

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашнее задание №3

Выполнил:

Соловьева П.А.

Группа К3344

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Реализовать автодокументирование средствами Swagger и документацию API средствами Postman для системы управления недвижимостью на базе Node.js/Express/TypeORM.

Ход работы

1. Автодокументирование средствами Swagger

В маршруты добавлена аннотация для Swagger/OpenAPI. Пример на Пользователях:

UserRoutes

```
import { Router } from "express";
import { UserController } from "../controllers/user.controller";
import { authMiddleware } from "../middleware/authMiddleware";

const router = Router();

/**
 * @swagger
 * tags:
 *   name: Users
 *   description: Работа с пользователями
 */

/**
 * @swagger
 * /users:
 *   get:
 *     summary: Получить список пользователей (только авторизованные)
 *     tags: [Users]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: query
 *         name: skip
 *         schema:
 *           type: integer
 *         description: Пропустить N пользователей
 *       - in: query
 *         name: take
 *         schema:
 *           type: integer
 *         description: Количество пользователей для выборки
 *       - in: query
 *         name: email
 *         schema:
 *           type: string
 *           format: email
 *         description: Фильтр по email
 *     responses:
 *       200:
 *         description: Список пользователей
```

```

*         content:
*             application/json:
*                 schema:
*                     type: array
*                     items:
*                         $ref: '#/components/schemas/UserResponse'
*
*     401:
*         description: Unauthorized
*/
router.get("/", authMiddleware, UserController.findAll);

/**
 * @swagger
 * /users/{id}:
 *   get:
 *     summary: Получить пользователя по ID (только авторизованные)
 *     tags: [Users]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
 *           type: string
 *           format: uuid
 *         description: ID пользователя
 *     responses:
 *       200:
 *         description: Пользователь найден
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/UserResponse'
 *       401:
 *         description: Unauthorized
 *       404:
 *         description: Пользователь не найден
 */
router.get("/:id", authMiddleware, UserController.findById);

/**
 * @swagger
 * /users/{id}:
 *   put:
 *     summary: Обновить свои данные пользователя
 *     tags: [Users]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
 *           type: string
 *           format: uuid
 *         description: ID пользователя
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:

```

```

*         schema:
*             type: object
*             properties:
*                 first_name:
*                     type: string
*                 last_name:
*                     type: string
*                 email:
*                     type: string
*                     format: email
*                 phone_number:
*                     type: string
*                     nullable: true
*                 password_hash:
*                     type: string
*                 role:
*                     type: string
*                     enum: [owner, tenant, admin]
*     responses:
*         200:
*             description: Пользователь обновлён
*             content:
*                 application/json:
*                     schema:
*                         $ref: '#/components/schemas/UserResponse'
*         401:
*             description: Unauthorized
*         403:
*             description: Forbidden (пользователь пытается изменить чужие
даннные)
*         404:
*             description: Пользователь не найден
*/
router.put("/:id", authMiddleware, UserController.update);

/**
* @swagger
* /users/{id}:
*   delete:
*     summary: Удалить своего пользователя
*     tags: [Users]
*     security:
*       - bearerAuth: []
*     parameters:
*       - in: path
*         name: id
*         required: true
*         schema:
*           type: string
*           format: uuid
*         description: ID пользователя
*     responses:
*       200:
*         description: Пользователь удалён
*       401:
*         description: Unauthorized
*       403:
*         description: Forbidden (пользователь пытается удалить чужие
даннные)
*       404:
*         description: Пользователь не найден

```

```
*/  
router.delete("/:id", authMiddleware, UserController.delete);  
  
export default router;
```

