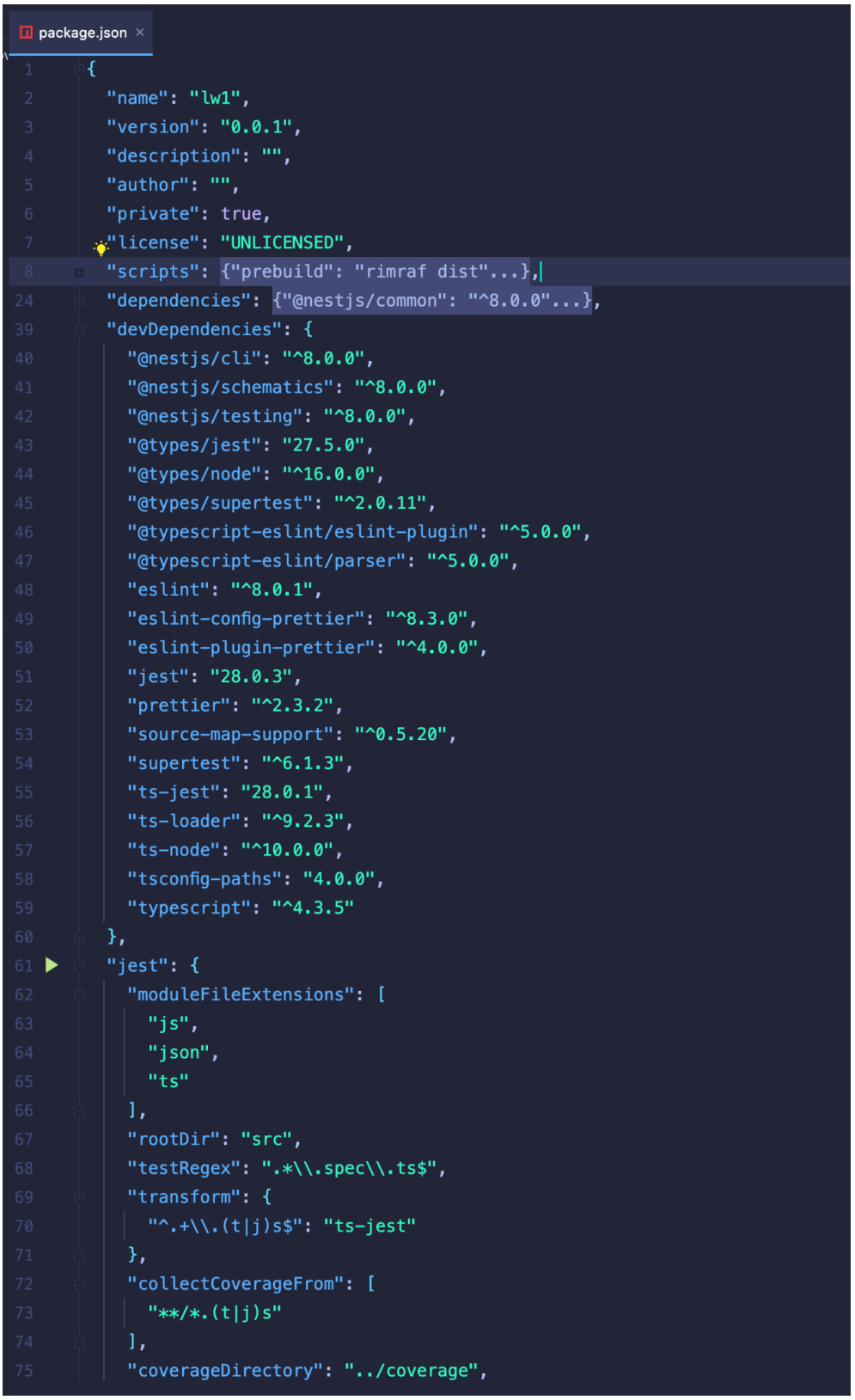
4. Сервисы для работы с моделями (реализуем паттерн “репозиторий”)

