

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа

Выполнил:

Шайтор Илья

Группа К3339

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- Реализовать документацию средствами сваггер и постман
- |

Ход работы

Нужно сделать доку

Реализация в коде:

```
@ApiTags('Work Experiences')

@Controller('workExperiences')

export class WorkExperiencesController {

    constructor(private readonly workExperiencesService: WorkExperiencesService) {}

    @Get()

    @ApiOperation({ summary: 'Получить весь опыт работы' })

    @ApiResponse({ status: 200, description: 'Список опыта работы успешно получен' })

    @ApiBearerAuth()

    findAll() {

        return this.workExperiencesService.workExperienceFindAll();

    }

    @Get('/:id')

    @ApiOperation({ summary: 'Получить опыт работы по ID' })

    @ApiResponse({ status: 200, description: 'Опыт работы успешно получен' })

    @ApiResponse({ status: 404, description: 'Опыт работы не найден' })

    @ApiParam({ name: 'id', type: 'number', description: 'ID опыта работы' })

    getWorkExperience(@Param('id', ParseIntPipe) id: number) {

        return this.workExperiencesService.workExperienceGetById(id);

    }

}
```

```

@Post()

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Создать новый опыт работы' })

@ApiResponse({ status: 201, description: 'Опыт работы успешно создан' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiBody({ type: CreateWorkExperiencesDto })

@ApiBearerAuth()

create(@Body() dto: CreateWorkExperiencesDto) {

    return this.workExperiencesService.workExperienceCreate(dto);

}


@Put('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить опыт работы' })

@ApiResponse({ status: 200, description: 'Опыт работы успешно обновлен' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Опыт работы не найден' })

@ApiParam({ name: 'id', type: 'number', description: 'ID опыта работы' })

@ApiBearerAuth()

update(

    @Param('id', ParseIntPipe) id: number,

    @Body() dto: TUpdateWorkExperiencesDto,

) {

    return this.workExperiencesService.workExperienceUpdate(id, dto);

}


@Delete('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить опыт работы' })

```

```

@ApiResponse({ status: 200, description: 'Опыт работы успешно удален' })

@ApiResponse({ status: 404, description: 'Опыт работы не найден' })

@ApiParam({ name: 'id', type: 'number', description: 'ID опыта работы' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {

    return this.workExperiencesService.workExperienceDelete(id);

}

}

```

```

@ApiTags('Application')

@Controller('application')

export class ApplicationController {

    constructor(private readonly applicationsService: ApplicationService) {}

    @Get()

    @ApiOperation({ summary: 'Получить все заявки' })

    @ApiResponse({ status: 200, description: 'Список заявок успешно получен' })

    @ApiBearerAuth()

    findAll() {

        return this.applicationsService.applicationFindAll();

    }

    @Get('/:id')

    @ApiOperation({ summary: 'Получить заявку по ID' })

    @ApiResponse({ status: 200, description: 'Заявка успешно получена' })

    @ApiResponse({ status: 404, description: 'Заявка не найдена' })

    @ApiParam({ name: 'id', type: 'number', description: 'ID заявки' })

    getApplication(@Param('id', ParseIntPipe) id: number) {

        return this.applicationsService.applicationGetById(id);

    }

}

```

```

}

@Post()

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Создать новую заявку' })

@ApiResponse({ status: 201, description: 'Заявка успешно создана' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiBody({ type: CreateApplicationsDto })

@ApiBearerAuth()

create(@Body() dto: CreateApplicationsDto) {

    return this.applicationsService.applicationCreate(dto);

}

@Put('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить заявку' })

@ApiResponse({ status: 200, description: 'Заявка успешно обновлена' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Заявка не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID заявки' })

@ApiBearerAuth()

update(

    @Param('id', ParseIntPipe) id: number,

    @Body() dto: TUpdateApplicationsDto,

) {

    return this.applicationsService.applicationUpdate(id, dto);

}

@Delete('/:id')

```

```

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить заявку' })

@ApiResponse({ status: 200, description: 'Заявка успешно удалена' })

@ApiResponse({ status: 404, description: 'Заявка не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID заявки' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {

    return this.applicationsService.applicationDelete(id);

}

}

```

```

@ApiTags('Company')

@Controller('company')

export class CompanyController {

    constructor(private readonly companysService: CompanyService) {}

    @Get()

    @ApiOperation({ summary: 'Получить все компании' })

    @ApiResponse({ status: 200, description: 'Список компаний успешно получен' })

    @ApiBearerAuth()

    findAll() {

        return this.companysService.companyFindAll();

    }

    @Get('/:id')

    @ApiOperation({ summary: 'Получить компанию по ID' })

    @ApiResponse({ status: 200, description: 'Компания успешно получена' })

    @ApiResponse({ status: 404, description: 'Компания не найдена' })