Swagger.ts

```
import swaggerJsdoc from "swagger-jsdoc";  
import swaggerUi from "swagger-ui-express";  
import { Express } from "express";  
  
export const setupSwagger = (app: Express, title: string, port: number) =>  
{  
  const options: swaggerJsdoc.Options = {  
    definition: {  
      openapi: "3.0.0",  
      info: {  
        title,  
        version: "1.0.0",  
        description: `${title} - RESTful API`,  
      },  
      servers: [  
        { url: process.env.API_URL || 'http://localhost:5001' }  
      ],  
      components: {  
        securitySchemes: {  
          bearerAuth: {  
            type: "http",  
            scheme: "bearer",  
            bearerFormat: "JWT",  
          },  
        },  
        schemas: {  
          UserResponse: {  
            type: "object",  
            properties: {  
              id: { type: "string", format: "uuid" },  
              first_name: { type: "string" },  
              last_name: { type: "string" },  
              email: { type: "string", format: "email" },  
              phone_number: { type: "string", nullable: true },  
              role: { type: "string", enum: ["owner", "tenant", "admin"] },  
              created_at: { type: "string", format: "date-time" },  
              updated_at: { type: "string", format: "date-time" },  
            },  
          },  
          UserCreate: {  
            type: "object",  
            properties: {  
              first_name: { type: "string" },  
              last_name: { type: "string" },  
              email: { type: "string", format: "email" },  
              password: { type: "string" },  
              phone_number: { type: "string", nullable: true },  
              role: { type: "string", enum: ["owner", "tenant", "admin"] },  
            },  
            required: ["first_name", "last_name", "email", "password"],  
          },  
          AuthLogin: {  
            type: "object",  
            properties: {
```

```

        email: { type: "string", format: "email" },
        password: { type: "string" },
    },
    required: ["email", "password"],
},
AuthResponse: {
    type: "object",
    properties: {
        token: { type: "string" },
        user: { $ref: "#/components/schemas/UserResponse" },
    },
},
},
},
},
},
apis: ["/src/routes/**/*.ts"],
};

const specs = swaggerJsdoc(options);
app.use("/api/docs", swaggerUi.serve, swaggerUi.setup(specs));
};

```

Импортирование swagger в index.ts:

```

import express from "express";
import { AppDataSource } from "../config/data-source";

import userRoutes from "../routes/user.routes";
import propertyRoutes from "../routes/property.routes";
import rentalRoutes from "../routes/rental.routes";
import messageRoutes from "../routes/message.routes";
import reviewRoutes from "../routes/review.routes";
import amenityRoutes from "../routes/amenity.routes";
import propertyAmenityRoutes from "../routes/propertyAmenity.routes";

import { loggerMiddleware } from "../middleware/loggerMiddleware";
import { errorMiddleware } from "../middleware/errorMiddleware";
import { setupSwagger } from "../swagger";

const app = express();
app.use(express.json());

// Логирование всех запросов
app.use(loggerMiddleware);

// Роуты
app.use("/api/users", userRoutes);
app.use("/api/properties", propertyRoutes);
app.use("/api/rentals", rentalRoutes);
app.use("/api/messages", messageRoutes);
app.use("/api/reviews", reviewRoutes);
app.use("/api/amenities", amenityRoutes);
app.use("/api/property-amenities", propertyAmenityRoutes);

// Swagger
setupSwagger(app);

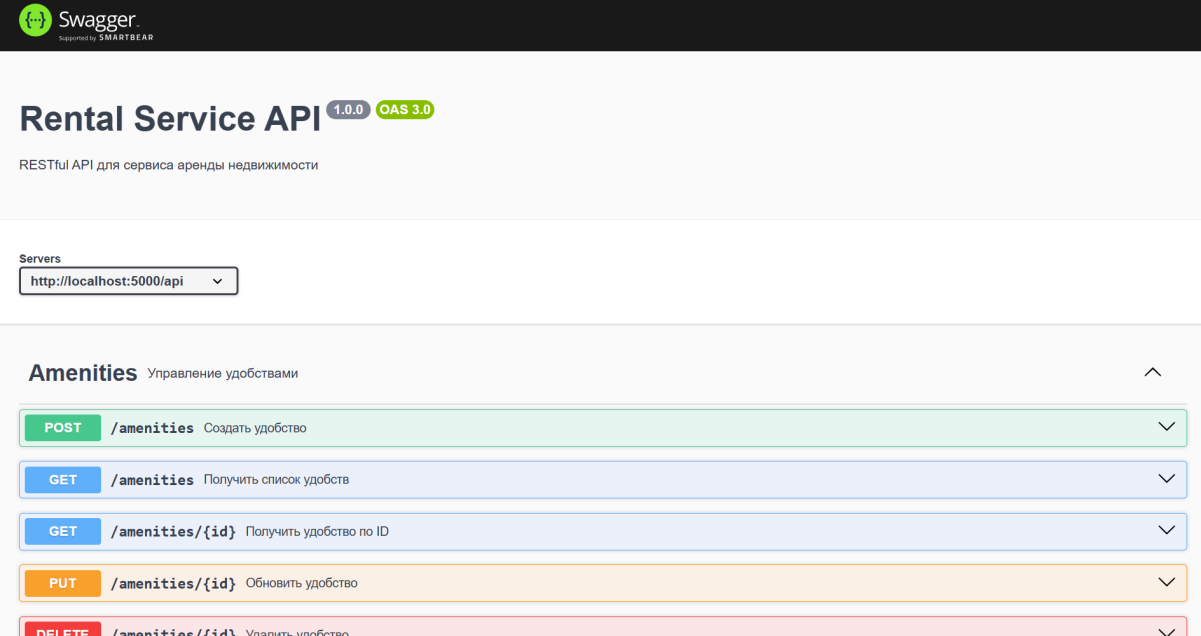
// Глобальная обработка ошибок
app.use(errorMiddleware);