Ход работы

Структура проекта



Зависимости проекта



В качестве ORM была выбрана TypeORM

Модель пользователя

user.entity.ts

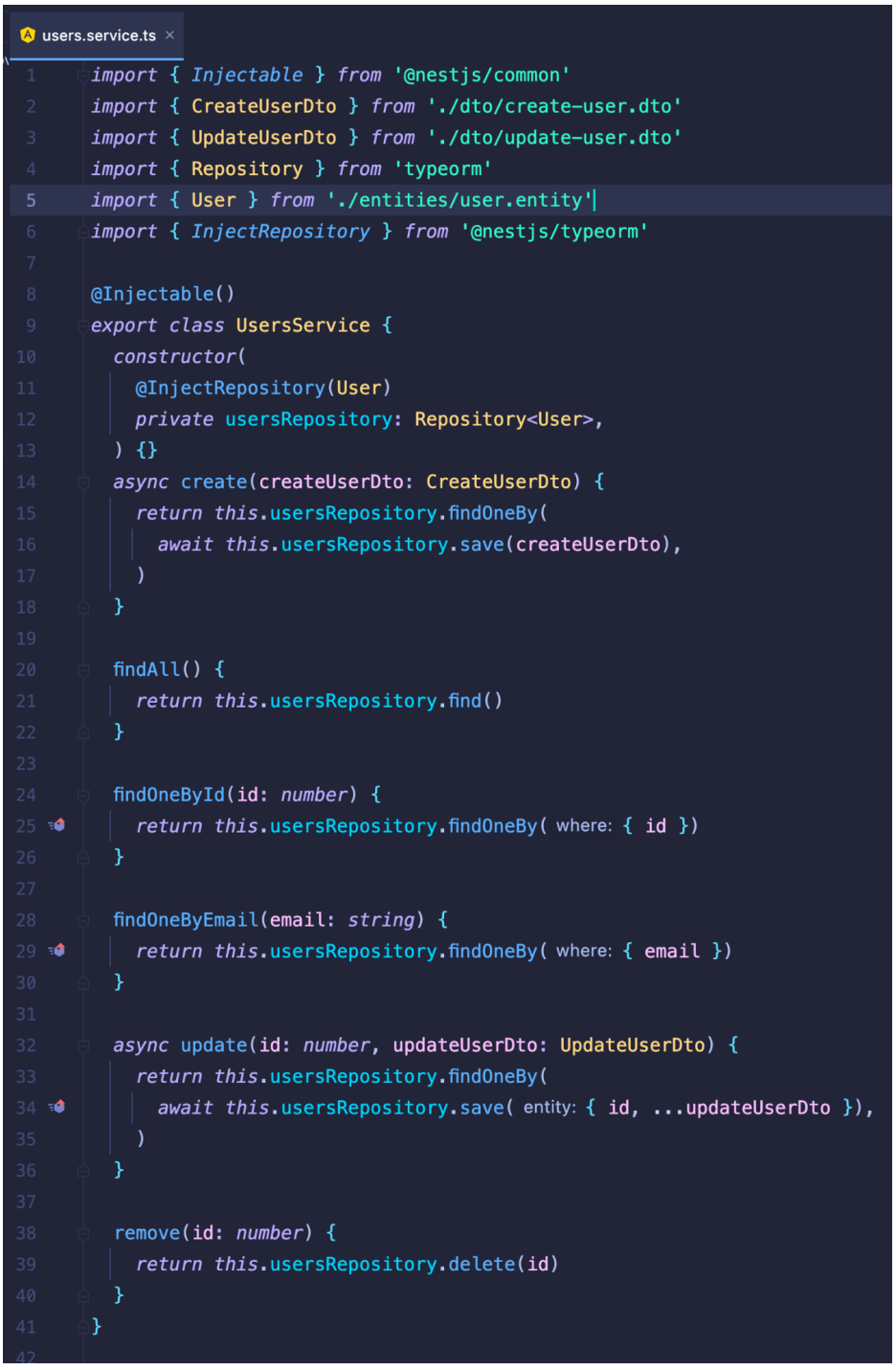
```ts
import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm'

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id!: number

  @Column()
  firstName!: string

  @Column()
  lastName!: string

  @Column( options: {unique: true})
  email!: string

  @Column()
  password!: string
}

```

Конфиг TypeORM

ormconfig.ts

```ts
import 'dotenv/config'

export = {
    host: process.env.DB_HOST,
    type: `postgres`,
    port: 5432,
    username: process.env.POSTGRES_USER,
    password: process.env.POSTGRES_PASSWORD,
    database: process.env.POSTGRES_USER,
    entities: [`dist/src/**/*.entity.js`],
    migrations: [`dist/migrations/*.ts`],
    cli: {
        migrationsDir: `migrations`,
    },
    synchronize: false,
}
```

Модель


users.service.ts
import { Injectable } from '@nestjs/common'
import { CreateUserDto } from './dto/create-user.dto'
import { UpdateUserDto } from './dto/update-user.dto'
import { Repository } from 'typeorm'
import { User } from './entities/user.entity'
import { InjectRepository } from '@nestjs/typeorm'
@Injectable()
export class UsersService {
constructor(
@InjectRepository(User)
private usersRepository: Repository<User>,
) {}
async create(createUserDto: CreateUserDto) {
return this.usersRepository.findOneBy(
await this.usersRepository.save(createUserDto),
)
}
findAll() {
return this.usersRepository.find()
}
findOneById(id: number) {
return this.usersRepository.findOneBy(where: { id })
}
findOneByEmail(email: string) {
return this.usersRepository.findOneBy(where: { email })
}
async update(id: number, updateUserDto: UpdateUserDto) {
return this.usersRepository.findOneBy(
await this.usersRepository.save(entity: { id, ...updateUserDto }),
)
}
remove(id: number) {
return this.usersRepository.delete(id)
}
}

