

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ САНКТ-ПЕТЕРБУРГСКИЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
Факультет информационных технологий и программирования

Лабораторная работа №5

«Создание пользовательских представлений для дальнейшего связывания их с
интерфейсом»
по дисциплине «Web программирование»

Выполнил: студент группы М33122 Белоногов Е.В.

Реализован программный интерфейс для взаимодействия с сущностями users и comments.

Доступные для users методы

- create -- POST /users
- findAll -- GET /users
- findOneById -- GET /users/id/{id}
- findByName -- GET /users/name/{name}
- findByLogin -- GET /users/login/{login}
- remove -- DELETE /users/{id}/delete

Доступные для comments методы

- create -- POST /comments
- findAll -- GET /comments
- findOneById -- GET /comments/id/{id}
- findByUserId -- GET /comments/user/{id}
- remove -- DELETE /comments/{id}/delete

Контроллеры и провайдеры методов:

The screenshot shows a VS Code editor with a project named 'MY-BACKEND-PROJECT'. The Explorer sidebar on the left displays the file structure, including folders like 'src', 'dto', and 'shared', and various TypeScript files. The file 'TS users.controller.ts' is selected and open in the main editor. The breadcrumb navigation at the top of the editor indicates the path: 'src > users > TS users.controller.ts > UsersController > create'.

```
src > users > TS users.controller.ts > UsersController > create
1
2
3
4
5
6
7
8 @ApiTags('users')
9 @Controller('users')
10 export class UsersController {
11     constructor(private readonly userService: UsersService) {}
12
13     @ApiOperation({ summary: 'create new user' })
14     @ApiResponse({ status: 201, description: 'created' })
15     @ApiResponse({ status: 400, description: 'bad request' })
16     @ApiResponse({ status: 403, description: 'forbidden' })
17     @ApiResponse({ status: 501, description: 'not implemented' })
18     @Post()
19     async create(@Body() createUserDto: CreateUserDto): Promise<User> {
20         return await this.userService.create(createUserDto);
21     }
22
23     @ApiOperation({ summary: 'get all users' })
24     @ApiResponse({ status: 200, description: 'all users found' })
25     @ApiResponse({ status: 401, description: 'not authorized' })
26     @ApiResponse({ status: 403, description: 'forbidden' })
27     @ApiResponse({ status: 501, description: 'not implemented' })
28     @Get()
29     async findAll(): Promise<User[]> {
30         return await this.userService.findAll();
31     }
32
33     @ApiOperation({ summary: 'get user by id' })
34     @ApiResponse({ status: 200, description: 'user found' })
35     @ApiResponse({ status: 204, description: 'no content' })
36     @ApiResponse({ status: 400, description: 'bad request' })
37     @ApiResponse({ status: 401, description: 'not authorized' })
38     @ApiResponse({ status: 403, description: 'forbidden' })
39     @ApiResponse({ status: 501, description: 'not implemented' })
40     @Get('/:id/:id')
41     async findById(@Param('id', ParseIntPipe) id: string): Promise<User> {
42         return await this.userService.findById(id);
43     }
44
45     @ApiOperation({ summary: 'get user by name' })
46     @ApiResponse({ status: 200, description: 'user found' })
47     @ApiResponse({ status: 204, description: 'no content' })
48     @ApiResponse({ status: 401, description: 'not authorized' })
49     @ApiResponse({ status: 403, description: 'forbidden' })
50     @ApiResponse({ status: 501, description: 'not implemented' })
51     @Get('/:name/:name')
52     async findByName(@Param('name') name: string): Promise<User[]> {
53         return await this.userService.findByName(name);
54     }
55
56     @ApiOperation({ summary: 'get user by login' })
57     @ApiResponse({ status: 200, description: 'user found' })
58     @ApiResponse({ status: 204, description: 'no content' })
59     @ApiResponse({ status: 401, description: 'not authorized' })
60     @ApiResponse({ status: 403, description: 'forbidden' })
61     @ApiResponse({ status: 501, description: 'not implemented' })
62     @Get('/:login/:login')
63     async findByLogin(@Param('login') login: string): Promise<User> {
64         return await this.userService.findByLogin(login.toLowerCase());
65     }
66
67     @ApiOperation({ summary: 'delete user by id' })
68     @ApiResponse({ status: 200, description: 'user deleted' })
69     @ApiResponse({ status: 204, description: 'user not found' })
70     @ApiResponse({ status: 401, description: 'not authorized' })
71     @ApiResponse({ status: 403, description: 'forbidden' })
72     @ApiResponse({ status: 501, description: 'not implemented' })
73     @Delete('/:id/delete')
74     async remove(@Param('id', ParseIntPipe) id: number): Promise<void> {
75         return await this.userService.remove(id);
76     }
77
78
79
80
81
82
```

