ДИСЦИПЛИНА	Фронтенд и бэкенд разработка
ИНСТИТУТ	ИПТИП
КАФЕДРА	Индустриального программирования
ВИД УЧЕБНОГО МАТЕРИАЛА	Методические указания к практическим занятиям по дисциплине
ПРЕПОДАВАТЕЛЬ	Загородних Николай Анатольевич Краснослободцева Дарья Борисовна
СЕМЕСТР	4 семестр, 2025/2026 уч. год

Практическое занятие 5

Расширенный REST API

Подробнее рассмотрим REST API, а также иструментарий для его проектирования, документирования, тестирования и развертывания - Swagger. Решение практического задания осуществляется внутри соответствующей рабочей тетради, расположенной в СДО.

Что такое Swagger и зачем он нужен?

Swagger (или OpenAPI) - это набор инструментов для проектирования, документирования и тестирования REST API. Вместо того чтобы писать документацию вручную в Wiki или Word-документах, разработчики описывают API в структурированном виде (YAML или JSON), а Swagger автоматически превращает это описание в красивый и интерактивный веб-интерфейс.

Преимущества использования Swagger:

1. Интерактивность: прямо в браузере можно отправлять запросы к API и видеть ответы.
2. Актуальность: документация всегда соответствует коду, если генерируется из аннотаций.
3. Удобство: понятно не только разработчикам, но и тестировщикам, аналитикам и менеджерам.
4. Стандартизация: OpenAPI стал индустриальным стандартом описания API.

В рамках практического занятия мы возьмем готовое приложение (клиент на React и сервер на Express) и добавим к нему автоматическую генерацию документации Swagger на серверной части. В качестве примера данных вместо товаров интернет-магазина будем использовать сущность **пользователь**.

Подготовка бэкенда

В качестве основы возьмем сервер на Express из предыдущего занятия. Мы будем использовать библиотеку `swagger-jsdoc` для генерации спецификации из JSDoc-комментариев и `swagger-ui-express` для отображения интерфейса.

Установка зависимостей

Перейдите в папку с вашим backend-проектом и установите необходимые пакеты:

```
npm i express nanoid
npm i -D swagger-jsdoc swagger-ui-express
```

