

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ**

федеральное государственное автономное образовательное учреждение высшего
образования

Санкт-Петербургский национальный исследовательский университет информационных
технологий механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

Лабораторная работа №4

По дисциплине «Web-программирование»

Создание контроллеров страниц и спецификации

**Выполнил студент группы
М33122:**

Федотенко Николай Владимирович

Проверил:

Приискалов Роман Андреевич

САНКТ-ПЕТЕРБУРГ

2022

Цель работы:

Получение спецификации, содержащей все необходимые методы для работы с ранее созданной моделью, посредством выделения контроллеров, которые бы работали с доменной моделью, спроектированной в рамках предыдущей работы.

Краткие теоретические сведения:

- **Модуль** – это набор классов (контроллеров, сервисов, моделей и т. п.), решающих одну конкретную задачу/конкретный вариант использования.
- **Контроллер** – это компонент, отвечающих за обработку входящих **запросов** и возврат **ответов** клиенту. Для создания контроллера используется класс и декоратор.
- **CRUD** (*Create, Read, Update, Delete*) – это акроним, обозначающий 4 базовые функции, используемые при работе с базами данных.
- **OpenAPI спецификация** – это независимый от языка формат определения, используемый для описания RESTful API. Nest предоставляет специальный модуль, позволяющий генерировать такую спецификацию с помощью декораторов.

Ход выполнения работы:

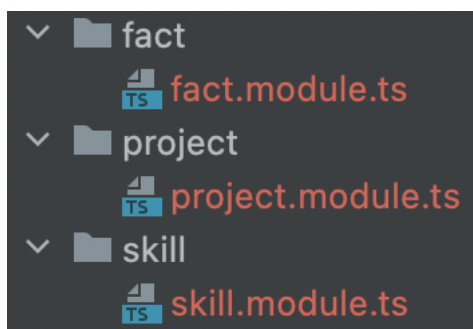
Данная лабораторная работа выполнена в операционной системе *macOS*.

Используемая IDE: *WebStorm 2021.3* (by JetBrains)

1. Генерация отдельных модулей:

```
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate module fact
CREATE fact/fact.module.ts (81 bytes)
UPDATE app.module.ts (840 bytes)
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate module skill
CREATE skill/skill.module.ts (82 bytes)
UPDATE app.module.ts (905 bytes)
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate module project
CREATE project/project.module.ts (84 bytes)
UPDATE app.module.ts (978 bytes)
```

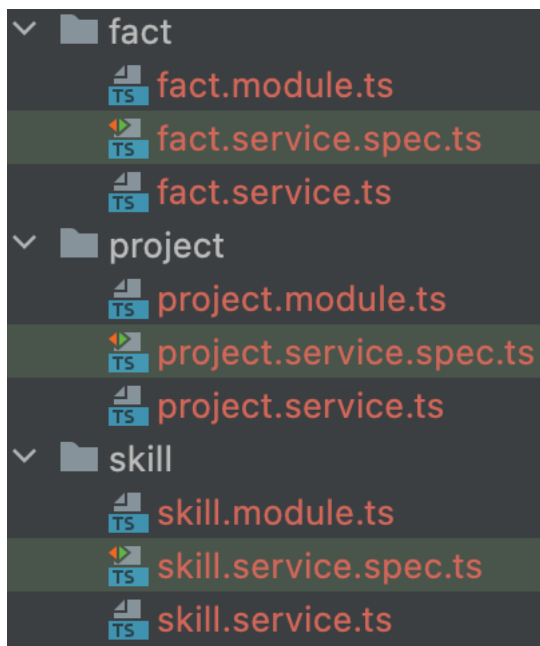
Убедимся, что модули сгенерированы:



