

Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

по лабораторной работе №4
по дисциплине «Web-программирование»

Автор: Шарапков Егор Викторович М33081

Преподаватель: Приискалов Роман Андреевич



УНИВЕРСИТЕТ ИТМО

Санкт-Петербург 2022

Генерируем модули с помощью nest cli.

```
PS C:\Users\Egor\Desktop\web 6 sem\applemarket> nest g module user
CREATE src/user/user.module.ts (81 bytes)
UPDATE src/app.module.ts (254 bytes)
PS C:\Users\Egor\Desktop\web 6 sem\applemarket> nest g module device
CREATE src/device/device.module.ts (83 bytes)
UPDATE src/app.module.ts (323 bytes)
PS C:\Users\Egor\Desktop\web 6 sem\applemarket> nest g module basket
```

Будут созданы соответствующие директории и файл module.

Под модулем, согласно документации NestJS, понимается набор классов (контроллеров, сервисов, моделей и т. п.) решающих одну конкретную задачу/вариант использования.

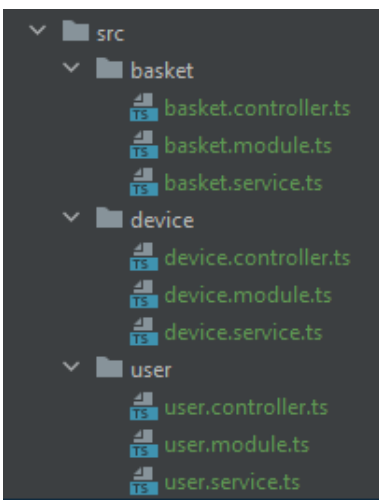
Далее, для работы нашего API нам нужно создать контроллеры и сервисы, которые будут импортированы в соответствующие модули.

Генерируем контроллеры и сервисы.

```
$nest g controller user
```

```
$nest g service user
```

В результате должна получиться данная иерархия.



Далее нам нужно создать конечные точки для каждой из модели. Пример реализации контроллера:

```
import {Controller, Delete, Get, Param, Post} from '@nestjs/common';
import {UserService} from './user.service';
import {User} from '@prisma/client';
import {ApiOperation, ApiResponse, ApiTags} from '@nestjs/swagger';

@ApiTags('user')
@Controller({prefix: 'user'})
export class UserController {
  constructor(private readonly userService: UserService) {}

  @ApiOperation({options: {
    summary: 'Get a user by id and name'
  }})
  @ApiResponse({options: {
    status: 501,
    description: 'The method is not implemented yet'
  }})
  @Get({path: ':user'})
  async getUser(@Param('user') id: number, name: string): Promise<User> {
    return await this.userService.findUser(id, name);
  }
}
```

```

    @ApiOperation({ options: {
      summary: 'Create a user by email and name'
    }})
    @ApiResponse({ options: {
      status: 501,
      description: "The method is not implemented yet"
    }})
    @Post({ path: 'create' })
    async createUser(email: string, name: string): Promise<User> {
      return await this.userService.createUser(email, name);
    }

    @ApiOperation({ options: {
      summary: 'Delete a user by id and name'
    }})
    @ApiResponse({ options: {
      status: 501,
      description: "The method is not implemented yet"
    }})
    @Delete({ path: ':user/delete' })
    async deleteUser(@Param('user') id: number, name: string): Promise<User> {
      return await this.userService.deleteUser(id, name);
    }
  }
}

```

Также нужно создать сервис для этой модели, в котором бизнес-логика будет реализована в будущем.

```

import { Injectable, NotImplementedException } from '@nestjsjs/common';
import { User } from '@prisma/client';

@Injectable()
export class UserService {
  async findUser(id: number, name: string): Promise<User> {
    throw new NotImplementedException();
  }

  async createUser(email: string, name: string): Promise<User> {
    throw new NotImplementedException();
  }

  async deleteUser(id: number, name: string): Promise<User> {
    throw new NotImplementedException();
  }
}

```

После описания всех модулей необходимо настроить автоматическую генерацию OpenAPI спецификации. Для этого воспользуемся swagger.