Создание базового сервера с Swagger

Создадим файл `app.js`, в котором будет описан наш API управления пользователями с полной документацией.

```
const express = require('express');
const { nanoid } = require('nanoid');

// Подключаем Swagger
const swaggerJsdoc = require('swagger-jsdoc');
const swaggerUi = require('swagger-ui-express');

const app = express();
const port = 3000;

// Middleware для парсинга JSON
app.use(express.json());

// Middleware для логирования запросов
app.use((req, res, next) => {
    res.on('finish', () => {
        console.log(`[${new Date().toISOString()}] [${req.method}]
${res.statusCode} ${req.path}`);
        if (req.method === 'POST' || req.method === 'PUT' || req.method ===
'PATCH') {
            console.log('Body:', req.body);
        }
    });
    next();
});

let users = [
    {id: nanoid(6), name: 'Петр', age: 16},
    {id: nanoid(6), name: 'Иван', age: 18},
    {id: nanoid(6), name: 'Дарья', age: 20},
```

```
];

// Swagger definition
// Описание основного API
const swaggerOptions = {
    definition: {
        openapi: '3.0.0',
        info: {
            title: 'API управления пользователями',
            version: '1.0.0',
            description: 'Простое API для управления пользователями',
        },
        servers: [
            {
                url: `http://localhost:${port}`,
                description: 'Локальный сервер',
            },
        ],
    },
    // Путь к файлам, в которых мы будем писать JSDoc-комментарии (наш
текущий файл)
    apis: ['./app.js'],
};

const swaggerSpec = swaggerJsdoc(swaggerOptions);

// Подключаем Swagger UI по адресу /api-docs
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));

/**
 * @swagger
 * components:
 *   schemas:
 *     User:
 *       type: object
 *       required:
 *         - name
 *         - age
 *       properties:
 *         id:
 *           type: string
 *           description: Автоматически сгенерированный уникальный ID
пользователя
 *         name:
 *           type: string
 *           description: Имя пользователя
 *         age:
 *           type: integer
 *           description: Возраст пользователя
 *       example:
```

```
 *         id: "abc123"
 *         name: "Петр"
 *         age: 16
 */

// Функция-помощник для получения пользователя из списка
function findUserOr404(id, res) {
    const user = users.find(u => u.id == id);
    if (!user) {
        res.status(404).json({ error: "User not found" });
        return null;
    }
    return user;
}

/**
 * @swagger
 * /api/users:
 *   post:
 *     summary: Создает нового пользователя
 *     tags: [Users]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - name
 *               - age
 *             properties:
 *               name:
 *                 type: string
 *               age:
 *                 type: integer
 *     responses:
 *       201:
 *         description: Пользователь успешно создан
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/User'
 *       400:
 *         description: Ошибка в теле запроса
 */
app.post("/api/users", (req, res) => {
    const { name, age } = req.body;

    if (!name || age === undefined) {
        return res.status(400).json({ error: "Name and age are required" });
```

```
    }

    const newUser = {
        id: nanoid(6),
        name: name.trim(),
        age: Number(age),
    };

    users.push(newUser);
    res.status(201).json(newUser);
});

/**
 * @swagger
 * /api/users:
 *   get:
 *     summary: Возвращает список всех пользователей
 *     tags: [Users]
 *     responses:
 *       200:
 *         description: Список пользователей
 *         content:
 *           application/json:
 *             schema:
 *               type: array
 *               items:
 *                 $ref: '#/components/schemas/User'
 */
app.get("/api/users", (req, res) => {
    res.json(users);
});

/**
 * @swagger
 * /api/users/{id}:
 *   get:
 *     summary: Получает пользователя по ID
 *     tags: [Users]
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: string
 *         required: true
 *         description: ID пользователя
 *     responses:
 *       200:
 *         description: Данные пользователя
 *         content:
 *           application/json:
```

```
 *             schema:
 *               $ref: '#/components/schemas/User'
 *       404:
 *         description: Пользователь не найден
 */
app.get("/api/users/:id", (req, res) => {
    const user = findUserOr404(req.params.id, res);
    if (!user) return;

    res.json(user);
});

/**
 * @swagger
 * /api/users/{id}:
 *   patch:
 *     summary: Обновляет данные пользователя
 *     tags: [Users]
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: string
 *         required: true
 *         description: ID пользователя
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               name:
 *                 type: string
 *               age:
 *                 type: integer
 *     responses:
 *       200:
 *         description: Обновленный пользователь
 *         content:
 *           application/json:
 *             schema:
 *               $ref: '#/components/schemas/User'
 *       400:
 *         description: Нет данных для обновления
 *       404:
 *         description: Пользователь не найден
 */
app.patch("/api/users/:id", (req, res) => {
    const id = req.params.id;
```

```
    const user = findUserOr404(id, res);
    if (!user) return;

    // Нельзя PATCH без полей
    if (req.body?.name === undefined && req.body?.age === undefined) {
        return res.status(400).json({
            error: "Nothing to update",
        });
    }

    const { name, age } = req.body;

    if (name !== undefined) user.name = name.trim();
    if (age !== undefined) user.age = Number(age);

    res.json(user);
});

/**
 * @swagger
 * /api/users/{id}:
 *   delete:
 *     summary: Удаляет пользователя
 *     tags: [Users]
 *     parameters:
 *       - in: path
 *         name: id
 *         schema:
 *           type: string
 *         required: true
 *         description: ID пользователя
 *     responses:
 *       204:
 *         description: Пользователь успешно удален (нет тела ответа)
 *       404:
 *         description: Пользователь не найден
 */
app.delete("/api/users/:id", (req, res) => {
    const id = req.params.id;

    const exists = users.some((u) => u.id === id);
    if (!exists) return res.status(404).json({ error: "User not found" });

    users = users.filter((u) => u.id !== id);

    // Правильнее 204 без тела
    res.status(204).send();
});
```

```
// 404 для всех остальных маршрутов
app.use((req, res) => {
    res.status(404).json({ error: "Not found" });
});

// Глобальный обработчик ошибок (чтобы сервер не падал)
app.use((err, req, res, next) => {
    console.error("Unhandled error:", err);
    res.status(500).json({ error: "Internal server error" });
});

// Запуск сервера
app.listen(port, () => {
    console.log(`Сервер запущен на http://localhost:${port}`);
    console.log(`Swagger UI доступен по адресу http://localhost:${port}/api-
docs`);
});
```

