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»

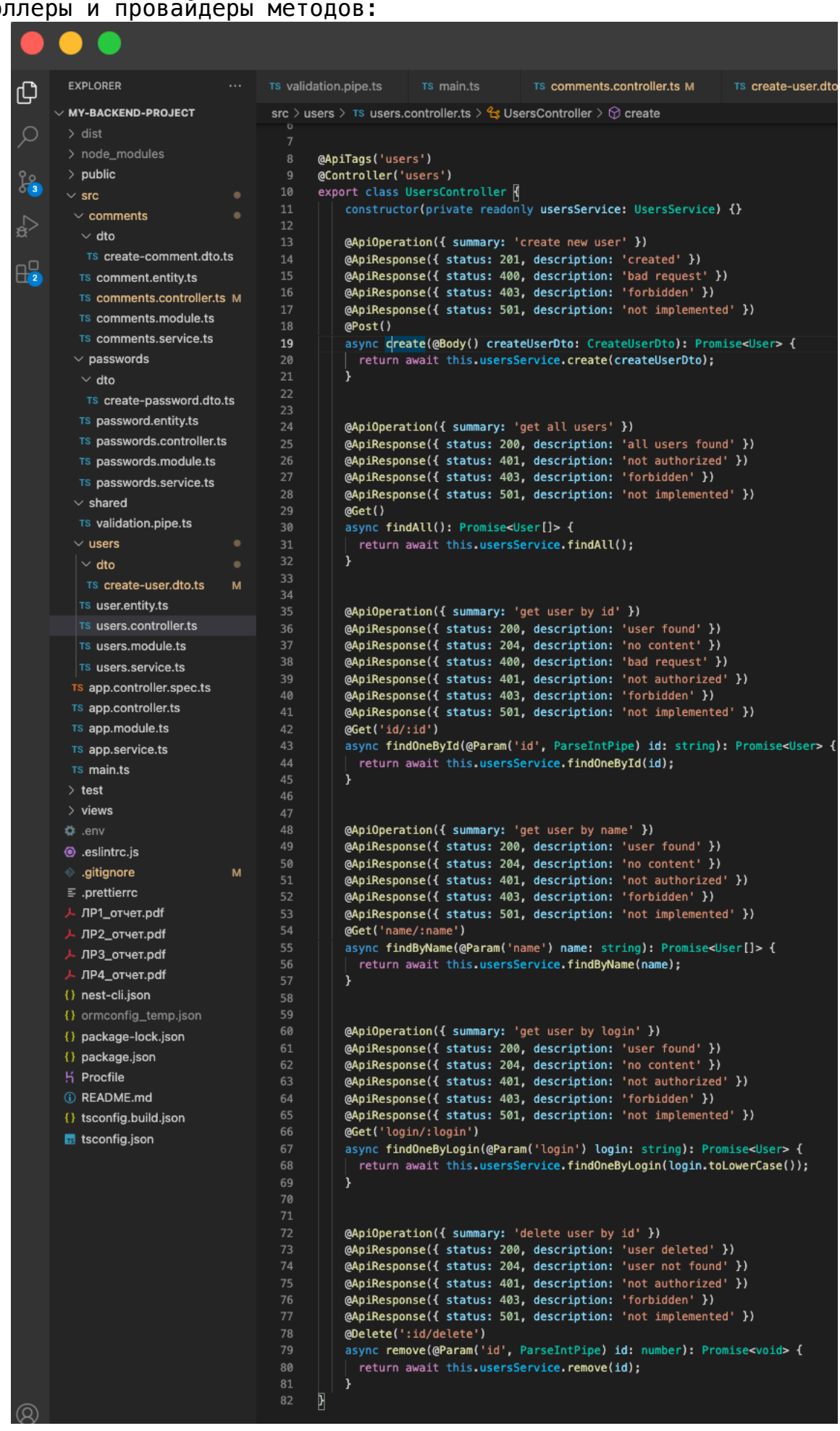
Выполнил: студент группы M33122 Белоногов Е.В.