```

```

@ApiParam({ name: 'id', type: 'number', description: 'ID компании' })

getUser(@Param('id', ParseIntPipe) id: number) {

    return this.companysService.companyGetById(id);

}


@Post()

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Создать новую компанию' })

@ApiResponse({ status: 201, description: 'Компания успешно создана' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiBody({ type: CreateCompanyDto })

@ApiBearerAuth()

create(@Body() dto: CreateCompanyDto) {

    return this.companysService.companyCreate(dto);

}


@Put('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить компанию' })

@ApiResponse({ status: 200, description: 'Компания успешно обновлена' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Компания не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID компании' })

@ApiBearerAuth()

update(

    @Param('id', ParseIntPipe) id: number,

    @Body() dto: TUpdateCompanyDto,

) {

    return this.companysService.companyUpdate(id, dto);
}

```

```

}

@Delete('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить компанию' })

@ApiResponse({ status: 200, description: 'Компания успешно удалена' })

@ApiResponse({ status: 404, description: 'Компания не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID компании' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {

    return this.companysService.companyDelete(id);

}

}

```

```

@ApiTags('Education')

@Controller('educations')

export class EducationController {

    constructor(private readonly educationsService: EducationService) {}

    @Get()

    @ApiOperation({ summary: 'Получить все записи об образовании' })

    @ApiResponse({ status: 200, description: 'Список записей об образовании успешно получен' })

    @ApiBearerAuth()

    findAll() {

        return this.educationsService.educationFindAll();

    }

    @Get('/:id')

```



```

@ApiOperation({ summary: 'Получить запись об образовании по ID' })

@ApiResponse({ status: 200, description: 'Запись об образовании успешно получена'
})

@ApiResponse({ status: 404, description: 'Запись об образовании не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID записи об образовании' })

getEducation(@Param('id', ParseIntPipe) id: number) {

    return this.educationsService.educationGetById(id);
}

@Post()

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Создать новую запись об образовании' })

@ApiResponse({ status: 201, description: 'Запись об образовании успешно создана'
})

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiBody({ type: CreateEducationsDto })

@ApiBearerAuth()

create(@Body() dto: CreateEducationsDto) {

    return this.educationsService.educationCreate(dto);
}

@Put('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить запись об образовании' })

@ApiResponse({ status: 200, description: 'Запись об образовании успешно обновлена'
})

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Запись об образовании не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID записи об образовании' })

@ApiBearerAuth()

update(

```

```

        @Param('id', ParseIntPipe) id: number,

        @Body() dto: TUpdateEducationsDto,
    ) {

        return this.educationsService.educationUpdate(id, dto);
    }

    @Delete('/:id')

    @UsePipes(new ValidationPipe())

    @ApiOperation({ summary: 'Удалить запись об образовании' })

    @ApiResponse({ status: 200, description: 'Запись об образовании успешно удалена'
    })

    @ApiResponse({ status: 404, description: 'Запись об образовании не найдена' })

    @ApiParam({ name: 'id', type: 'number', description: 'ID записи об образовании' })

    @ApiBearerAuth()

    delete(@Param('id', ParseIntPipe) id: number) {

        return this.educationsService.educationDelete(id);
    }
}

```

```

@ApiTags('Industry')

@Controller('industry')

export class IndustryController {

    constructor(private readonly industrysService: IndustryService) {}

    @Get()

    @ApiOperation({ summary: 'Получить все отрасли' })

    @ApiResponse({ status: 200, description: 'Список отраслей успешно получен' })

    @ApiBearerAuth()

    findAll() {

```

```

        return this.industrysService.industryFindAll();
    }

    @Get('/:id')

    @ApiOperation({ summary: 'Получить отрасль по ID' })

    @ApiResponse({ status: 200, description: 'Отрасль успешно получена' })

    @ApiResponse({ status: 404, description: 'Отрасль не найдена' })

    @ApiParam({ name: 'id', type: 'number', description: 'ID отрасли' })

    getIndustry(@Param('id', ParseIntPipe) id: number) {

        return this.industrysService.industryGetById(id);
    }

    @Post()

    @UsePipes(new ValidationPipe())

    @ApiOperation({ summary: 'Создать новую отрасль' })

    @ApiResponse({ status: 201, description: 'Отрасль успешно создана' })

    @ApiResponse({ status: 400, description: 'Неверные данные' })

    @ApiBody({ type: CreateIndustrysDto })

    @ApiBearerAuth()

    create(@Body() dto: CreateIndustrysDto) {

        return this.industrysService.industryCreate(dto);
    }

    @Put('/:id')

    @UsePipes(new ValidationPipe())

    @ApiOperation({ summary: 'Обновить отрасль' })

    @ApiResponse({ status: 200, description: 'Отрасль успешно обновлена' })

    @ApiResponse({ status: 400, description: 'Неверные данные' })

    @ApiResponse({ status: 404, description: 'Отрасль не найдена' })

