

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №3

Выполнил:

Корчагин Вадим

Группа
К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Тема

Платформа для фитнес-тренировок и здоровья.

Задача

- реализовать автодокументирование средствами swagger;
- реализовать документацию API средствами Postman.

Технологии:

- Swagger (routing-controllers-openapi, swagger-ui-express)
- Postman (импорт через openapi2postmanv2)
- curl, jq — автоматизация через Makefile
- Makefile — сборка документации, выгрузка и синхронизация с Postman API

Ход работы

Swagger

В проекте используется генерация спецификации OpenAPI из метаданных routing-controllers и class-validator. Это обеспечивается следующим:

Модули:

- getMetadataArgsStorage — сбор метаданных контроллеров
- validationMetadatasToSchemas — генерация схем DTO
- routingControllersToSpec — генерация спецификации OpenAPI 3.0
- swagger-ui-express — отображение UI

1. Декораторы в контроллерах влияют на описание маршрутов

- @Get("/") — метод HTTP и путь
- @OpenAPI(...) — описание для Swagger (summary, description, теги, параметры)
- @ResponseSchema(...) — тип возвращаемых данных (генерирует 200 OK и структуру JSON)

```
@JsonController("/blog-posts")
export class BlogPostController extends BaseController<BlogPost> {
  private readonly blogPostService: BlogPostService;

  constructor() {
    super(new BlogPostService());
    this.blogPostService = this.service as BlogPostService;
  }

  @Get("/")
  @OpenAPI({ summary: "Get all blog posts" })
  @ResponseSchema(BlogPostResponseDto, { isArray: true })
  async getAll() {
    return this.blogPostService.findAllWithRelations();
  }
}
```

2. DTO и class-validator влияют на входные схемы

Swagger получает метаинформацию и создаёт JSON Schema: типы, обязательность, описание.

- `@IsEmail`, `@IsString` → влияют на Swagger как требования к полям
- `@Type(() => String)` → гарантирует правильный тип в спецификации

```
export class BlogPostResponseDto {
  @IsInt()
  | @Type(() => Number)
  id: number;

  @IsInt()
  | @Type(() => Number)
  author_id: number;

  @IsString()
  | @Type(() => String)
  title: string;

  @IsString()
  | @Type(() => String)
  content: string;

  @IsDate()
  | @Type(() => Date)
  created_at: Date;

  @IsDate()
  | @Type(() => Date)
  updated_at: Date;
}
```

3. `validationMetadatasToSchemas` генерирует схемы из DTO

Эта часть создаёт раздел `components.schemas` в `swagger.json` — там описаны все DTO.

```
export function useSwagger(app: Application, options?: { controllers: any[] }) {
  try {
    const schemas = validationMetadatasToSchemas({
      classTransformerMetadataStorage: defaultMetadataStorage,
      refPointerPrefix: "#/components/schemas/",
    });
  }
}
```

4. `routingControllersToSpec` → собирает всё в спецификацию

Это главный шаг — он создаёт весь swagger.json.

- controllers — список всех контроллеров
- components.schemas — DTO и валидации
- securitySchemes — схемы авторизации (Bearer)

```
export function useSwagger(app: Application, options?: { controllers: any[] }) {  
  const spec = routingControllersToSpec(  
    storage,  
    {  
      controllers: options?.controllers || [],  
      defaultErrorHandler: false,  
    },  
    {  
      components: {  
        schemas,  
        securitySchemes: {  
          bearerAuth: {  
            type: "http",  
            scheme: "bearer",  
            bearerFormat: "JWT",  
          },  
        },  
      },  
      security: [{ bearerAuth: [] }],  
      info: {  
        title: "Fitness API",  
        version: "1.0.0",  
        description: "API documentation for the Fitness application",  
      },  
      servers: [  
        {  
          url: "http://localhost:3000",  
          description: "Local server",  
        },  
      ],  
    },  
  );  
}
```

5. swagger-ui-express отображает интерфейс

Swagger доступен по адресу /docs

Можно визуально тестировать каждый маршрут

Можно вставить JWT-токен в UI и использовать защищённые @Authorized() методы

```
app.get("/swagger.json", (_req, res) => {
  res.json(spec);
});
app.use("/docs", swaggerUi.serve, swaggerUi.setup(spec));
} catch (error) {
  console.error("Error setting up Swagger:", error);
}
```

6. В Swagger также настраивается авторизация

В Swagger UI появляется кнопка Authorize

Токен передаётся автоматически в заголовке Authorization: Bearer ...

```
securitySchemes: {
  bearerAuth: {
    type: "http",
    scheme: "bearer",
    bearerFormat: "JWT",
  },
},
security: [{ bearerAuth: [] }],
```

7. Использование Swagger

```
useSwagger(app, {
  controllers: [
    BlogCommentController,
    AuthController,
    BlogPostController,
    UserController,
    OrderController,
    PaymentController,
    RoleController,
    WorkoutController,
    TrainingPlanController,
    UserTrainingPlanController,
    UserProgressController,
    TrainingPlanWorkoutController,
  ],
});
```

Результаты:

Fitness API

1.0.0

OAS 3.0

API documentation for the Fitness application

Servers

http://localhost:3000 - Local server

Authorize

Blog Comment

GET

/blog-comments/

Get all blog comments

POST

/blog-comments/

Create a new blog comment

GET

/blog-comments/{id}

Get blog comment by ID

PATCH

/blog-comments/{id}

Update a blog comment

DELETE

/blog-comments/{id}

Delete a blog comment

Auth

POST

/auth/login

User login

Blog Post

POST

/blog-comments/

Create a new blog comment

Parameters

Try it out

No parameters

Request body

application/json

CreateBlogCommentDto

Example Value

Schema

```
{
  "post_id": 0,
  "user_id": 0,
  "comment_text": "string"
}
```

Responses

Code	Description	Links
200	<div><div>Media type</div><div>application/json</div><div>Controls Accept header.</div><div>Example Value</div><div>Schema</div></div> <div><pre>{ "id": 0, "post_id": 0, "user_id": 0, "comment_text": "string", "created_at": "string" }</pre></div>	No links

Schemas



```
CreateBlogCommentDto {  
  post_id* > [...]  
  user_id* > [...]  
  comment_text* > [...]  
}
```

UpdateBlogCommentDto >

BlogCommentResponseDto >

LoginRequestDto >

LoginResponseDto >

CreateBlogPostDto >

Postman

Postman используется как инструмент для:

- Тестирования REST API вручную
- Подготовки и демонстрации коллекций запросов
- Документирования API для внешних команд (frontend, QA, заказчики)
- Интеграции с CI/CD и автотестами

Вместо ручного создания запросов в Postman, используется генерация коллекции из OpenAPI-спецификации (swagger.json).

Инструмент - openapi2postmanv2 — утилита для конвертации спецификации OpenAPI → Postman Collection.

Makefile: Автоматизация процесса

Процесс автоматизирован с помощью Makefile-команд. Это исключает ручные ошибки и упрощает поддержку.

1. Генерация swagger.json

Получает актуальную спецификацию API с сервера.

```
swagger.json:
    curl http://localhost:3000/swagger.json -o
swagger.json
```

2. Конвертация в Postman Collection

```
postman.json: swagger.json
    openapi2postmanv2 -s swagger.json -o postman.json -p
```

Генерирует postman.json — коллекцию, которую можно открыть в Postman.

Флаг -p означает: "добавлять примеры" на основе схем из Swagger.

3. Обёртка в формат Postman API

```
wrap:
    jq '{ collection: . }' postman.json > wrapped-
postman.json
```

Оборачивает JSON в объект collection, как того требует Postman API.

4. Синхронизация с Postman по API

Обновляет опубликованную коллекцию на Postman Cloud с помощью API-ключа.

sync:

```
curl -X PUT https://api.getpostman.com/collections/$
(COLLECTION_UID) \
  -H "X-API-Key: $(POSTMAN_API_KEY)" \
  -H "Content-Type: application/json" \
  -d @wrapped-postman.json
```

После выполнения команд:

- Автоматически создаётся актуальная коллекция Postman
- Swagger-спецификация и Postman остаются в синхронизации
- Все маршруты, тела запросов, ответы и авторизация — уже настроены

```
labs > lab1 > makefile
You, 6 days ago | 1 author (You)
1 POSTMAN_API_KEY := $(POSTMAN_API_KEY)
2 COLLECTION_UID := $(COLLECTION_UID)
3
4 swagger.json:
5     curl http://localhost:3000/swagger.json -o swagger.json
6
7 postman.json: swagger.json
8     openapi2postmanv2 -s swagger.json -o postman.json -p
9
10 wrap:
11     jq '{ collection: . }' postman.json > wrapped-postman.json
12
13 sync: swagger.json
14     openapi2postmanv2 -s swagger.json -o postman.json -p
15     jq '{ collection: . }' postman.json > wrapped-postman.json
16     curl -X PUT https://api.getpostman.com/collections/$(COLLECTION_UID) \
17         -H "X-API-Key: $(POSTMAN_API_KEY)" \
18         -H "Content-Type: application/json" \
19         -d @wrapped-postman.json
20
21 clean:
22     rm -f swagger.json postman.json wrapped-postman.json
```

Результаты:

gerenny's Workspace

NewImport

Collections

Environments

Flows

History

+

▼ Fitness API

> blog-comments

> auth

> blog-posts

> {id}

> GET Get blog post by ID

> PATCH Update a blog post

> DEL Delete a blog post

> GET Get all blog posts

> POST Create a new blog post

> users

> orders

> payments

> roles

> workouts

> training-plans

> user-training-plans

> user-progress

> training-plan-workouts

Overview

POST Create a new blog pos

+

No environment

▼

HTTP Fitness API / blog-posts / Create a new blog post

Save▼Share

POST▼{{baseUrl}}/blog-posts/

Send▼

Params

Authorization

Headers (12)

Body●

Scripts

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON▼

Cookies

Beautify

1{

2"author_id": "<integer>",

3"title": "<string>",

4"content": "<string>"

5}

Response

History▼

Выводы