МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

Выполнили студент группы M33081
Аль Даббагх Харит Хуссейн

Проверил
Приискалов Роман Андреевич

САНКТ-ПЕТЕРБУРГ

2022

СОДЕРЖАНИЕ

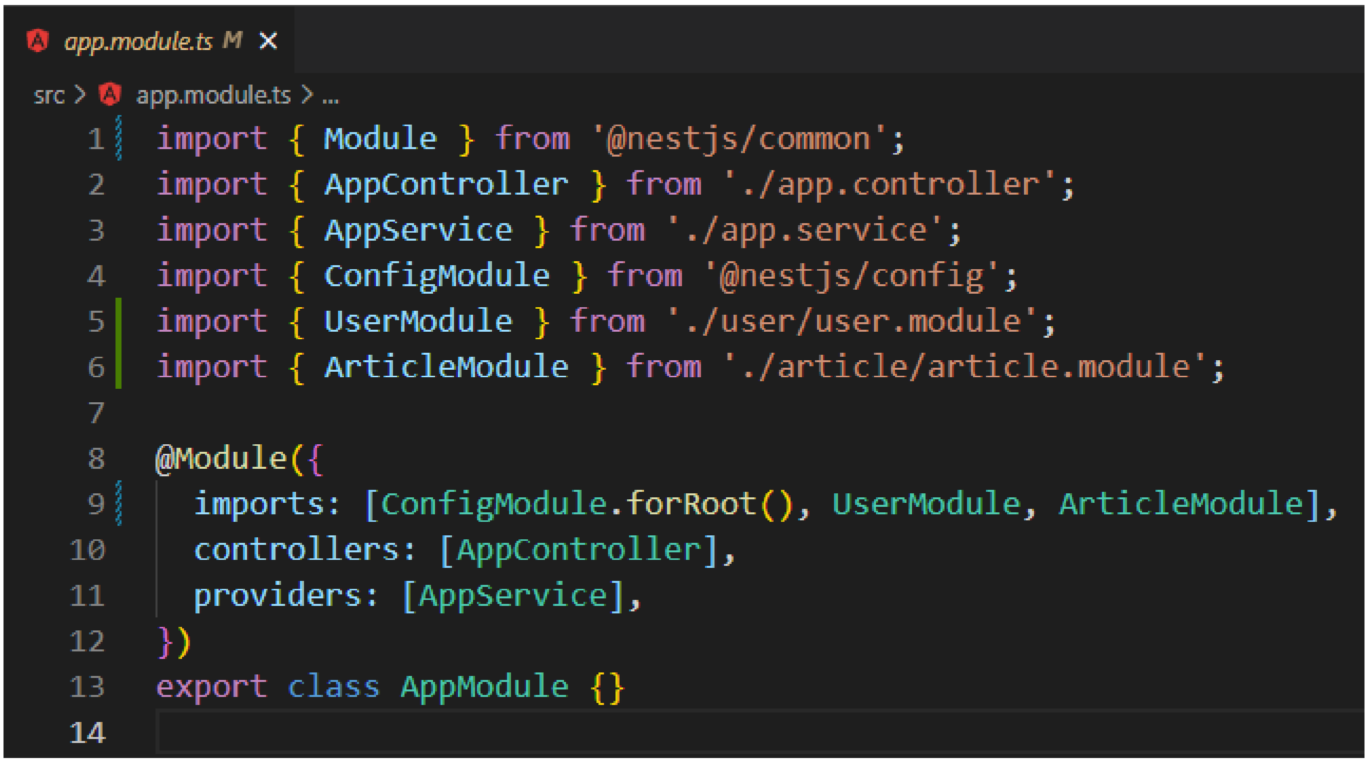
MODULE CREATION

Generating modules

In order to generate a module for our models, we can use the following commands:

```
nest g module user
nest g module article
```

This will create a module for our user model and another for the article model. Then it will automatically import these modules in our base app module.



app.module.ts M ×

src > app.module.ts > ...

