Лекция 6: Документирование API. Создаем интерактивную инструкцию с помощью Swagger и OpenAPI

Введение: Боль недокументированного API

Здравствуйте! На предыдущих лекциях мы создали мощный бэкенд. Но код, который не документирован, — это код, написанный наполовину. Представьте, что вы — фронтенд-разработчик. Вы получаете задачу сделать интерфейс для нашего API. Вы просите документацию, а бэкенд-тимлид отвечает вам фразой, которая стоит компаниям миллионов: **"Читай код, там всё понятно"**.

Почему это ужасный ответ?

- **Это медленно:** Чтобы понять один эндпоинт, нужно продраться через роутеры, middleware, контроллеры, валидацию.
- **Это неэффективно:** Время дорогого разработчика тратится на реверс-инжиниринг.
- **Это ведет к ошибкам:** Фронтендер может неправильно понять логику и отправить не те данные.

Хорошая документация API — это формальный контракт между разными частями команды (Frontend, Backend, QA, Mobile). Имея на руках этот контракт, команды могут работать **параллельно** и **независимо**.

Сегодня мы научимся создавать такую документацию не вручную, а **автоматически**, прямо из комментариев в нашем коде, используя промышленные стандарты **OpenAPI** и **Swagger**.

1. OpenAPI vs Swagger: Разбираемся в терминах

Эти два названия часто используются как синонимы, но важно понимать разницу.

- **OpenAPI Specification:** Это сам **стандарт**, "грамматика" языка описания REST API. Он определяет, как в формате YAML или JSON описать эндпоинты, параметры, модели данных, аутентификацию и так далее. Аналогия: это как стандарт музыкальной нотации (скрипичный ключ, ноты, диезы).
- **Swagger:** Это **набор инструментов**, которые "понимают" язык OpenAPI. Самый известный из них — **Swagger UI**.
    - **Swagger UI** — это инструмент, который берет вашу OpenAPI-спецификацию и превращает её в **красивую, интерактивную веб-страницу**, где можно не только почитать про API, но и отправлять к нему тестовые запросы прямо из браузера. Аналогия: это красивые, напечатанные ноты, которые удобно читать музыканту.

Наша цель — научиться генерировать OpenAPI-спецификацию и визуализировать её с помощью Swagger UI.

2. Подход "Documentation-as-Code"

Самая большая проблема с документацией — она почти всегда устаревает. Разработчик меняет что-то в коде (добавляет поле, меняет URL), а обновить отдельный файл с документацией — забывает.

Решение этой проблемы — **держать документацию максимально близко к коду**, который она описывает. Мы будем писать нашу OpenAPI-спецификацию в виде **JSDoc-комментариев** (/** ... */) прямо над каждым обработчиком роута.

Для этого мы будем использовать две npm-библиотеки:

1. swagger-jsdoc: Читает наши комментарии и на лету генерирует из них OpenAPI JSON-спецификацию.
2. swagger-ui-express: Берет эту спецификацию и создает эндпоинт в нашем Express-приложении (например, /api-docs), на котором "живет" красивая страница Swagger UI.

3. Практика: Документируем наше To-Do API

Давайте пошагово добавим документацию к нашему существующему приложению.

Шаг 1: Установка зависимостей

```
npm install swagger-jsdoc swagger-ui-express
```

Шаг 2: Базовая настройка

В нашем главном файле (index.js или app.js) мы должны настроить Swagger.

```
const swaggerJsdoc = require('swagger-jsdoc');
const swaggerUi = require('swagger-ui-express');

const options = {
  definition: {
    openapi: '3.0.0',
    info: {
      title: 'To-Do List API',
      version: '1.0.0',
      description: 'Простое API для управления списком задач',
    },
    servers: [
      { url: 'http://localhost:3000' }, // Адрес нашего сервера
    ],
    // Описываем схему безопасности для JWT
    components: {
      securitySchemes: {
        bearerAuth: {
          type: 'http',
          scheme: 'bearer',
          bearerFormat: 'JWT',
        },
      },
    },
  },
  // Указываем, где искать JSDoc-комментарии с описанием API
  apis: ['./routes/*.js'],
};

const specs = swaggerJsdoc(options);
```

```
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(specs));
```

Теперь, если вы запустите сервер и откроете http://localhost:3000/api-docs, вы увидите базовый интерфейс Swagger UI, но пока без эндпоинтов.