Документирование API с помощью JSDoc

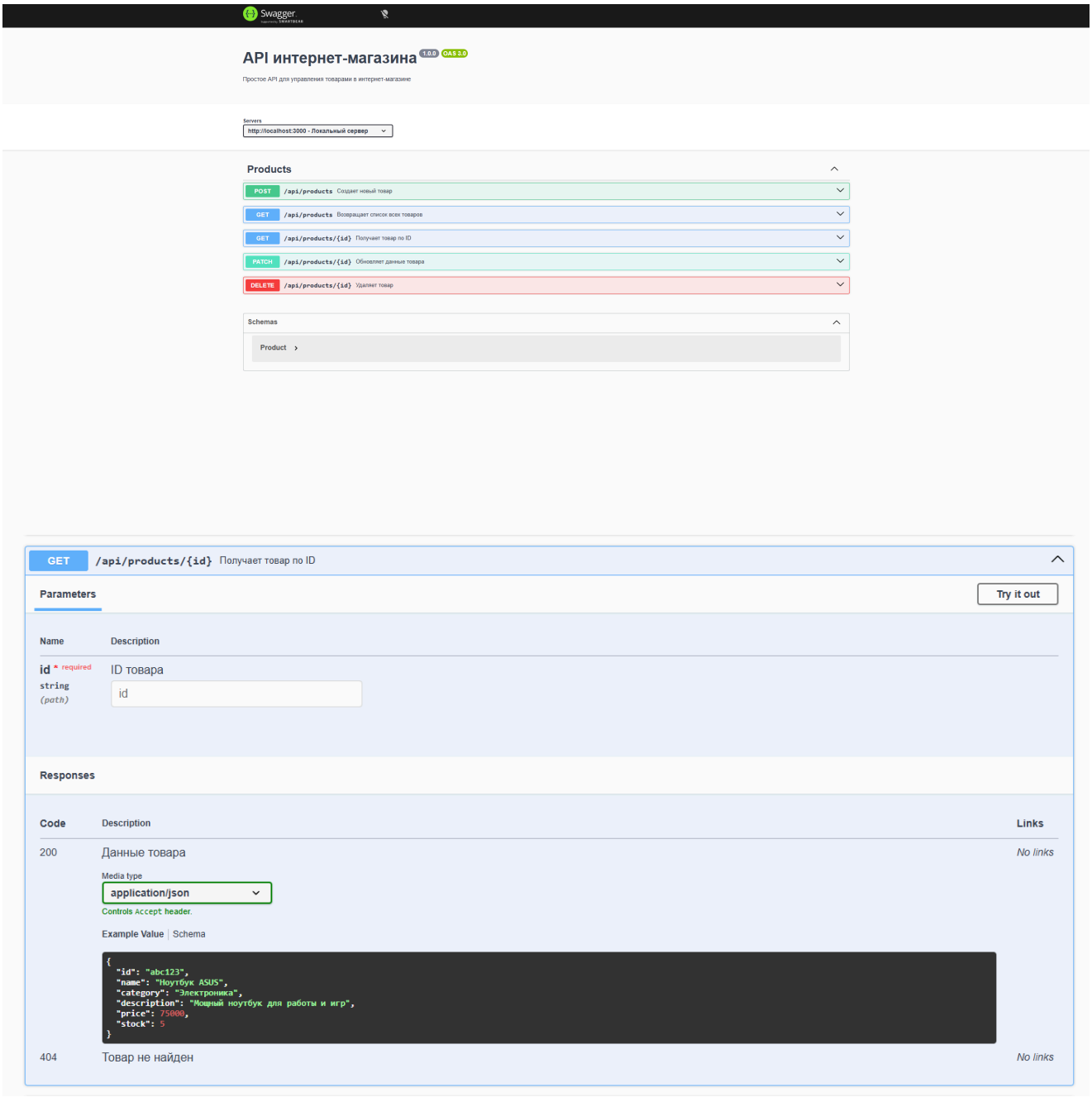
Обратите внимание на комментарии, начинающиеся с `/** @swagger */`. Это и есть JSDoc-аннотации, которые `swagger-jsdoc` превратит в спецификацию.