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ConfigModule } from '@nestjs/config';
import { UserModule } from './user/user.module';
import { ArticleModule } from './article/article.module';

@Module({
  imports: [ConfigModule.forRoot(), UserModule, ArticleModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}

```

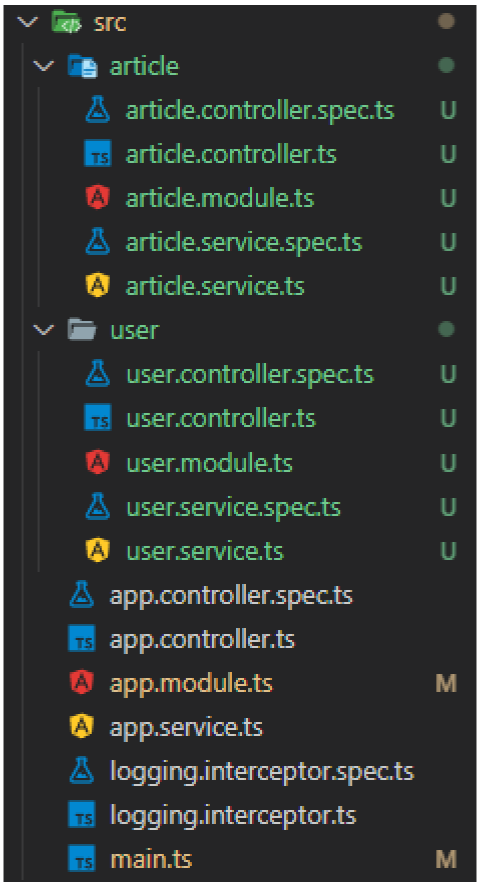
Modules serve as a way to organize our code especially when we'll have a lot of entities and team members working on the same project.

For our API to work we also need to create controllers and services, which will then be imported in their corresponding modules.

To do that from the CLI:

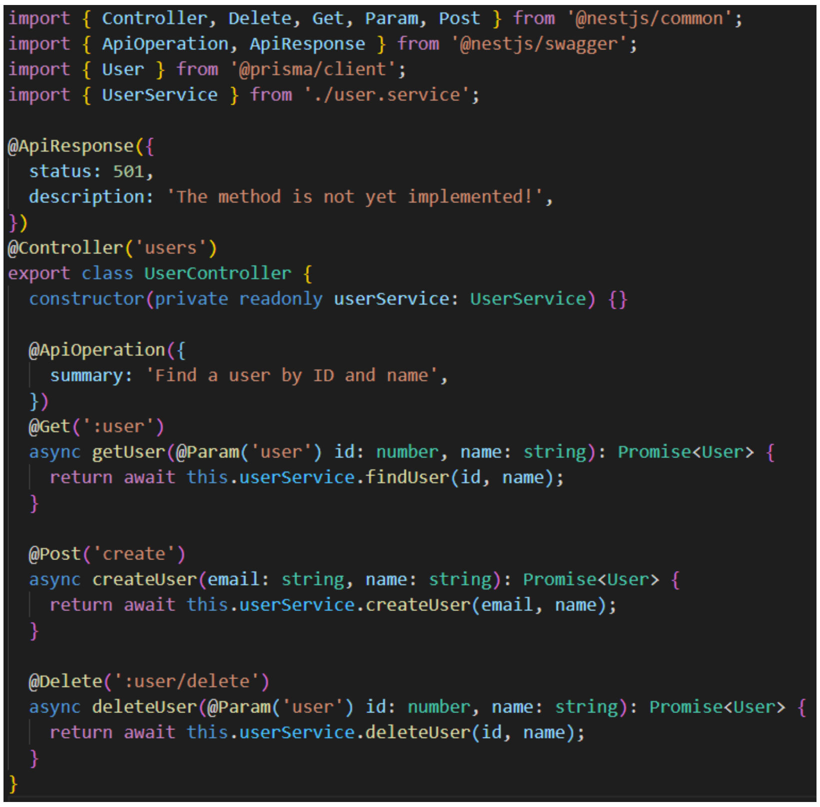
```
nest g controller user
nest g controller article

