

4. Создание контроллеров страниц и спецификации

В результате выполнения данной лабораторной работы должна быть получена спецификация, содержащая все необходимые методы для работы с ранее созданной моделью.

Контрольный срок сдачи работы: 10-ая неделя обучения.

Для выполнения данной лабораторной работы вам необходимо выделить в ранее созданном приложении, контроллеры, которые бы работали с доменной моделью, спроектированной в рамках предыдущей работы.

Прежде чем приступить к описанию контроллеров, предлагается первым делом спроектировать верхнеуровневое взаимодействие модулей. Под модулем, согласно документации NestJS (<https://docs.nestjs.com/modules>), понимается набор классов (контроллеров, сервисов, моделей и т.п.) решающих одну конкретную задачу/конкретный вариант использования.

Например, если вы разрабатываете систему блогов с личный кабинетом пользователя, то можно разбить всё на три модуля: Модуль для представления профилей пользователей, модуль для работы с самими пользователями и модуль для работы/отображения статей (постов блога).

Рекомендуется выделить для каждого модуля отдельную директорию (см. теоретический материал по DDD) внутри которой будут в дальнейшем описаны все необходимые классы для работы. Пример файловой структуры такого проекта можно посмотреть [здесь](#).

В рамках данной лабораторной работы не требуется имплементация бизнес логики, необходимо лишь выстроить каркас (описать контроллеры, зарегистрировать сервисы внутри модуля и связать модули для корректной работы DI механизма).

Требуется создать контроллеры, которые бы получали как зависимость необходимые сервисы. Внутри контроллера описать все варианты использования в REST-овой нотации в виде отдельных Action'ов зарегистрированных на обработку по конкретному Endpoint'у.

Пример полностью описанного контроллера можно посмотреть [по ссылке](#).

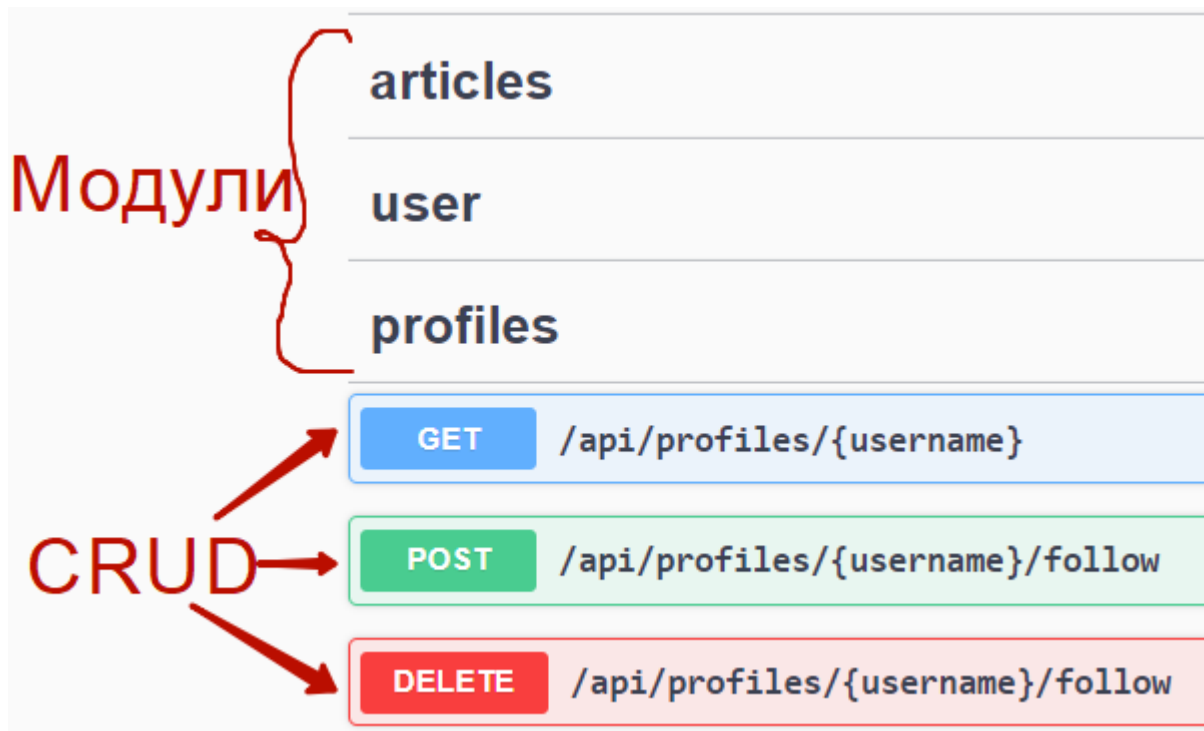
Внутри самих сервисов, в рамках данной лабораторной работы, разрешается не реализовывать бизнес логику самого приложения на данном этапе.

```
async findProfile(id: number, username: string):  
Promise<ProfileRO> {  
    throw new NotImplementedException();  
}
```