```

```
const PORT = process.env.PORT || 5000;

AppDataSource.initialize()
  .then(() => {
    console.log("Data Source initialized");
    app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
  })
  .catch(err => console.error("Error during Data Source initialization",
err));
```

Документация



The image shows the Swagger UI for the 'Rental Service API'. At the top, the Swagger logo is visible with the text 'Swagger' and 'powered by SMARTBEAR'. Below this, the API title 'Rental Service API' is displayed, followed by version tags '1.0.0' and 'OAS 3.0'. A description in Russian states 'RESTful API для сервиса аренды недвижимости'. A 'Servers' section contains a dropdown menu with the value 'http://localhost:5000/api'. The main content area is titled 'Amenities Управление удобствами' and lists five API endpoints with their methods and descriptions: POST /amenities (Создать удобство), GET /amenities (Получить список удобств), GET /amenities/{id} (Получить удобство по ID), PUT /amenities/{id} (Обновить удобство), and DELETE /amenities/{id} (Удалить удобство). Each entry has a corresponding colored button (green for POST, blue for GET, orange for PUT, red for DELETE) and a dropdown arrow on the right.

2. Документирование с помощью Postman

Примеры запросов:

Rental

POST Create Amenities

GET Get A

GET Get L

PUT Update Amenities

GET Delete Amenities

POST Create Amenities

GET Get A

GET Get F

PUT Update Amenities

No environment

Create Amenities

Save

Share

POST

http://localhost:5000/api/amenities

Send

Params

Authorization

Headers (9)

Body

Scripts

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Cookies

Beautify

```
1 {
2   "name": "Кровать на потолке",
3   "description": "Действительно удобно"
4 }
```

Body

Cookies

Headers (7)

Test Results

201 Created

17 ms

386 B

Visualize

JSON

Preview

```
1 {
2   "name": "Кровать на потолке",
3   "description": "Действительно удобно",
4   "id": "d3248476-2275-44a1-9ae2-a6b7c8ded924"
5 }
```

Update Property

Save

Share

PUT

http://localhost:5000/api/properties/id

Send

Params

Authorization

Headers (9)

Body

Scripts

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

Cookies

Beautify

```
1 {
2   "title": "Обновлённая квартира",
3   "price_per_month": 6000
4 }
```

Body

Cookies

Headers (7)

Test Results

200 OK

16 ms

572 B

Visualize

JSON

Preview

```
1 {
2   "id": "0a3cd9ae-8692-4fa9-bbb4-3179e09a8b83",
3   "title": "Обновлённая квартира",
4   "description": "string",
5   "type": "string",
6   "location": "string",
7   "price_per_month": 6000,
8   "photos": [
9     "string"
10  ],
11   "available_from": "2025-09-18",
12 }
```