```

```

@ApiParam({ name: 'id', type: 'number', description: 'ID отрасли' })

@ApiBearerAuth()

update(
    @Param('id', ParseIntPipe) id: number,
    @Body() dto: TUpdateIndustrysDto,
) {
    return this.industrysService.industryUpdate(id, dto);
}

@Delete('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить отрасль' })

@ApiResponse({ status: 200, description: 'Отрасль успешно удалена' })

@ApiResponse({ status: 404, description: 'Отрасль не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID отрасли' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {
    return this.industrysService.industryDelete(id);
}
}

```

```

@ApiTags('Resumes')

@Controller('resumes')

export class ResumeController {

    constructor(private readonly resumesService: ResumeService) {}

    @Get()

```

```

@ApiOperation({ summary: 'Получить все резюме' })

@ApiResponse({ status: 200, description: 'Список резюме успешно получен' })

@ApiBearerAuth()

findAll() {

    return this.resumesService.resumeFindAll();
}


@Get('/:id')

@ApiOperation({ summary: 'Получить резюме по ID' })

@ApiResponse({ status: 200, description: 'Резюме успешно получено' })

@ApiResponse({ status: 404, description: 'Резюме не найдено' })

@ApiParam({ name: 'id', type: 'number', description: 'ID резюме' })

getResume(@Param('id', ParseIntPipe) id: number) {

    return this.resumesService.resumeGetById(id);
}


@Post()

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Создать новое резюме' })

@ApiResponse({ status: 201, description: 'Резюме успешно создано' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiBody({ type: CreateResumesDto })

@ApiBearerAuth()

create(@Body() dto: CreateResumesDto) {

    return this.resumesService.resumeCreate(dto);
}


@Put('/:id')

```

```

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить резюме' })

@ApiResponse({ status: 200, description: 'Резюме успешно обновлено' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Резюме не найдено' })

@ApiParam({ name: 'id', type: 'number', description: 'ID резюме' })

@ApiBearerAuth()

update(

    @Param('id', ParseIntPipe) id: number,

    @Body() dto: TUpdateResumesDto,

) {

    return this.resumesService.resumeUpdate(id, dto);

}

@Delete('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить резюме' })

@ApiResponse({ status: 200, description: 'Резюме успешно удалено' })

@ApiResponse({ status: 404, description: 'Резюме не найдено' })

@ApiParam({ name: 'id', type: 'number', description: 'ID резюме' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {

    return this.resumesService.resumeDelete(id);

}

}

```

```

@ApiTags('Users')

@Controller('users')

```

```
export class UsersController {

    constructor(private readonly usersService: UsersService) {}

    @Get()

    @ApiOperation({ summary: 'Получить всех пользователей' })

    @ApiResponse({ status: 200, description: 'Список пользователей успешно получен' })

    @ApiBearerAuth()

    findAll() {

        return this.usersService.userFindAll();

    }

    @Get('/:id')

    @ApiOperation({ summary: 'Получить пользователя по ID' })

    @ApiResponse({ status: 200, description: 'Пользователь успешно получен' })

    @ApiResponse({ status: 404, description: 'Пользователь не найден' })

    @ApiParam({ name: 'id', type: 'number', description: 'ID пользователя' })

    getUser(@Param('id', ParseIntPipe) id: number) {

        return this.usersService.userGetById(id);

    }

    @Post()

    @UsePipes(new ValidationPipe())

    @ApiOperation({ summary: 'Создать нового пользователя' })

    @ApiResponse({ status: 201, description: 'Пользователь успешно создан' })

    @ApiResponse({ status: 400, description: 'Неверные данные' })

    @ApiBody({ type: CreateUsersDto })

    create(@Body() dto: CreateUsersDto) {

        return this.usersService.userCreate(dto);

    }

}
```

```

@Put('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить пользователя' })

@ApiResponse({ status: 200, description: 'Пользователь успешно обновлен' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Пользователь не найден' })

@ApiParam({ name: 'id', type: 'number', description: 'ID пользователя' })

@ApiBearerAuth()

update(
    @Param('id', ParseIntPipe) id: number,
    @Body() dto: TUpdateUsersDto,
) {
    return this.usersService.userUpdate(id, dto);
}

@Delete('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить пользователя' })

@ApiResponse({ status: 200, description: 'Пользователь успешно удален' })

@ApiResponse({ status: 404, description: 'Пользователь не найден' })

@ApiParam({ name: 'id', type: 'number', description: 'ID пользователя' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {
    return this.usersService.userDelete(id);
}