Шаг 3: Документирование моделей (Schemas)

Чтобы не повторять описание структуры наших данных (User, Task) в каждом эндпоинте, мы определим их один раз в специальной секции components/schemas. Лучшее место для этого — прямо в файле роутера, который с ними работает.

```
// routes/users.js или в отдельном файле DTO

/**
 * @openapi
 * components:
 *   schemas:
 *     User:
 *       type: object
 *       properties:
 *         id:
 *           type: integer
 *           description: Уникальный ID пользователя.
 *           example: 1
 *         email:
 *           type: string
 *           format: email
 *           description: Email пользователя.
 *           example: "user@example.com"
 *         name:
 *           type: string
 *           description: Имя пользователя.
 *           example: "Иван"
 */
```

Шаг 4: Документирование эндпоинтов

Теперь самое интересное. Мы будем добавлять JSDoc-комментарии прямо над нашими обработчиками роутов.

Документируем GET /users:

```
/**
 * @openapi
 * /users:
 *   get:
 *     tags: [Users]
 *     summary: Получить список всех пользователей
 *     description: Возвращает массив всех пользователей.
 *     responses:
 *       '200':
 *         description: Успешный ответ со списком пользователей.
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/User' // Ссылка на нашу схему!
 */
router.get('/', async (req, res) => { /* ... */ });
```

Документируем POST /auth/register (с телом запроса):

```
/**
 * @openapi
 * /auth/register:
 *   post:
 *     tags: [Auth]
 *     summary: Регистрация нового пользователя
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required: [email, password]
 *             properties:
 *               email:
 *                 type: string
 *                 format: email
 *               password:
 *                 type: string
 *                 format: password
 *     responses:
 *       '201':
 *         description: Пользователь успешно создан.
 */
router.post('/register', ...);
```

Шаг 5: Документирование безопасности

Теперь укажем, какие эндпоинты требуют аутентификации.

```
/**
 * @openapi
 * /tasks:
 *   post:
 *     tags: [Tasks]
 *     summary: Создать новую задачу
 *     security:
 *       - bearerAuth: [] // Применяем схему 'bearerAuth' к этому эндпоинту
 *     requestBody:
 *       // ... (описание тела запроса)
 *     responses:
 *       // ... (описание ответов)
 */
router.post('/', authMiddleware, ...);
```

Теперь в интерфейсе Swagger UI у этого эндпоинта появится иконка "замочка", а вверху страницы — кнопка "Authorize", куда можно будет вставить свой Bearer-токен для отправки тестовых запросов.

4. Интерактивная магия: Тестируем API из документации

"Киллер-фича" Swagger UI — это возможность **отправлять реальные HTTP-запросы** к вашему API прямо со страницы документации.

1. Вы находите нужный эндпоинт (например, POST /tasks).

2. Нажимаете кнопку "Try it out".
3. Если эндпоинт защищен, нажимаете "Authorize" и вставляете свой JWT.
4. Заполняете параметры или тело запроса в удобном интерфейсе.
5. Нажимаете "Execute".

Swagger UI отправит запрос на ваш запущенный сервер и покажет вам полный ответ: статус-код, заголовки и тело. Это невероятно удобный инструмент для бэкендеров (чтобы быстро протестировать свой код), для фронтендеров (чтобы "поиграться" с API и понять, как оно работает) и для тестировщиков.

Заключение

Мы прошли путь от хаоса "читай код" до профессионального, автоматизированного и интерактивного документирования. Подход "Documentation-as-Code" гарантирует, что ваша документация всегда будет актуальной, так как она живет вместе с кодом. Использование стандартов OpenAPI и инструментов Swagger делает ваше API понятным для всей команды и для внешнего мира.

На следующей лекции мы добавим последний, но самый важный штрих к нашему приложению — **автоматизированные тесты**. Мы построим "сеть безопасности", которая будет гарантировать, что наш код работает так, как описано в нашей новой, красивой документации.