Delete Amenities

Save

Share

DELETE

http://localhost:5000/api/amenities/id

Send

Params

Authorization

Headers (7)

Body

Scripts

Settings

Path Variables

Key

Value

Description

Bulk Edit

Key	Value	Description
id	398bcf97-827f-4c02-81fc-57bf35835592	Description

Body

Cookies

Headers (7)

Test Results

200 OK

8 ms

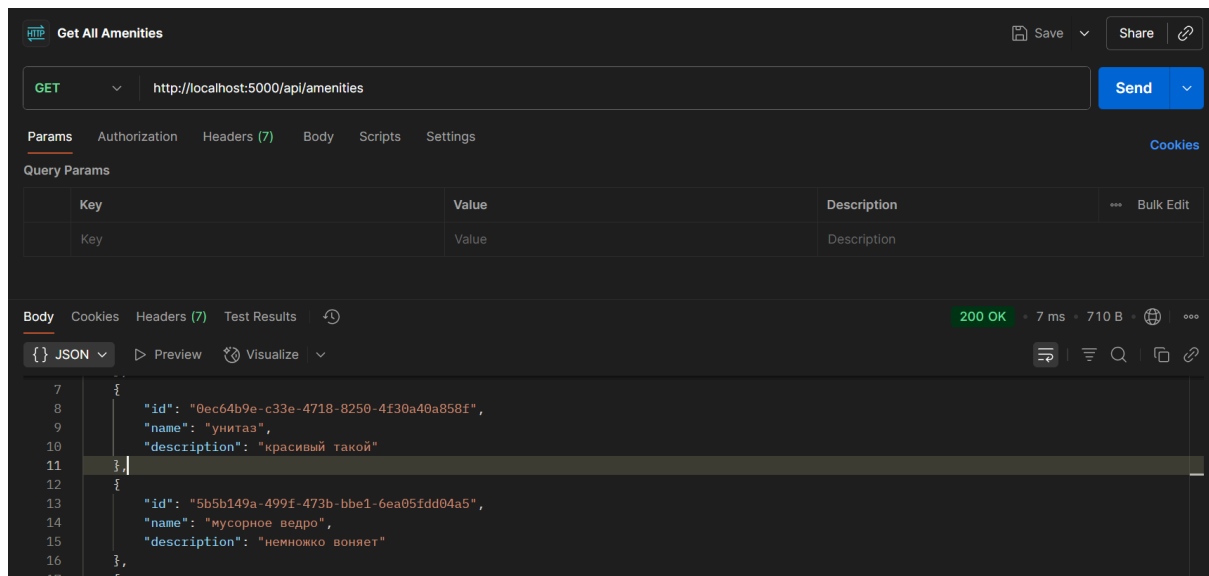
258 B

Visualize

JSON

Preview

```
1 {
2   "raw": [],
3   "affected": 1
4 }
```

Вывод

Swagger (OpenAPI 3.0):

- Настроена интеграция `swagger-jsdoc` и `swagger-ui-express`.
- Для всех сущностей (`User`, `Property`, `Rental`, `Message`, `Review`, `Amenity`, `PropertyAmenity`) описаны схемы (`components.schemas`).
- Для каждого маршрута добавлены аннотации `@swagger`, которые автоматически формируют документацию.
- Документация доступна по маршруту `/api/docs` и позволяет не только просматривать описание API, но и отправлять тестовые запросы.

Postman:

- Создана коллекция с запросами для всех эндпоинтов API.
- Для каждого метода (`GET`, `POST`, `PUT`, `DELETE`) настроены примеры входных и выходных данных.
- Документация в Postman позволяет удобно тестировать работу API и использовать готовые запросы для взаимодействия с сервером.

Таким образом, реализовано удобное и наглядное документирование API, которое обеспечивает:

- быстрый доступ к спецификации;
- прозрачность структуры запросов и ответов;
- простоту тестирования и интеграции для разработчиков и сторонних пользователей.