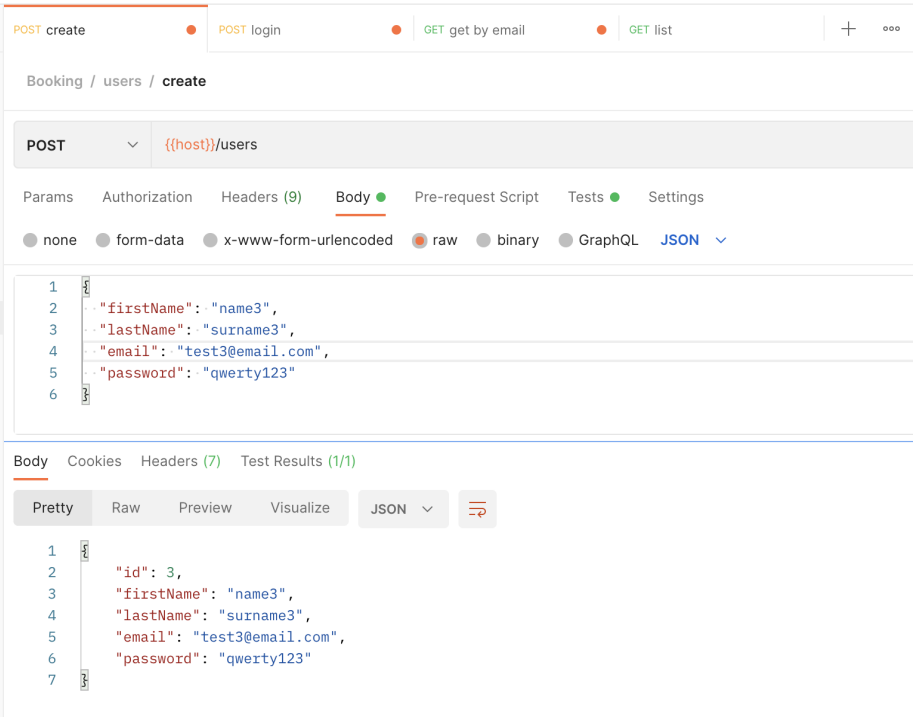
Контроллер

```
  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: UpdateUse
    return this.usersService.update(+id, updateUserDto)
  }

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

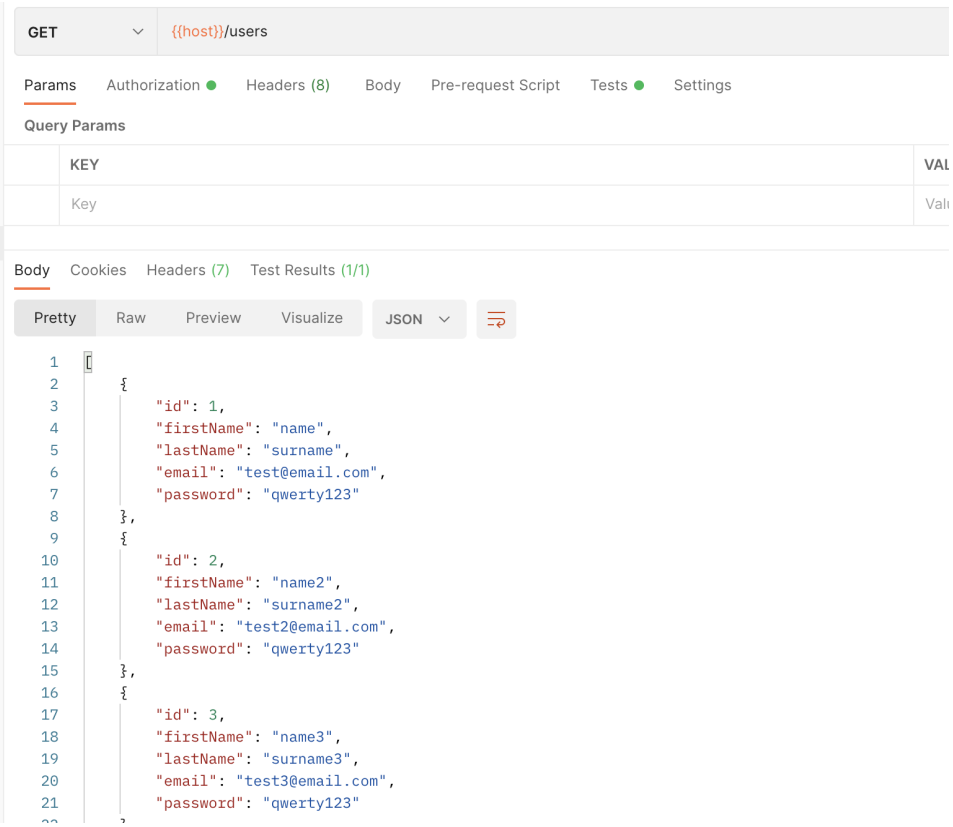
Так как работаем с NestJS нет необходимости отдельно описывать роуты, с

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

Проверим сервис
Создадим пользователя



Выведем список всех пользователей



Вывод

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