

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №3

Выполнил:

Шалунов Андрей

Группа К3340

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Реализовать автодокументирование средствами swagger;

Реализовать документацию API средствами Postman.

Ход работы

1. Конфигурация

Создал файл *config/swagger.ts*, где:

- собираются метаданные контроллеров (*getMetadataArgsStorage()*),
- генерируются схемы валидации (*validationMetadatasToSchemas()*),
- формируется спецификация OpenAPI через *routingControllersToSpec()*.

Добавил два маршрута в Express:

- */docs* — веб-интерфейс Swagger UI;
- */docs.json* — «сырой» JSON-файл спецификации.

```
import { Application } from 'express';

import {
    getMetadataArgsStorage,
    RoutingControllersOptions,
} from 'routing-controllers';

import { routingControllersToSpec } from 'routing-controllers-openapi';

import * as swaggerUi from 'swagger-ui-express';

import { validationMetadatasToSchemas } from 'class-validator-jsonschema';

import SETTINGS from './settings';

export function useSwagger(
    app: Application,
    options: RoutingControllersOptions,
```

```
) : Application {

  try {

    const schemas = validationMetadatasToSchemas({

      refPointerPrefix: '#/definitions/',

    });

    const storage = getMetadataArgsStorage();

    const spec = routingControllersToSpec(storage, options, {

      components: {

        schemas,

        securitySchemes: {

          bearerAuth: {

            type: 'http',

            scheme: 'bearer',

            bearerFormat: 'JWT',

          },

        },

      },

      info: {

        title: 'RealEstate API',

        description: 'Автодокументация REST API для проекта RealEstate',

        version: '1.0.0',

      },

      servers: [

        {

          url:

`${SETTINGS.APP_PROTOCOL}://${SETTINGS.APP_HOST}:${SETTINGS.APP_PORT}${SETTIN
GS.APP_API_PREFIX}`,
```

```

    },

    ],

  });

  app.use('/docs', swaggerUi.serve, swaggerUi.setup(spec));

  app.get('/docs.json', (_req, res) => {

    res.json(spec);

  });

  return app;

} catch (error) {

  console.error('Ошибка настройки Swagger:', error);

  return app;

}
}

```

2. Аннотации в контроллерах

Каждый метод контроллера отмечен декораторами `@OpenAPI({ summary })` и `@ResponseSchema(...)` для описания целей и ответов.

Пример контроллера *auth.controller.ts*

```

import { JsonController, Post, Body, HttpStatusCode } from 'routing-controllers';

import { OpenAPI, ResponseSchema } from 'routing-controllers-openapi';

import { RegisterDto, LoginDto } from '../dto/auth.dto';

import { AuthService } from '../services/auth.service';

class AuthResponse {

  user_id!: number;

  email!: string;

  name!: string;

}

```

```
}

class TokenResponse {

    token!: string;

}

class ErrorResponse {

    message!: string;

}

@JsonController('/auth')

export class AuthController {

    @Post('/register')

    @HttpCode(201)

    @OpenAPI({ summary: 'Register new user' })

    @ResponseSchema(AuthResponse, { statusCode: 201 })

    @ResponseSchema(ErrorResponse, { statusCode: 400 })

    async register(@Body() dto: RegisterDto) {

        const user = await AuthService.register(dto.email, dto.password,
dto.name);

        return { user_id: user.user_id, email: user.email, name: user.name };

    }

    @Post('/login')

    @OpenAPI({ summary: 'Login and get JWT token' })

    @ResponseSchema(TokenResponse, { statusCode: 200 })

    @ResponseSchema(ErrorResponse, { statusCode: 401 })

    async login(@Body() dto: LoginDto) {

        const token = await AuthService.login(dto.email, dto.password);

        return { token };

    }

}
```

