МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИИ

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 №5

По дисциплине «Web-программирование»

Создание пользовательских представлений и дальнейшего связывания их с интерфейсом

Выполнил студент группы
***M33122*:**
Федотенко Николай Владимирович

Проверил:
Приискалов Роман Андреевич

САНКТ-ПЕТЕРБУРГ

2022

Цель работы:

Достижение работоспособности всех кнопок и замены зашитых в код данных на отображение реальных данных из базы данных, полученных путём отправки сетевых запросов на сервер.

Краткие теоретические сведения:

- **Guards** – это встроенные механизмы NestJS, имеющие единственную ответственность. Они определяют, будет ли запрос обработан обработчиком маршрута или нет, в зависимости от определённых условий, существующих во время выполнения.
- **Валидация** – это процесс проверки данных по критериям корректности и полезности для конкретного применения. NestJS предлагает *ValidationPipe* как инструмент для проверки входных данных.
- **DTO (*Data Transfer Object*)** – это один из шаблонов проектирования, используется для передачи данных между подсистемами приложения. Не должен содержать какого-либо поведения.
- **Фильтры исключений** – это встроенный в NestJS уровень исключений, который отвечает за обработку всех необработанных исключений в приложении. Когда исключение не обрабатывается кодом приложения, оно перехватывается данным уровнем, который затем автоматически отправляет соответствующий ответ.
- **Пагинация** – разделение большого массива данных, имеющихся на сайте, на отдельные страницы для удобства использования.

Ход выполнения работы:

Данная лабораторная работа выполнена в операционной системе *macOS*.

Используемая IDE: *WebStorm 2022.1* (by JetBrains)

1. Установка необходимых компонентов:

```
npm i --save class-validator class-transformer
```

2. Привязка ValidationPipe на уровне приложения (*main.ts*):

```
import { ValidationPipe } from '@nestjs/common';
…
app.useGlobalPipes(new ValidationPipe());
```

3. Реализация обёрток (сервисов) для взаимодействия через Prisma Client API:

prisma.service.ts

```ts
import { INestApplication, Injectable, OnModuleInit } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService extends PrismaClient implements OnModuleInit {
  async onModuleInit() {
    await this.$connect();
  }

  async enableShutdownHooks(app: INestApplication) {
    this.$on('beforeExit', async () => {
      await app.close();
    });
  }
}
```

fact.service.ts (аналогично выглядят сервисы *skill* и *project*):

```ts
import { Injectable } from '@nestjs/common';
import { PrismaService } from '../prisma.service';
import { Fact, Prisma } from '@prisma/client';

@Injectable()
export class FactService {
  constructor(private prisma: PrismaService) {}

  async fact(
    factWhereUniqueInput: Prisma.FactWhereUniqueInput,
  ): Promise<Fact | null> {
    return this.prisma.fact.findUnique({
      where: factWhereUniqueInput,
    });
  }

  async facts(params: {
    skip?: number;
    take?: number;
    cursor?: Prisma.FactWhereUniqueInput;
    where?: Prisma.FactWhereInput;
    orderBy?: Prisma.FactOrderByWithRelationInput;
  }): Promise<Fact[]> {
    const { skip, take, cursor, where, orderBy } = params;
    return this.prisma.fact.findMany({
      skip,
      take,
      cursor,
      where,
      orderBy,
    });
  }

  async createFact(data: Prisma.FactCreateInput): Promise<Fact> {
    return this.prisma.fact.create({
      data,
    });
  }
```

```
  async updateFact(params: {
    where: Prisma.FactWhereUniqueInput;
    data: Prisma.FactUpdateInput;
  }): Promise<Fact> {
    const { data, where } = params;
    return this.prisma.fact.update({
      data,
      where,
    });
  }

  async deleteFact(where: Prisma.FactWhereUniqueInput): Promise<Fact> {
    return this.prisma.fact.delete({
      where,
    });
  }
}
```

4. Реализация DTO:

```
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty } from 'class-validator';

export class FactDto {
  @ApiProperty( options: {
    description: 'Fact',
    example: 'Студент ИТМО',
  })
  @IsNotEmpty()
  readonly fact: string;
}
```

```
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty } from 'class-validator';

export class SkillDto {
  @ApiProperty( options: {
    description: 'Skill ID',
    example: 'one',
  })
  @IsNotEmpty()
  readonly id: string;

  @ApiProperty( options: {
    description: 'Skill name',
    example: 'C++',
  })
  readonly name: string;

  @ApiProperty( options: {
    description: 'Link to the skill logo',
    example:
      'https://camo.githubusercontent.com/341
  })
  @IsNotEmpty()
  readonly link: string;
}
```

```
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty } from 'class-validator';

export class ProjectDto {
  @ApiProperty( options: {
    description: 'Project name',
    example: 'Witcher 3',
  })
  readonly name: string;

  @ApiProperty( options: {
    description: 'Link to the project image',
    example:
      'https://ixbt.online/gametech/games/2021/01/31/CEgdlW9GHLsXlYnUXse3tNBWvu8EWiAMVvoUqCfI.jpg',
  })
  @IsNotEmpty()
  readonly link: string;
}
```

