САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет Практическая работа №2

Выполнил:

Екушев Владислав

K33402

Проверил: Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

- 1. Продумать свою собственную модель пользователя
- 2. Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize
- 3. Написать запрос для получения пользователя по id/email

Ход работы

Будем использовать NestJS, Prisma и код из лабораторной работы №1.

Объявим модель пользователя с полями для email, пароля, JWT токена, набора заметок и тегов к ним.

```
💧 schema.prisma M 🗙
       generator client {
    provider = "prisma-client-js"
    5 datasource db {
   6 | provider = "postgresql"
7 | url = env("DATABASE_URL")
8 }
  You, 2 months ago | 1 author (You)

10 model User {

11 id Int @id @def

12 email String @unique

13 password String
                                          @id @default(autoincrement())
  14 refreshToken String?
  15 notes Note[]
16 Tag Tag[]
17 }
  19 model Note {
20 id Int
21 title String
                                         @id @default(autoincrement())
  22 markdown String @default("")
         tags Tag[]
dateCreated DateTime @default(now())
dateUpdated DateTime @default(now())
User User @relation(fields: [userId], references: [id])
userId Int
  30 model Tag {
  id Int @id @default(autoincrement())

label String

icon String?

notes Note[]

User User? @relation(fields: [userId], references: [id])
         userId Int?
```

Код для сервиса пользователя в бекенде на NestJS:

```
@Injectable()
export class UserService {
 private readonly logger = new Logger('UserSerivce')
  constructor(private prisma: PrismaService) {}
  async user(
   userWhereUniqueInput: Prisma.UserWhereUniqueInput
  ): Promise<User | null> {
   return await this.prisma.user.findUnique({
      where: \ user {\tt WhereUniqueInput},
   })
  async users(params: {
   skip?: number
    take?: number
   cursor?: Prisma.UserWhereUniqueInput
   where?: Prisma.UserWhereInput
   orderBy?: Prisma.UserOrderByWithRelationInput
  }): Promise<User[]> {
   const { skip, take, cursor, where, orderBy } = params
    return await this.prisma.user.findMany({
      skip,
     take,
     cursor,
     where,
     orderBy,
   })
  async createUser(data: Prisma.UserCreateInput): Promise<User> {
   return await this.prisma.user.create({
     data,
   })
  async updateUser(params: {
   where: Prisma.UserWhereUniqueInput
   data: Prisma.UserUpdateInput
  }): Promise<User> {
   const { where, data } = params
    return await this.prisma.user.update({
     data,
     where.
   })
  async deleteUser(where: Prisma.UserWhereUniqueInput): Promise<User> {
    return await this.prisma.user.delete({
     where.
   })
}
```

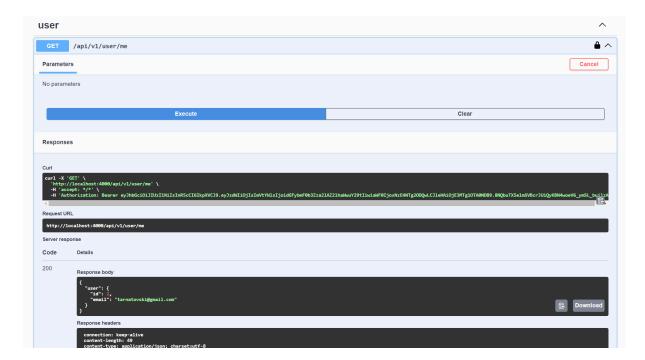
Представлены все CRUD операции для пользователя, следовательно, мы можем описать нужные по заданию ручки в контроллере:

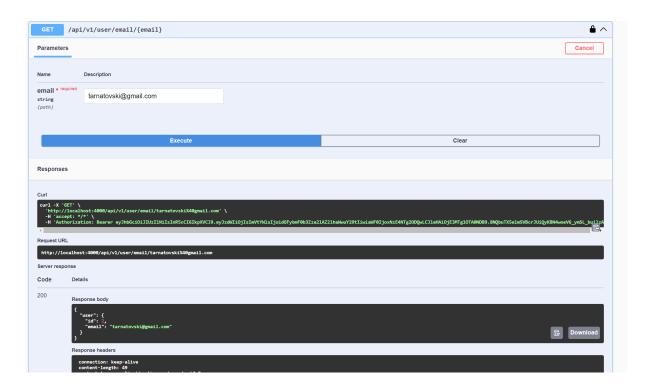
```
user.controller.ts M X
                                                                                                                                                          ▷ ₩ ↔
apps > backend > src > user > 

user.controller.ts > 

UserController > 

getByEmail
         import { Controller, Get, Param, Request, UseGuards } from '@nestjs/common'
        import { UserService } from './user.service'
import { RequestWithUser } from '../auth/auth.controller'
import { JwtAuthGuard } from '../auth/strategies/jwt.strategy'
import { ApiTags, ApiBearerAuth } from '@nestjs/swagger'
        import { UserGetSelfRes } from '@app/shared'
        You, 2 seconds ago | 1 author (You) @ApiTags('user') @Controller('user')
        export class UserController {
    constructor(private readonly userService: UserService) {}
           @UseGuards(JwtAuthGuard)
           @Get('me')
           @ApiBearerAuth()
           async getSelf(@Request() req: RequestWithUser): Promise<UserGetSelfRes> {
             const user = await this.userService.getSelf(req.user.userId)
           @UseGuards(JwtAuthGuard)
           @Get('email/:email')
           @ApiBearerAuth()
           async getByEmail(@Param('email') email: string): Promise<UserGetSelfRes> [
             const user = await this.userService.getByEmail(email)
```





Вывод

В ходе работы были получены навыки работы с Prisma и NestJS.