

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет
Практическая работа №2

Выполнил:

Екушев Владислав

К33402

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 x
prisma > schema.prisma > ...
You, 2 months ago | 1 author (You) | Generate
1 generator client {
2   provider = "prisma-client-js"
3 }
4
You, 2 months ago | 1 author (You)
5 datasource db {
6   provider = "postgresql"
7   url       = env("DATABASE_URL")
8 }
9
You, 2 months ago | 1 author (You)
10 model User {
11   id          Int    @id @default(autoincrement())
12   email       String @unique
13   password    String
14   refreshToken String?
15   notes       Note[]
16   Tag         Tag[]
17 }
18 You, 2 months ago * Initialize boilerplate
You, 2 months ago | 1 author (You)
19 model Note {
20   id          Int    @id @default(autoincrement())
21   title       String
22   markdown    String @default("")
23   tags        Tag[]
24   dateCreated DateTime @default(now())
25   dateUpdated DateTime @default(now())
26   User        User   @relation(fields: [userId], references: [id])
27   userId      Int
28 }
29
You, 2 months ago | 1 author (You)
30 model Tag {
31   id          Int    @id @default(autoincrement())
32   label       String
33   icon        String?
34   notes       Note[]
35   User        User?  @relation(fields: [userId], references: [id])
36   userId      Int?
37 }
38
```

Код для сервиса пользователя в бекенде на NestJS:

```
...

@Injectable()
export class UserService {
  private readonly logger = new Logger('UserService')

  constructor(private prisma: PrismaService) {}

  async user(
    userWhereUniqueInput: Prisma.UserWhereUniqueInput
  ): Promise<User | null> {
    return await this.prisma.user.findUnique({
      where: userWhereUniqueInput,
    })
  }

  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
You, 3 seconds ago | 1 author (You)
1 import { Controller, Get, Param, Request, UseGuards } from '@nestjs/common'
2 import { UserService } from '../user.service'
3 import { RequestWithUser } from '../auth/auth.controller'
4 import { JwtAuthGuard } from '../auth/strategies/jwt.strategy'
5 import { ApiTags, ApiBearerAuth } from '@nestjs/swagger'
6 import { UserGetSelfRes } from '@app/shared'
7
8 You, 2 seconds ago | 1 author (You)
9 @ApiTags('user')
10 @Controller('user')
11 export class UserController {
12   constructor(private readonly userService: UserService) {}
13
14   @UseGuards(JwtAuthGuard)
15   @Get('me')
16   @ApiBearerAuth()
17   async getSelf(@Request() req: RequestWithUser): Promise<UserGetSelfRes> {
18     const user = await this.userService.getSelf(req.user.userId)
19     return { user }
20   }
21
22   @UseGuards(JwtAuthGuard)
23   @Get('email/:email')
24   @ApiBearerAuth()
25   async getByEmail(@Param('email') email: string): Promise<UserGetSelfRes> {
26     const user = await this.userService.getByEmail(email)
27     return { user }
28   }
29 }
```