2. Генерация сервисов:

```
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate service fact
CREATE fact/fact.service.spec.ts (446 bytes)
CREATE fact/fact.service.ts (88 bytes)
UPDATE fact/fact.module.ts (155 bytes)
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate service skill
CREATE skill/skill.service.spec.ts (453 bytes)
CREATE skill/skill.service.ts (89 bytes)
UPDATE skill/skill.module.ts (159 bytes)
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate service project
CREATE project/project.service.spec.ts (467 bytes)
CREATE project/project.service.ts (91 bytes)
UPDATE project/project.module.ts (167 bytes)
```

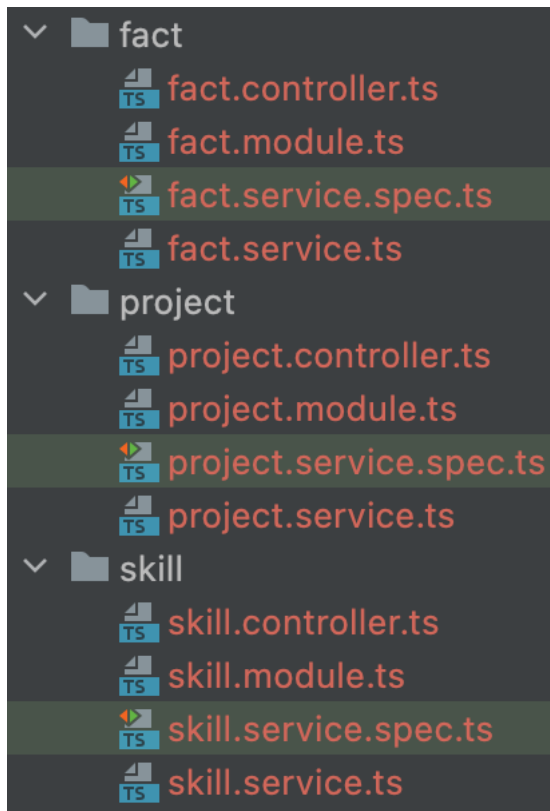
Убедимся, что сервисы сгенерированы:



3. Генерация контроллеров:

```
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate controller fact --no-spec
CREATE fact/fact.controller.ts (97 bytes)
UPDATE fact/fact.module.ts (240 bytes)
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate controller skill --no-spec
CREATE skill/skill.controller.ts (99 bytes)
UPDATE skill/skill.module.ts (247 bytes)
(base) kalkolay@MacBook-Pro-Nikolay src % nest generate controller project --no-spec
CREATE project/project.controller.ts (103 bytes)
UPDATE project/project.module.ts (261 bytes)
```

Убедимся, что контроллеры сгенерированы:



4. Реализация контроллеров:

fact.controller.ts:

```
import {
  Controller,
  Get,
  Post,
  Delete,
  Body,
  Param,
  NotImplementedException,
} from '@nestjs/common';
import { FactService } from '../fact.service';
import { Fact } from '@prisma/client';

@Controller('fact')
export class FactController {
  constructor(private factService: FactService) {}

  @Get('all')
  public async getAllFacts(): Promise<Fact[]> {
    throw new NotImplementedException();
  }

  @Post('create')
  // eslint-disable-next-line @typescript-eslint/no-unused-vars
  public async addFact(@Body() id: number, fact: string): Promise<void> {
    throw new NotImplementedException();
  }

  @Delete('/:id/delete')
  // eslint-disable-next-line @typescript-eslint/no-unused-vars
  public async deleteFactById(@Param('id') id: number): Promise<void> {
```

```

        throw new NotImplementedException();
    }
}

```

skill.controller.ts:

```

import {
    Controller,
    Get,
    Post,
    Delete,
    Body,
    Param,
    NotImplementedException,
} from '@nestjsjs/common';
import { SkillService } from '../skill.service';
import { Skill } from '@prisma/client';

@Controller('skill')
export class SkillController {
    constructor(private skillService: SkillService) {}

    @Get('all')
    public async getAllSkills(): Promise<Skill[]> {
        throw new NotImplementedException();
    }

    @Post('create')
    // eslint-disable-next-line @typescript-eslint/no-unused-vars
    public async addSkill(
        @Body() id: number,
        name = '',
        link: string,
    ): Promise<void> {
        throw new NotImplementedException();
    }

    @Delete('/:id/delete')
    // eslint-disable-next-line @typescript-eslint/no-unused-vars
    public async deleteSkillById(@Param('id') id: number): Promise<void> {
        throw new NotImplementedException();
    }
}