5. Модификация существующих контроллеров:

fact.controller.ts:

```
import {
  Controller,
  Get,
  Post,
  Delete,
  Body,
  Param,
  HttpException,
  HttpStatus,
} from '@nestjs/common';
import {
  ApiOperation,
  ApiOkResponse,
  ApiBadRequestResponse,
} from '@nestjs/swagger';
import { FactService } from './fact.service';
import { FactDto } from './dto/fact.dto';
import { Fact } from '@prisma/client';

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

  @ApiOperation({
    summary: 'Get all facts',
  })
  @ApiOkResponse({
    description: 'OK',
  })
  @Get('all')
  public async getAllFacts(): Promise<Fact[]> {
    return this.factService.facts({});
  }

  @ApiOperation({
    summary: 'Add a new fact',
  })
  @ApiOkResponse({
    description: 'Fact has been successfully added',
  })
  @Post('create')
  public async addFact(@Body() factData: FactDto): Promise<Fact> {
    const factToAdd = { fact: factData.fact };
    return await this.factService.createFact(factToAdd);
  }

  @ApiOperation({
    summary: 'Delete existing fact by ID',
  })
  @ApiOkResponse({
    description: 'Fact has been successfully deleted',
  })
  @ApiBadRequestResponse({
    description: 'Fact not found',
  })
  @Delete(':id/delete')
  public async deleteFactById(@Param('id') id: number): Promise<Fact> {
    const factToDelete = await this.factService.fact({ id: Number(id) });
    if (factToDelete == null) {
      throw new HttpException('Fact not found', HttpStatus.BAD_REQUEST);
```

```
    }
    return this.factService.deleteFact({ id: Number(id) });
  }
}
```

Аналогично модифицируем *project.controller.ts* и *skill.controller.ts*.

Также, подключаем Prisma в модули:

```
import { Module } from '@nestjs/common';
import { FactService } from './fact.service';
import { PrismaService } from '../prisma.service';
import { FactController } from './fact.controller';

@Module({
  providers: [FactService, PrismaService],
  controllers: [FactController],
})
export class FactModule {}
```

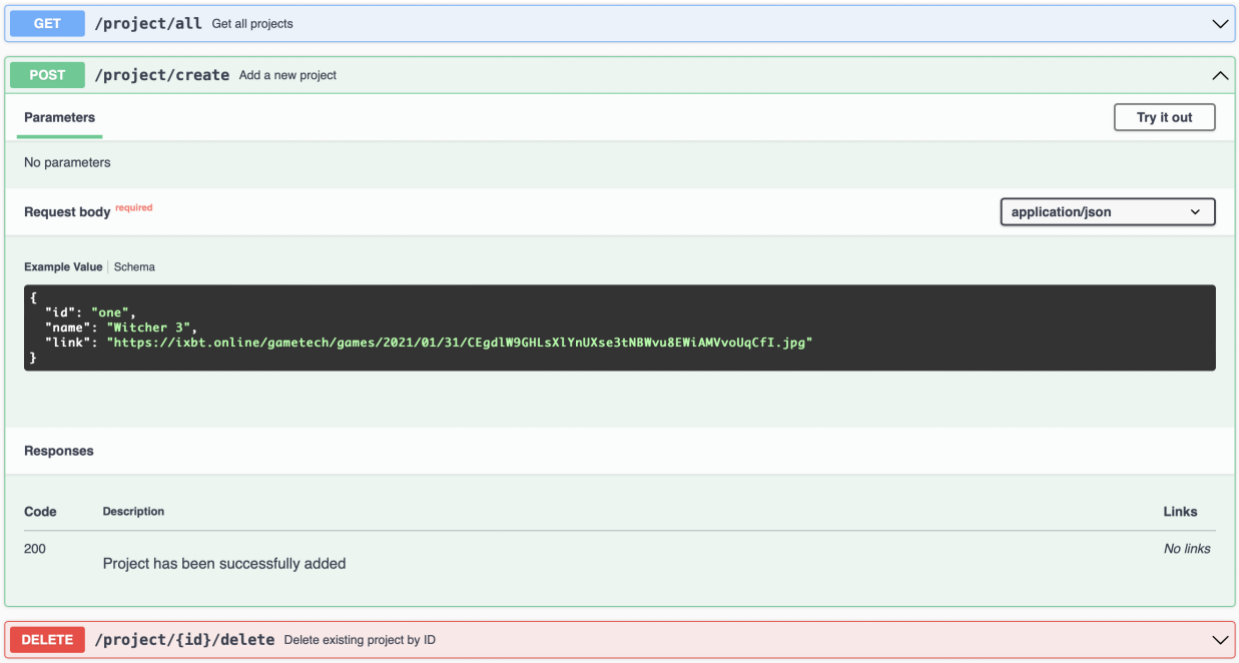
6. Создание фильтра исключений HTTP:

```
import {
  Catch,
  HttpException,
  ExceptionFilter,
  ArgumentsHost,
} from '@nestjs/common';
import { Request, Response } from 'express';

@Catch(HttpException)
export class HttpExceptionFilter implements ExceptionFilter {
  catch(exception: HttpException, host: ArgumentsHost) {
    const ctx = host.switchToHttp();
    const response = ctx.getResponse<Response>();
    const request = ctx.getRequest<Request>();
    const status = exception.getStatus();

    response.status(status).json({
      statusCode: status,
      message: exception.message,
      timestamp: new Date().toISOString(),
      path: request.url,
    });
  }
}
```

Инициализируем его в модуле приложения:

```
providers: [
  AppService,
  {
    provide: APP_FILTER,
    useClass: HttpExceptionFilter,
  },
],
```

7. Проверка результата:



Вывод:

Я достигнул работоспособности всех кнопок и заменил зашитые в код данные на отображение реальных данных из базы данных, полученных путём отправки сетевых запросов на сервер.