Санкт-Петербург
2022

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:




validation.pipe.ts
main.ts
comments.controller.ts M
create-user.dt
src > users > users.service.ts > UsersService
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { CreateUserDto } from './dto/create-user.dto';
import { User } from './user.entity';
@Injectable()
export class UsersService {
constructor(
@InjectRepository(User)
private readonly usersRepository: Repository<User>,
) {}
create(createUserDto: CreateUserDto): Promise<User> {
// throw new NotImplementedException();
const user = new User();
user.name = createUserDto.name;
user.surname = createUserDto.surname;
user.login = createUserDto.login.toLowerCase();
// user.isOnline = true;
// user.password = createUserDto.password;
return this.usersRepository.save(user);
}
findAll(): Promise<User[]> {
return this.usersRepository.find();
}
findOneById(id: string): Promise<User> {
return this.usersRepository.findOne(id);
}
findByName(username: string): Promise<User[]> {
return this.usersRepository.find({ name: username });
}
findOneByLogin(login: string): Promise<User> {
return this.usersRepository.findOne({ login: login });
}
async remove(id: number): Promise<void> {
await this.usersRepository.delete(id);
}
}

src > comments > TS comments.controller.ts > CommentsController > findByUserId

```ts
@ApiTags('comments')
@Controller('comments')
export class CommentsController {
    constructor(private readonly commentsService: CommentsService) {}


    @ApiOperation({ summary: 'create new comment' })
    @ApiResponse({ status: 201, description: 'created' })
    @ApiResponse({ status: 400, description: 'bad request' })
    @ApiResponse({ status: 401, description: 'not authorized' })
    @ApiResponse({ status: 403, description: 'forbidden' })
    @ApiResponse({ status: 501, description: 'not implemented' })
    @Post()
    async create(@Body() createCommentDto: CreateCommentDto): Promise<Comment> {
        return await this.commentsService.create(createCommentDto);
    }


    @ApiOperation({ summary: 'get all comments '})
    @ApiResponse({ status: 200, description: 'all comments found', type: Comment })
    @ApiResponse({ status: 401, description: 'not authorized' })
    @ApiResponse({ status: 403, description: 'forbidden' })
    @ApiResponse({ status: 501, description: 'not implemented' })
    @Get()
    async findAll(): Promise<Comment[]> {
        return await this.commentsService.findAll();
    }


    @ApiOperation({ summary: 'get comment by id' })
    @ApiResponse({ status: 200, description: 'comment found', type: Comment })
    @ApiResponse({ status: 204, description: 'no content' })
    @ApiResponse({ status: 400, description: 'bad request' })
    @ApiResponse({ status: 401, description: 'not authorized' })
    @ApiResponse({ status: 403, description: 'forbidden' })
    @ApiResponse({ status: 501, description: 'not implemented' })
    @Get('id/:id')
    async findOneById(@Param('id', ParseIntPipe) id: number): Promise<Comment> {
        return await this.commentsService.findOneById(id);
    }


    @ApiOperation({ summary: 'get comments by user id' })
    @ApiResponse({ status: 200, description: 'comments found', type: Comment })
    @ApiResponse({ status: 204, description: 'no content' })
    @ApiResponse({ status: 400, description: 'bad request' })
    @ApiResponse({ status: 401, description: 'not authorized' })
    @ApiResponse({ status: 403, description: 'forbidden' })
    @ApiResponse({ status: 501, description: 'not implemented' })
    @Get('user/:id')
    async findByUserId(@Param('id', ParseIntPipe) user_id: number): Promise<Comment[]> {
        return await this.commentsService.findByUserId(user_id);
    }


    @ApiOperation({ summary: 'delete comment by id' })
    @ApiResponse({ status: 200, description: 'comment deleted' })
    @ApiResponse({ status: 204, description: 'comment not found' })
    @ApiResponse({ status: 400, description: 'bad request' })
    @ApiResponse({ status: 401, description: 'not authorized' })
    @ApiResponse({ status: 403, description: 'forbidden' })
    @ApiResponse({ status: 501, description: 'not implemented' })
    @Delete(':id/delete')
    async remove(@Param('id', ParseIntPipe) id: number): Promise<void> {
        return await this.commentsService.remove(id);
    }
}
```

```typescript
@Injectable()
export class CommentsService {
  constructor(
    @InjectRepository(Comment)
    private commentsRepository: Repository<Comment>,
  ) {}

  create(createCommentDto: CreateCommentDto): Promise<Comment> {
    const comment = new Comment();
    comment.user_id = createCommentDto.user_id;
    comment.comment = createCommentDto.comment;

    return this.commentsRepository.save(comment);
  }

  findAll(): Promise<Comment[]> {
    return this.commentsRepository.find();
  }

  findOneById(id: number): Promise<Comment> {
    return this.commentsRepository.findOne(id);
  }

  findByUserId(user_id: number): Promise<Comment[]> {
    return this.commentsRepository.find({ user_id: user_id });
  }

  async remove(id: number): Promise<void> {
    await this.commentsRepository.delete(id);
  }
}
```

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/