Устанавливаем его.

```

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

```

Для корректной работы инициализируем swagger в main.ts

```

const config = new DocumentBuilder()
  .setTitle('Apple Market API')
  .setDescription('Early version of Api')
  .setVersion('0.1')
  .addTag({ name: 'user' })
  .addTag({ name: 'device' })
  .addTag({ name: 'basket' })
  .build();

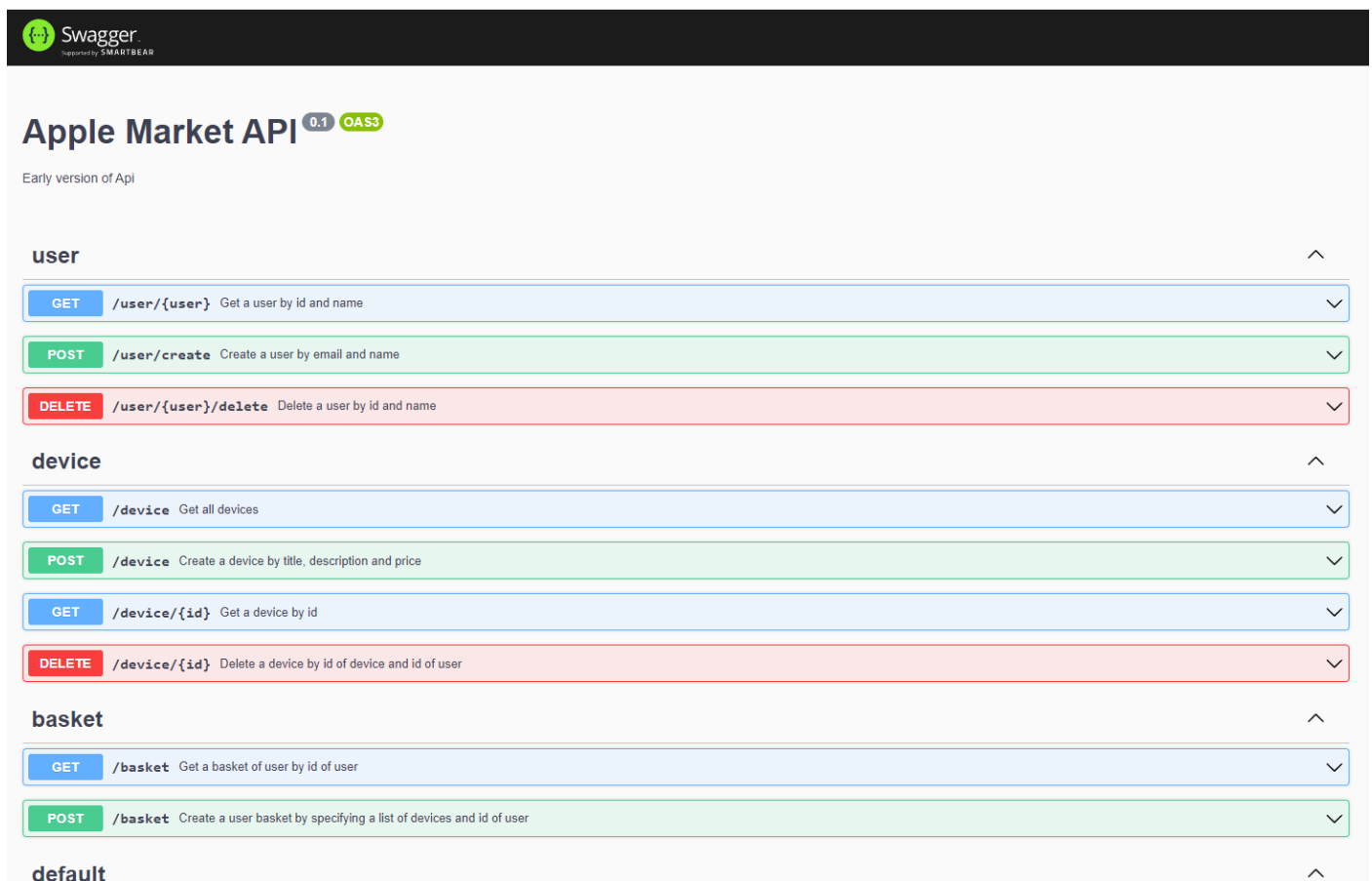
const document = SwaggerModule.createDocument(app, config);
SwaggerModule.setup('api', app, document);