```

project.controller.ts:

```

import {
    Controller,
    Get,
    Post,
    Delete,
    Body,
    Param,
    NotImplementedException,
} from '@nestjsjs/common';
import { ProjectService } from '../project.service';
import { Project } from '@prisma/client';

@Controller('project')
export class ProjectController {
    constructor(private projectService: ProjectService) {}

    @Get('all')

```

```

public async getAllProjects(): Promise<Project[]> {
  throw new NotImplementedException();
}

@Post('create')
// eslint-disable-next-line @typescript-eslint/no-unused-vars
public async addProject(
  @Body() id: string,
  name: '',
  link: string,
): Promise<void> {
  throw new NotImplementedException();
}

@Delete(':id/delete')
// eslint-disable-next-line @typescript-eslint/no-unused-vars
public async deleteProjectById(@Param('id') id: number): Promise<void> {
  throw new NotImplementedException();
}
}

```

5. Настройка спецификации:

а. Установка Swagger:

```
npm install --save @nestjs/swagger swagger-ui-express
```

б. Подключение Swagger к проекту (*main.ts*):

```

const config = new DocumentBuilder()
  .setTitle('Nikolay Fedotenko: Portfolio')
  .setDescription('My Portfolio API description')
  .setVersion('1.0')
  .addTag('ITMO')
  .build();
const document = SwaggerModule.createDocument(app, config, {
  include: [FactModule, SkillModule, ProjectModule],
});
SwaggerModule.setup('api', app, document);

```

с. Модификация контроллеров:

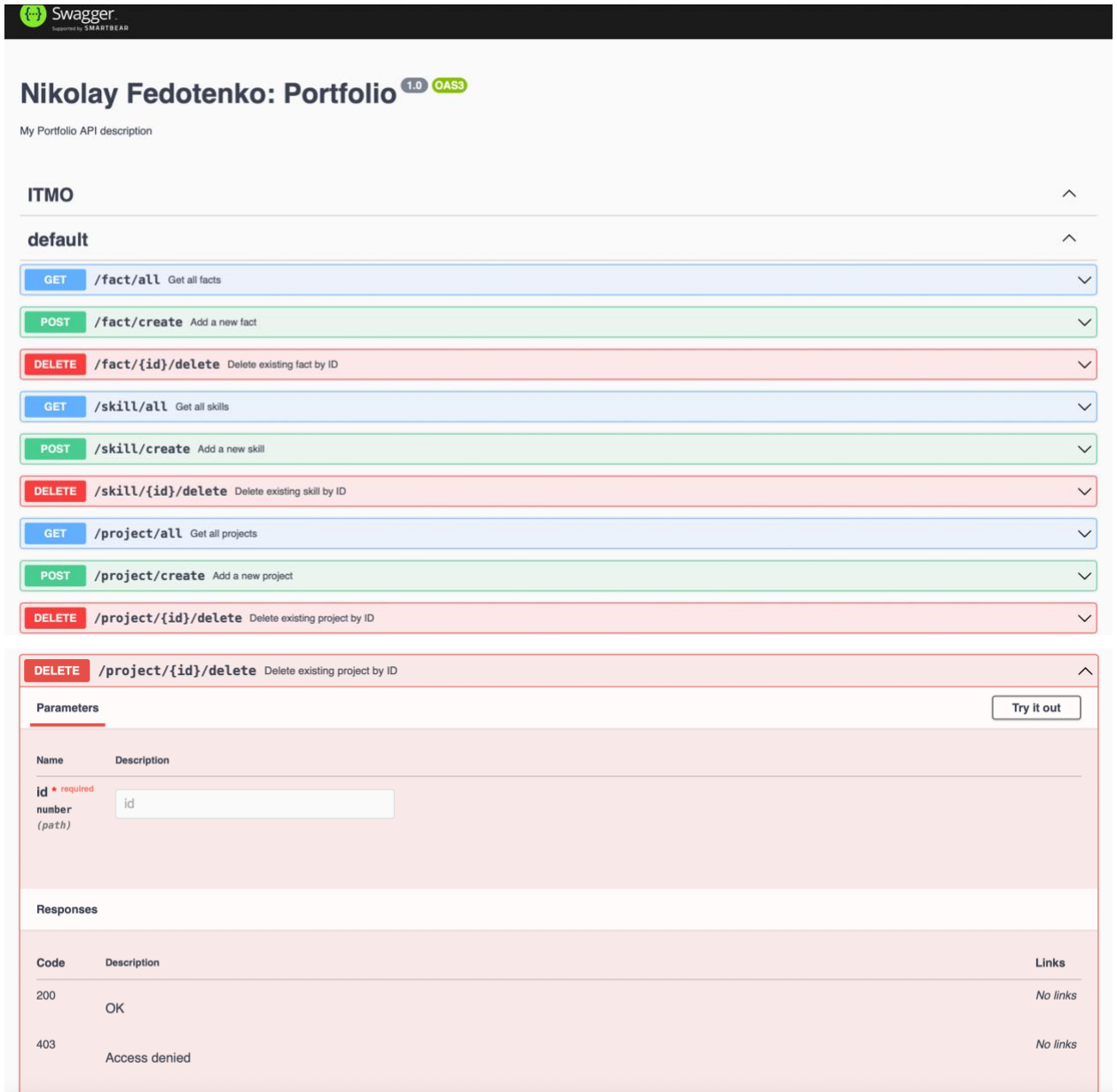
Пример (*fact.controller.ts*, остальное по аналогии):

```

@ApiOperation({
  summary: 'Delete existing fact by ID',
})
@ApiResponse({
  status: 200,
  description: 'OK',
})
@ApiResponse({
  status: 403,
  description: 'Access denied',
})
@Delete(':id/delete')
// eslint-disable-next-line @typescript-eslint/no-unused-vars
public async deleteFactById(@Param('id') id: number): Promise<void> {
  throw new NotImplementedException();
}

```

6. Сгенерированный результат (<http://itmo-web-fedotenko.herokuapp.com/api>):



The image shows a Swagger UI interface for an API titled "Nikolay Fedotenko: Portfolio". The API version is 1.0 and it is licensed under OAS3. The description is "My Portfolio API description". The interface is organized into sections: "ITMO" and "default". Under "default", there are several endpoints for facts, skills, and projects, each with GET, POST, and DELETE methods. The "DELETE /project/{id}/delete" endpoint is expanded, showing its parameters and responses.

ITMO

default

- GET** `/fact/all` Get all facts
- POST** `/fact/create` Add a new fact
- DELETE** `/fact/{id}/delete` Delete existing fact by ID
- GET** `/skill/all` Get all skills
- POST** `/skill/create` Add a new skill
- DELETE** `/skill/{id}/delete` Delete existing skill by ID
- GET** `/project/all` Get all projects
- POST** `/project/create` Add a new project
- DELETE** `/project/{id}/delete` Delete existing project by ID

DELETE `/project/{id}/delete` Delete existing project by ID

Parameters

Name	Description
id * required number (path)	id

Responses

Code	Description	Links
200	OK	No links
403	Access denied	No links

Вывод:

Я получил спецификацию, содержащую все необходимые методы для работы с ранее созданной моделью, посредством выделения контроллеров, которые работают с доменной моделью, спроектированной в рамках предыдущей работы.