nest g service user
nest g service article
```

The result file heirarchy should look something like in the right screenshot:



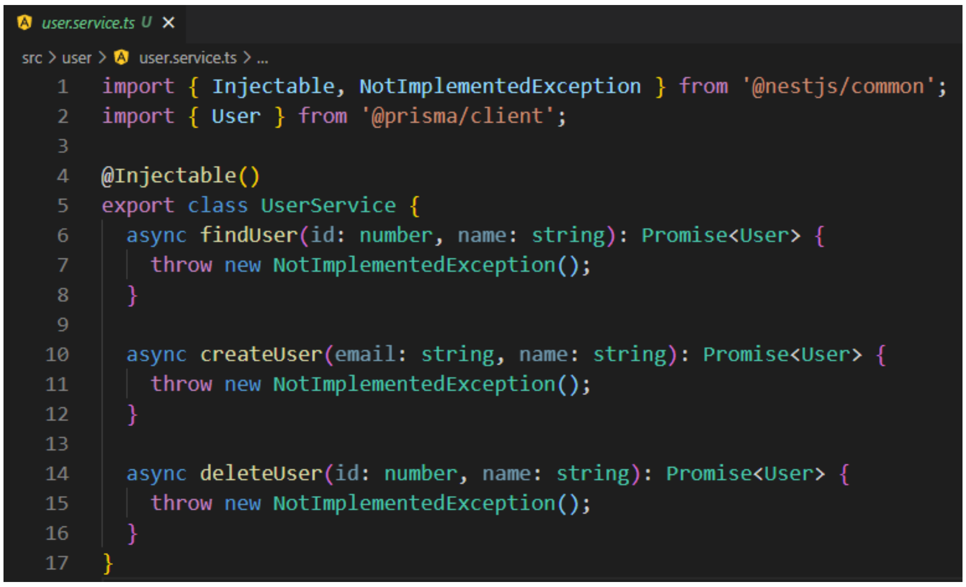
```
src
  article
    article.controller.spec.ts  U
    article.controller.ts       U
    article.module.ts           U
    article.service.spec.ts     U
    article.service.ts          U
  user
    user.controller.spec.ts     U
    user.controller.ts          U
    user.module.ts              U
    user.service.spec.ts        U
    user.service.ts             U
  app.controller.spec.ts
  app.controller.ts
  app.module.ts                 M
  app.service.ts
  logging.interceptor.spec.ts
  logging.interceptor.ts
  main.ts                       M
```

Next we need to create our endpoints (controller) for each of these models. The implementation for the user model should look something like this:



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

@ApiResponse({
  status: 501,
  description: 'The method is not yet implemented!',
})
@Controller('users')
export class UserController {
  constructor(private readonly userService: UserService) {}

  @ApiOperation({
    summary: 'Find a user by ID and name',
  })
  @Get(':user')
  async getUser(@Param('user') id: number, name: string): Promise<User> {
    return await this.userService.findUser(id, name);
  }

  @Post('create')
  async createUser(email: string, name: string): Promise<User> {
    return await this.userService.createUser(email, name);
  }

  @Delete(':user/delete')
  async deleteUser(@Param('user') id: number, name: string): Promise<User> {
    return await this.userService.deleteUser(id, name);
  }
}
```

The same way we need to create the service for that model (In which the business logic will be implemented in the future). An example for the user service should look like this:



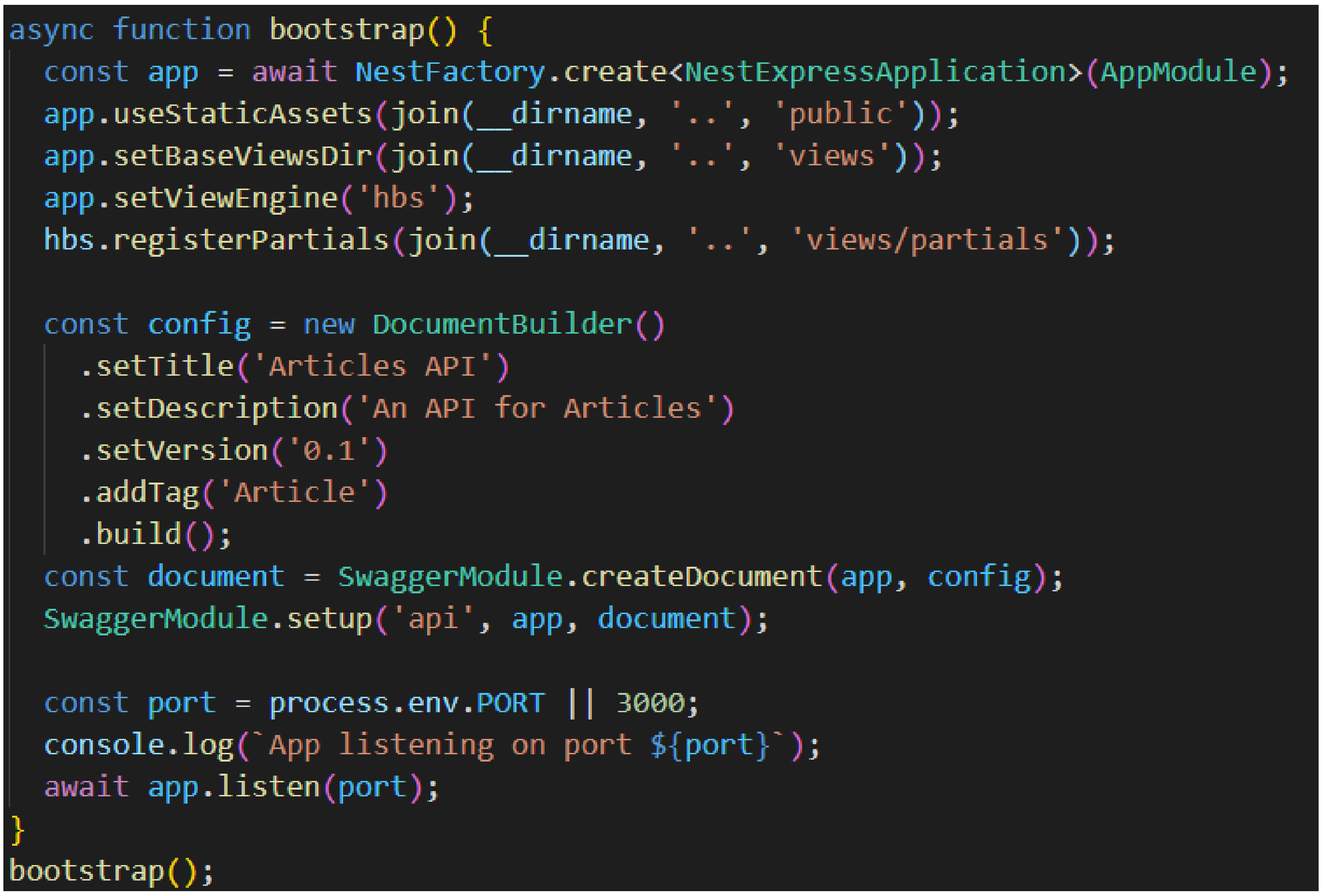
```
import { Injectable, NotImplementedException } from '@nestjs/common';
import { User } from '@prisma/client';

@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();
  }
}
```

Swagger OpenAPI

Swagger is a way for us to document and easily test our API. It needs to be installed using the command:

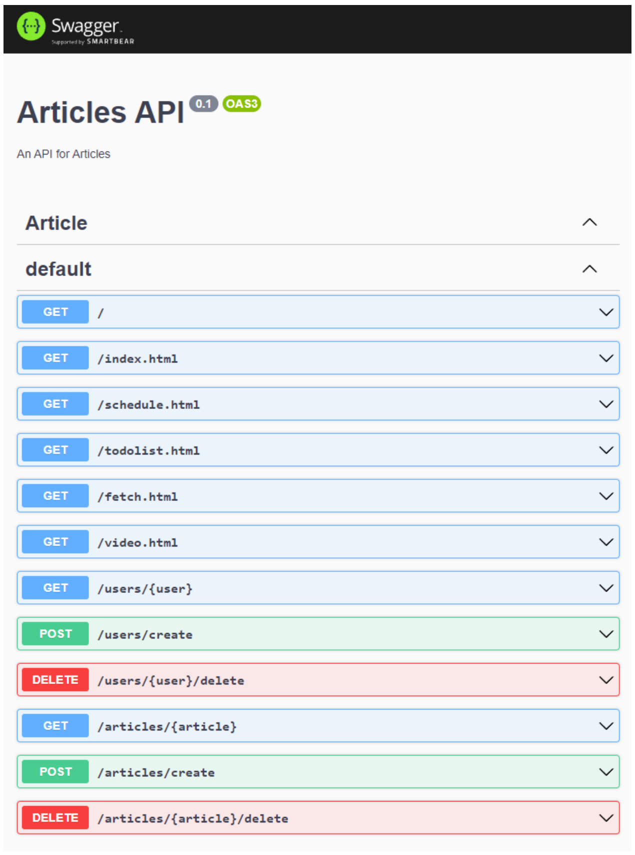
```
npm install --save @nestjs/swagger swagger-ui-express
```