1. **Описание схемы (`components/schemas/User`):**
   Мы описали, как выглядит объект "Пользователь". Какие поля у него есть, какие из них обязательные, и привели пример. Теперь в других частях документации мы можем ссылаться на эту схему через `$ref: '#/components/schemas/User'`.
2. **Описание эндпоинтов:**
   Для каждого HTTP-метода (`POST /api/users`, `GET /api/users`) мы добавили:
   - `summary` - краткое описание.
   - `tags` - группировка методов в интерфейсе Swagger.
   - `parameters` - описание параметров пути (например, `id`).
   - `requestBody` - описание тела запроса (для POST и PATCH).
   - `responses` - описание возможных ответов от сервера (коды 200, 201, 404 и т.д.) и примеры данных, которые придут.

Запуск и просмотр документации

1. Запустите сервер командой `node app.js`.
2. Откройте браузер и перейдите по адресу `http://localhost:3000/api-docs`.

Вы увидите интерактивную документацию:



Теперь вы можете изучить структуру API и развернуть любой запрос (например, `GET /api/users`). Также можете нажать кнопку "Try it out", чтобы отправить реальный запрос к вашему API и увидеть ответ прямо в браузере.

Работа с фронтендом

Поскольку сервер использует эндпоинты `/api/users`, фронтенд из предыдущей работы должен обращаться к ним.

Основные изменения в `src/api/index.js`:

```
import axios from "axios";

const apiClient = axios.create({
    baseURL: "http://localhost:3000/api",
    headers: {
        "Content-Type": "application/json",
```

```
        "accept": "application/json",
    }
});

export const api = {

    createUser: async (user) => {
        let response = await apiClient.post("/users", user);
        return response.data;
    },

    getUsers: async () => {
        let response = await apiClient.get("/users");
        return response.data;
    },

    getUserById: async (id) => {
        let response = await apiClient.get(`/users/${id}`);
        return response.data;
    },

    updateUser: async (id, user) => {
        let response = await apiClient.patch(`/users/${id}`, user);
        return response.data;
    },

    deleteUser: async (id) => {
        await apiClient.delete(`/users/${id}`);
    }
}
```

Соответственно, в компонентах (`UsersPage.jsx` , `UserItem.jsx` , `UserModal.jsx`) нужно использовать:

- `users` — список пользователей.
- `name` — имя пользователя.
- `age` — возраст пользователя.

Практическое задание

Необходимо доработать задание из практического занятия №4: подключите к нему `swagger-jsdoc` и `swagger-ui-express` , опишите с помощью JSDoc-аннотаций схему пользователя (`User`) и все CRUD-операции (`GET` , `POST` , `GET/:id` , `PATCH/:id` , `DELETE`).
Убедитесь, что документация доступна по адресу `/api-docs` и работает в интерактивном режиме (можно отправить тестовый запрос).

Формат отчета

В качестве ответа на задание необходимо прикрепить ссылку на репозиторий с реализованной практикой. Ссылка подгружается в соответствующий раздел СДО: Задания для самостоятельной работы -> Практические занятия 5-6.