```



```

@Post('login')

@ApiOperation({ summary: 'Вход пользователя и получение JWT-токена' })

@ApiResponse({ status: 200, description: 'JWT-токен успешно получен' })

@ApiResponse({ status: 401, description: 'Неверный email или пароль' })

@ApiBody({ type: LoginDto })

async login(@Body() dto: LoginDto): Promise<{ token: string }> {

    const user = await this.usersService.validateUser(dto.email, dto.password);

    const token = this.usersService.generateToken(user);

    return { token };

}

}

```

```

@ApiTags('Vacancies')

@Controller('vacancys')

export class VacancyController {

    constructor(private readonly vacancysService: VacancyService) {}

    @Get()

    @ApiOperation({ summary: 'Получить все вакансии' })

    @ApiResponse({ status: 200, description: 'Список вакансий успешно получен' })

    @ApiBearerAuth()

    findAll() {

        return this.vacancysService.vacancyFindAll();

    }

    @Get('/:id')

    @ApiOperation({ summary: 'Получить вакансию по ID' })

    @ApiResponse({ status: 200, description: 'Вакансия успешно получена' })

    @ApiResponse({ status: 404, description: 'Вакансия не найдена' })

```

```

@ApiParam({ name: 'id', type: 'number', description: 'ID вакансии' })

getUser(@Param('id', ParseIntPipe) id: number) {

    return this.vacancysService.vacancyGetById(id);

}


@Post()

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Создать новую вакансию' })

@ApiResponse({ status: 201, description: 'Вакансия успешно создана' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiBody({ type: CreateVacancysDto })

@ApiBearerAuth()

create(@Body() dto: CreateVacancysDto) {

    return this.vacancysService.vacancyCreate(dto);

}


@Put('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Обновить вакансию' })

@ApiResponse({ status: 200, description: 'Вакансия успешно обновлена' })

@ApiResponse({ status: 400, description: 'Неверные данные' })

@ApiResponse({ status: 404, description: 'Вакансия не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID вакансии' })

@ApiBearerAuth()

update(

    @Param('id', ParseIntPipe) id: number,

    @Body() dto: TUpdateVacancysDto,

) {

    return this.vacancysService.vacancyUpdate(id, dto);
}

```

```

}

@Delete('/:id')

@UsePipes(new ValidationPipe())

@ApiOperation({ summary: 'Удалить вакансию' })

@ApiResponse({ status: 200, description: 'Вакансия успешно удалена' })

@ApiResponse({ status: 404, description: 'Вакансия не найдена' })

@ApiParam({ name: 'id', type: 'number', description: 'ID вакансии' })

@ApiBearerAuth()

delete(@Param('id', ParseIntPipe) id: number) {

    return this.vacancysService.userDelete(id);

}

}

```

Документация в сваггере:

Users				^
GET	/api/users	Получить всех пользователей	🔒	▼
POST	/api/users	Создать нового пользователя		▼
GET	/api/users/{id}	Получить пользователя по ID		▼
PUT	/api/users/{id}	Обновить пользователя	🔒	▼
DELETE	/api/users/{id}	Удалить пользователя	🔒	▼
POST	/api/users/login	Вход пользователя и получение JWT-токена		▼

Vacancies				^
GET	/api/vacancys	Получить все вакансии	🔒	▼
POST	/api/vacancys	Создать новую вакансию	🔒	▼
GET	/api/vacancys/{id}	Получить вакансию по ID		▼
PUT	/api/vacancys/{id}	Обновить вакансию	🔒	▼
DELETE	/api/vacancys/{id}	Удалить вакансию	🔒	▼

Work Experiences				^
GET	/api/workExperiences	Получить весь опыт работы	🔒	▼
POST	/api/workExperiences	Создать новый опыт работы	🔒	▼
GET	/api/workExperiences/{id}	Получить опыт работы по ID		▼
PUT	/api/workExperiences/{id}	Обновить опыт работы	🔒	▼
DELETE	/api/workExperiences/{id}	Удалить опыт работы	🔒	▼

Resumes				^
GET	/api/resumes	Получить все резюме	🔒	▼
POST	/api/resumes	Создать новое резюме	🔒	▼
GET	/api/resumes/{id}	Получить резюме по ID	🔒	▼
PUT	/api/resumes/{id}	Обновить резюме	🔒	▼
DELETE	/api/resumes/{id}	Удалить резюме	🔒	▼

Industry				^
GET	/api/industry	Получить все отрасли	🔒	▼
POST	/api/industry	Создать новую отрасль	🔒	▼
GET	/api/industry/{id}	Получить отрасль по ID		▼
PUT	/api/industry/{id}	Обновить отрасль	🔒	▼
DELETE	/api/industry/{id}	Удалить отрасль	🔒	▼

Вывод: В данной домашней работе была реализована документация в сваггере