U

TS validation.pipe.ts
TS main.ts
TS comments.controller.ts M
TS create-user.dt

src > users > TS users.service.ts > UsersService

```

1  import { Injectable } from '@nestjs/common';
2  import { InjectRepository } from '@nestjs/typeorm';
3  import { Repository } from 'typeorm';
4  import { CreateUserDto } from '../dto/create-user.dto';
5  import { User } from '../user.entity';
6
7
8  @Injectable()
9  export class UsersService {
10     constructor(
11         @InjectRepository(User)
12         private readonly usersRepository: Repository<User>,
13     ) {}
14
15     create(createUserDto: CreateUserDto): Promise<User> {
16         // throw new NotImplementedException();
17
18         const user = new User();
19         user.name = createUserDto.name;
20         user.surname = createUserDto.surname;
21         user.login = createUserDto.login.toLowerCase();
22         // user.isOnline = true;
23         // user.password = createUserDto.password;
24
25         return this.usersRepository.save(user);
26     }
27
28     findAll(): Promise<User[]> {
29         return this.usersRepository.find();
30     }
31
32     findOneById(id: string): Promise<User> {
33         return this.usersRepository.findOne(id);
34     }
35
36     findByName(username: string): Promise<User[]> {
37         return this.usersRepository.find({ name: username });
38     }
39
40     findOneByLogin(login: string): Promise<User> {
41         return this.usersRepository.findOne({ login: login });
42     }
43
44     async remove(id: number): Promise<void> {
45         await this.usersRepository.delete(id);
46     }
47 }

```

```

8  @ApiTags('comments')
9  @Controller('comments')
10 export class CommentsController {
11     constructor(private readonly commentsService: CommentsService) {}
12
13
14     @ApiOperation({ summary: 'create new comment' })
15     @ApiResponse({ status: 201, description: 'created' })
16     @ApiResponse({ status: 400, description: 'bad request' })
17     @ApiResponse({ status: 401, description: 'not authorized' })
18     @ApiResponse({ status: 403, description: 'forbidden' })
19     @ApiResponse({ status: 501, description: 'not implemented' })
20     @Post()
21     async create(@Body() createCommentDto: CreateCommentDto): Promise<Comment> {
22         return await this.commentsService.create(createCommentDto);
23     }
24
25
26     @ApiOperation({ summary: 'get all comments' })
27     @ApiResponse({ status: 200, description: 'all comments found', type: Comment })
28     @ApiResponse({ status: 401, description: 'not authorized' })
29     @ApiResponse({ status: 403, description: 'forbidden' })
30     @ApiResponse({ status: 501, description: 'not implemented' })
31     @Get()
32     async findAll(): Promise<Comment[]> {
33         return await this.commentsService.findAll();
34     }
35
36
37     @ApiOperation({ summary: 'get comment by id' })
38     @ApiResponse({ status: 200, description: 'comment found', type: Comment })
39     @ApiResponse({ status: 204, description: 'no content' })
40     @ApiResponse({ status: 400, description: 'bad request' })
41     @ApiResponse({ status: 401, description: 'not authorized' })
42     @ApiResponse({ status: 403, description: 'forbidden' })
43     @ApiResponse({ status: 501, description: 'not implemented' })
44     @Get('/:id')
45     async findOneById(@Param('id', ParseIntPipe) id: number): Promise<Comment> {
46         return await this.commentsService.findOneById(id);
47     }
48
49
50     @ApiOperation({ summary: 'get comments by user id' })
51     @ApiResponse({ status: 200, description: 'comments found', type: Comment })
52     @ApiResponse({ status: 204, description: 'no content' })
53     @ApiResponse({ status: 400, description: 'bad request' })
54     @ApiResponse({ status: 401, description: 'not authorized' })
55     @ApiResponse({ status: 403, description: 'forbidden' })
56     @ApiResponse({ status: 501, description: 'not implemented' })
57     @Get('user/:id')
58     async findById(@Param('id', ParseIntPipe) user_id: number): Promise<Comment[]> {
59         return await this.commentsService.findById(user_id);
60     }
61
62
63     @ApiOperation({ summary: 'delete comment by id' })
64     @ApiResponse({ status: 200, description: 'comment deleted' })
65     @ApiResponse({ status: 204, description: 'comment not found' })
66     @ApiResponse({ status: 400, description: 'bad request' })
67     @ApiResponse({ status: 401, description: 'not authorized' })
68     @ApiResponse({ status: 403, description: 'forbidden' })
69     @ApiResponse({ status: 501, description: 'not implemented' })
70     @Delete('/:id/delete')
71     async remove(@Param('id', ParseIntPipe) id: number): Promise<void> {
72         return await this.commentsService.remove(id);
73     }

```

```

7
8  @Injectable()
9  export class CommentsService {
10     constructor(
11         @InjectRepository(Comment)
12         private commentsRepository: Repository<Comment>,
13     ) {}
14
15     create(createCommentDto: CreateCommentDto): Promise<Comment> {
16         const comment = new Comment();
17         comment.user_id = createCommentDto.user_id;
18         comment.comment = createCommentDto.comment;
19
20         return this.commentsRepository.save(comment);
21     }
22
23     findAll(): Promise<Comment[]> {
24         return this.commentsRepository.find();
25     }
26
27     findOneById(id: number): Promise<Comment> {
28         return this.commentsRepository.findOne(id);
29     }
30
31     findByUserId(user_id: number): Promise<Comment[]> {
32         return this.commentsRepository.find({ user_id: user_id });
33     }
34
35     async remove(id: number): Promise<void> {
36         await this.commentsRepository.delete(id);
37     }
38 }

```

В будущем, при реализации рабочей авторизации в приложении, будет добавлена реализация интерфейса для взаимодействия с passwords.

Ссылка на приложение:

<https://evgeny-backend-itmo.herokuapp.com/>

API:

<https://evgeny-backend-itmo.herokuapp.com/api/>