```

Добавляем декораторы. Тэг для отображения внутри соответствующего тэга, описание метода и ответ.

```
7  @ApiTags( tags: 'user')
8  @Controller( prefix: 'user')
9  export class UserController {
10     constructor(private readonly userService: UserService) {}
11
12     @ApiOperation( options: {
13         summary: 'Get a user by id and name'
14     })
15     @ApiResponse( options: {
16         status: 501,
17         description: "The method is not implemented yet"
18     })
19     @Get( path: ':user')
20     async getUser(@Param( property: 'user') id: number, name: string): Promise<User> {
21         return await this.userService.findUser(id, name);
22     }
23 }
```

Теперь мы можем запустить наше приложение, и пройдя к конечной точке /api, увидим следующее



The image shows the Swagger UI for the Apple Market API. The header includes the Swagger logo and the text "Supported by SMARTBEAR". The main title is "Apple Market API" with a version "0.1" and a tag "OAS3". Below the title, it says "Early version of Api". The API is organized into sections: "user", "device", "basket", and "default". Each section contains a list of endpoints with their HTTP methods, paths, and descriptions. The "user" section has three endpoints: GET /user/{user} (Get a user by id and name), POST /user/create (Create a user by email and name), and DELETE /user/{user}/delete (Delete a user by id and name). The "device" section has four endpoints: GET /device (Get all devices), POST /device (Create a device by title, description and price), GET /device/{id} (Get a device by id), and DELETE /device/{id} (Delete a device by id of device and id of user). The "basket" section has two endpoints: GET /basket (Get a basket of user by id of user) and POST /basket (Create a user basket by specifying a list of devices and id of user). The "default" section is currently empty.

Swagger
Supported by SMARTBEAR

Apple Market API 0.1 OAS3

Early version of Api

user

- GET** /user/{user} Get a user by id and name
- POST** /user/create Create a user by email and name
- DELETE** /user/{user}/delete Delete a user by id and name

device

- GET** /device Get all devices
- POST** /device Create a device by title, description and price
- GET** /device/{id} Get a device by id
- DELETE** /device/{id} Delete a device by id of device and id of user

basket

- GET** /basket Get a basket of user by id of user
- POST** /basket Create a user basket by specifying a list of devices and id of user

default

Можно протестировать эти конечные точки.

GET

/user/{user}

Get a user by id and name

^

Parameters

Cancel

Name	Description
user * required number (path)	<div>5</div>

Execute

Clear

Responses

Curl

curl -X 'GET' \ 'http://localhost:3000/user/5' \ -H 'accept: */*'

Request URL

http://localhost:3000/user/5

Server response

Code	Details
501	<div>Error: Not Implemented</div> <div>Response body</div> <div><div>{ "statusCode": 501, "message": "Not Implemented" }</div><div> Download</div></div> <div>Response headers</div> <div><div>connection: keep-alive content-length: 46 content-type: application/json; charset=utf-8 date: Mon, 11 Apr 2022 13:13:53 GMT etag: W/"2e-jABShTiHhXo6mSscMgC7jOH/DmY" keep-alive: timeout=5 x-powered-by: Express</div></div>

Responses

Code	Description	Links
501	The method is not implemented yet	No links