Исключения типа `NotImplementedException` будут обработаны Nest'ов автоматический и данный метод отработает согласно RFC и вернёт ответ HTTP 501 Not Implemented
(<https://docs.nestjs.com/exception-filters#built-in-http-exceptions>)

После описания всех модулей необходимо настроить автоматическую генерацию OpenAPI спецификации. Для этого рекомендуется воспользоваться модулем `swagger` (<https://docs.nestjs.com/openapi/introduction>) и материалами лекций по BFF.

В конечном счёте ожидается следующая структура внутри документации:



(ссылки на примеры удачной документации и нейминга методов смотрите в конце презентации по видам API и их документации)

Пример метода получения профиля пользователя:

GET

/api/profiles/{username}

Parameters

Cancel

Name	Description
username * required string (path)	<input type="text" value="admin"/>

Execute

Responses

Code	Description	Links
200		No links

Результат работы метода:

Server response

Code	Details
501 <i>Undocumented</i>	<div>Error: Not Implemented</div> <div>Response body</div> <pre>{ "statusCode": 501, "message": "Not Implemented"}</pre> <div>Response headers</div> <pre>access-control-allow-origin: * connection: keep-alive content-length: 46 content-type: application/json; charset=utf-8 date: Mon, 21 Mar 2022 18:43:44 GMT etag: W/"2e-jABShtUhMXo6mSscMgC7j0H/DmY" keep-alive: timeout=5 x-powered-by: Express</pre>

Для более подробного описания возможных вариантов ответа сервера и добавления текстового описания предназначения методов рекомендуется воспользоваться вспомогательными декораторами поддерживаемых модулем @nest/swagger (<https://docs.nestjs.com/openapi/decorators>)

Пример с подробным описанием выглядит следующим образом:

```
@ApiOperation({
  summary: 'Create comment'
})
@ApiParam({ name: 'slug', type: 'string' })
@ApiResponse({
  status: 201,
  description: 'The comment has been successfully
created.'
})
@ApiResponse({
  status: 403,
  description: 'Forbidden.'
})
@Post('/:slug/comments')
async createComment(
  @Param('slug') slug,
  @Body('comment') commentData: CreateCommentDto
) {
  return await this.articleService.addComment(slug,
commentData);
}
```

POST

/api/articles/{slug}/comments

Create comment

Parameters

Try it out

Name	Description
slug * required	
string	slug
(path)	

Responses

Code	Description	Links
201	The comment has been successfully created.	No links
403	Forbidden.	No links

Обращаю ваше внимание, что документация должна быть читабельной, т.е. не должно появляться вопросов “А что это операция делает?” или “За что отвечает данный параметр?”. Чем более подробно будут описаны модели и методы, тем легче конечным потребителям вашего API будет в нём разобраться.

В случаях, если ни автор доменный эксперт (т.е. вы), ни проверяющий не могут разобраться как должен работать API, то это может служить причиной для выставления неполного балла как результат плохого проектирования и/или составления документации.