For swagger to work we also need to initialize it in main.ts:



```
async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule);
  app.useStaticAssets(join(__dirname, '..', 'public'));
  app.setBaseViewsDir(join(__dirname, '..', 'views'));
  app.setViewEngine('hbs');
  hbs.registerPartials(join(__dirname, '..', 'views/partials'));

  const config = new DocumentBuilder()
    .setTitle('Articles API')
    .setDescription('An API for Articles')
    .setVersion('0.1')
    .addTag('Article')
    .build();
  const document = SwaggerModule.createDocument(app, config);
  SwaggerModule.setup('api', app, document);

  const port = process.env.PORT || 3000;
  console.log(`App listening on port ${port}`);
  await app.listen(port);
}
bootstrap();
```

If we try to run our project now and go to the endpoint /api, we can see the following:



Swagger

Articles API 0.1 OAS3

An API for Articles

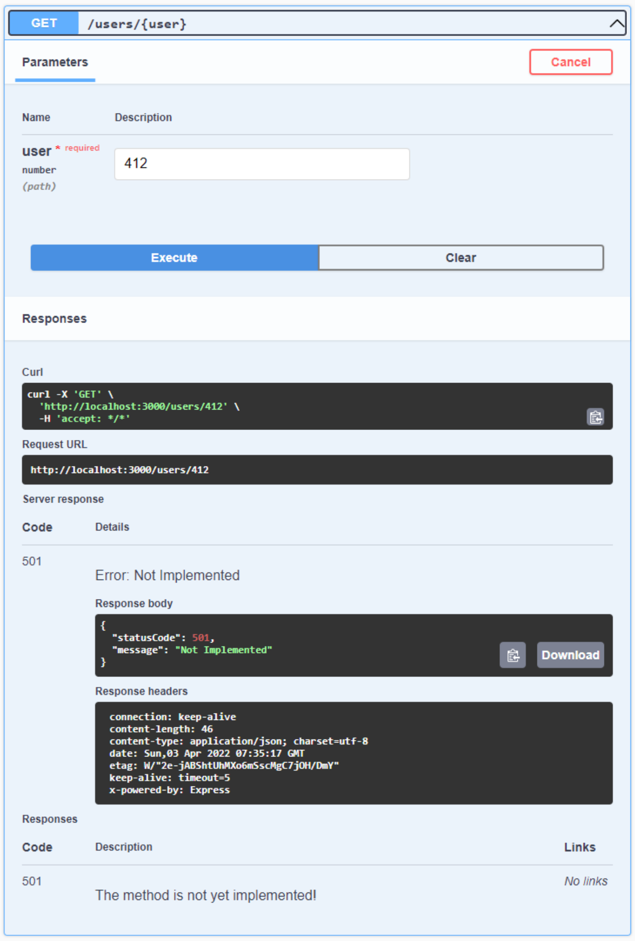
Article

default

```
GET     /
GET     /index.html
GET     /schedule.html
GET     /todolist.html
GET     /fetch.html
GET     /video.html
GET     /users/{user}
POST    /users/create
DELETE  /users/{user}/delete
GET     /articles/{article}
POST    /articles/create
DELETE  /articles/{article}/delete
```

Each of these points can be documented using decorators.

We can also test these endpoints:



GET /users/{user}

Parameters

Name	Description
user * required number *(path)*	412

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:3000/users/412' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:3000/users/412
```

Server response

Code	Details
501	Error: Not Implemented

Response body

```
{
  "statusCode": 501,
  "message": "Not Implemented"
}
```

Response headers

```
connection: keep-alive
content-length: 46
content-type: application/json; charset=utf-8
date: Sun,03 Apr 2022 07:35:17 GMT
etag: W/"2e-jABShtUhMXo6mSscMgC7jOH/DmY"
keep-alive: timeout=5
x-powered-by: Express
```

Responses

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