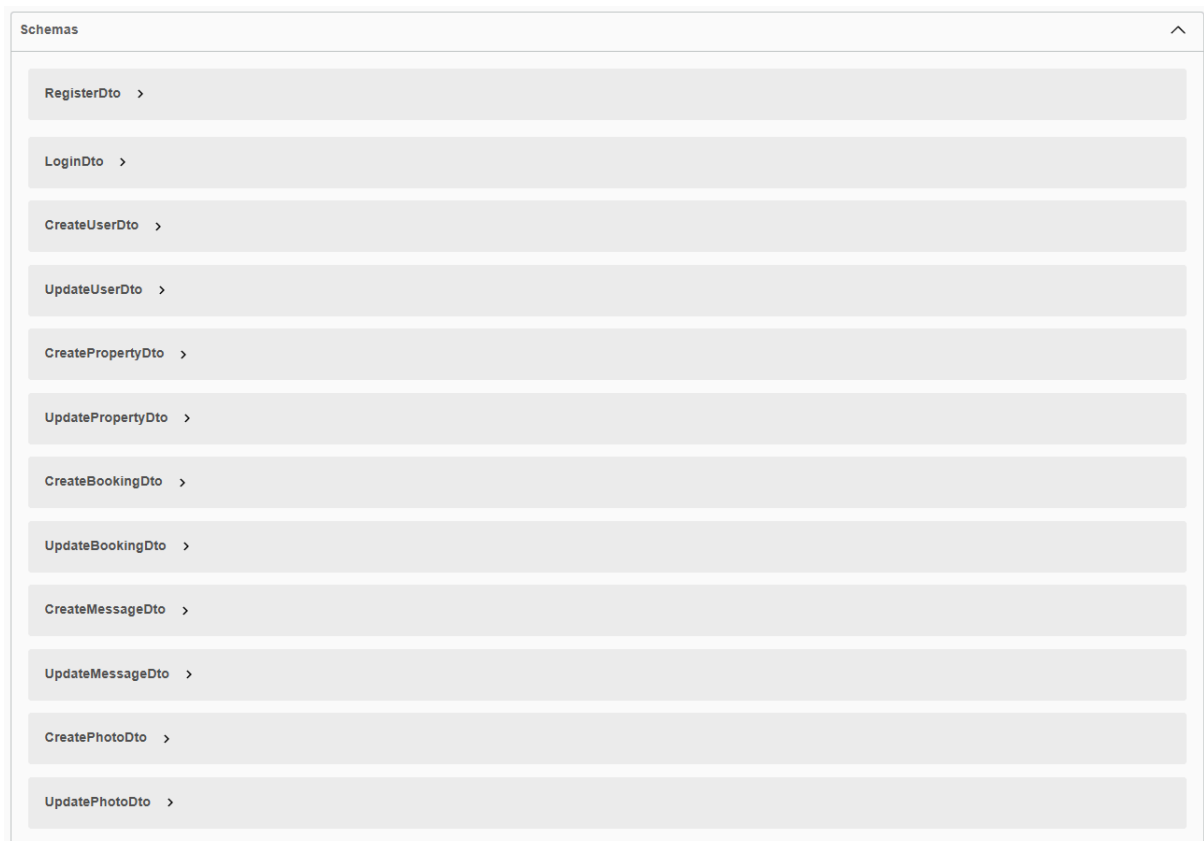
```
}
```

3. Проверка

После запуска сервера документация доступна по адресу

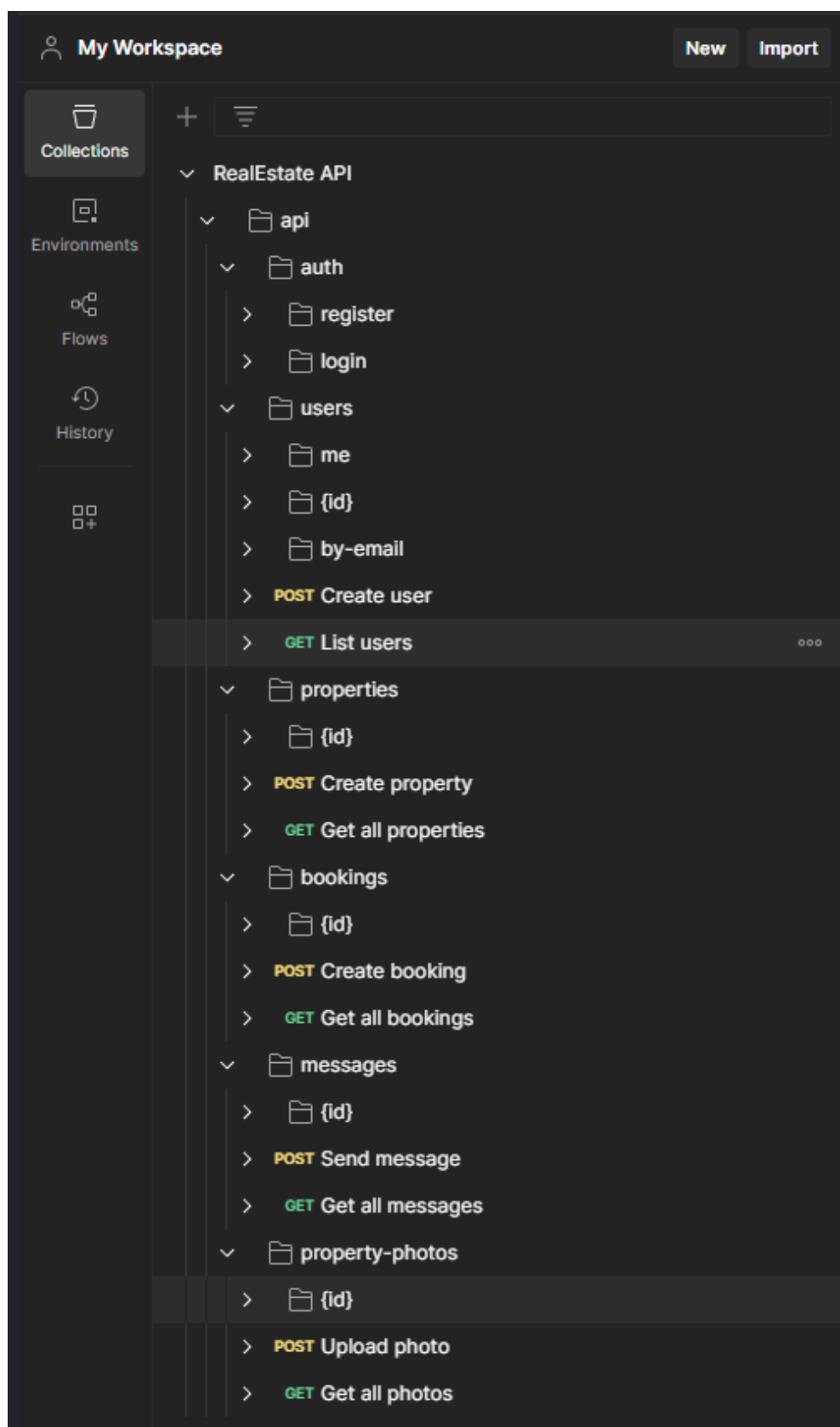
<http://localhost:3000/docs/>.

Auth			^
POST	/api/auth/register	Register new user	v
POST	/api/auth/login	Login and get JWT token	v
User			^
POST	/api/users/	Create user	v
GET	/api/users/	List users	v
GET	/api/users/me	Get current user	v
GET	/api/users/{id}	Get user by id	v
PATCH	/api/users/{id}	Update user	v
DELETE	/api/users/{id}	Delete user	v
GET	/api/users/by-email	Get user by email	v
Property			^
POST	/api/properties/	Create property	v
GET	/api/properties/	Get all properties	v
GET	/api/properties/{id}	Get property by id	v
PUT	/api/properties/{id}	Update property	v
DELETE	/api/properties/{id}	Delete property	v
Booking			^



Документация через Postman

Postman автоматически создал коллекцию RealEstate API с полным перечнем эндпоинтов и моделей:



После экспортировал файл *RealEstate API.postman_collection.json*:


```

1 {
2   "info": {
3     "_postman_id": "ca42fc1-a88b-481b-a10f-4326bb156f4f",
4     "name": "RealEstate API",
5     "description": "Автодокументация REST API для проекта RealEstate",
6     "schema": "https://schema.getpostman.com/json/collection/v2.1.0/collection.json",
7     "_exporter_id": "25956602"
8   },
9   "item": [
10    {
11      "name": "api",
12      "item": [
13        {
14          "name": "auth",
15          "item": [
16            {
17              "name": "register",
18              "item": [
19                {
20                  "name": "Register new user",
21                  "request": {
22                    "method": "POST",
23                    "header": [
24                      {
25                        "key": "Content-Type",
26                        "value": "application/json"
27                      },
28                      {
29                        "key": "Accept",
30                        "value": "application/json"
31                      }
32                    ],
33                    "body": {
34                      "mode": "raw",
35                      "raw": "{\n  \"email\": \"<email>\", \n  \"password\": \"<string>\", \n  \"name\": \"<string>\\n\"",
36                      "options": {
37                        "raw": {
38                          "headerFamily": "json",
39                          "language": "json"
40                        }
41                      }
42                    },
43                    "url": {
44                      "raw": "{{baseUrl}}/api/auth/register",
45                      "host": [

```

Вывод

Использование Swagger и Postman значительно упрощает документирование и тестирование REST-API и снижает вероятность расхождений между кодом и спецификацией.