

|  |  |
| --- | --- |
| https://lh7-us.googleusercontent.com/QuBaagabzZYLr7U3ZbYOtMusd5FbWWxIvMUTN8jrLHzSLMg534z9gXRTIG1Us4i_lOwmWlaBxKedNt-SQ26dm4WmyqwjGDmEO6z8GE3QrZosqvHM88J2EFeVf1u0GzyCZQlhWmp1Zeo85tKo4LJVXQ | МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)» |

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Отчёт по лабораторной работе №19  
по учебной дисциплине «Web-технологии»

на тему «Сервер на Node.js»

Выполнил  
студент группы М3О-119БВ-24

Нарзиев А.Т.

Принял

Ассистент каф.304 Борисов А.И.

Москва  
2024

# Содержание

[Задание 3](#_Toc194427981)

[Решение 4](#_Toc194427982)

[1. Разработка сервера HTTP 4](#_Toc194427983)

[1.1. Сервер – server.js: 4](#_Toc194427984)

[1.2. Страница с текстом – index.html: 5](#_Toc194427985)

[1.3. Страница с полем ввода – input.html: 5](#_Toc194427986)

[1.4. Страница с картинкой – image.html 6](#_Toc194427987)

[1.5. Страница с таблицей – table.html: 6](#_Toc194427988)

[1.6. Пример работы сервера: 6](#_Toc194427989)

[2. Разработка стилей 8](#_Toc194427990)

[3. Обработка POST запросов 8](#_Toc194427991)

[4. Разработка шаблонов 8](#_Toc194427992)

[5. Логирование 8](#_Toc194427993)

[6. Аутентификация пользователей 8](#_Toc194427994)

[7. Упрощение маршрутизации 8](#_Toc194427995)

[8. Кэширование ресурсов 8](#_Toc194427996)

[Вывод 9](#_Toc194427997)

# Задание

# Задание №19. Web-сервер на Node.js

1. Разработать программу HTTP-сервер с помощью Node.js. Для создания сервера подключить модуль http. Поднять сервер на порту 8080 в состоянии listen.

Для сервера необходимо создать 4 HTML страницы (4 файла, необходимо использовать модули fs и path) с различным наполнением:

* Текст,
* Поле ввода,
* Таблица,
* Картинка.

Картинка будет использоваться для отображения страницы с кодами 404 и 5xx. Оставшиеся 3 страницы будут отображаться с кодом 200 при различных URL-путях.

На каждой странице должен быть переход на следующую через HTML-теги.

1. В отчете привести пример работы web-сервера и отображения всех страниц. (можно отдельные фрагменты)
2. Добавить файл стилей CSS.
3. В отчете привести пример работы web-сервера и отображения всех страниц. (можно отдельные фрагменты) и код всех файлов сервера. Также привести запросы и ответы из вкладки Network в браузере.
4. Реализовать обработку POST запросов на сервере Node.js для отправки данных с формы на одну из HTML страниц. Сохранить полученные данные в файл или базу данных и вывести сообщение об успешной отправке данных на экран.
5. Реализовать использование шаблонизатора (например, EJS или Handlebars) для упрощения создания HTML страниц на сервере Node.js. Создать шаблоны для всех страниц и использовать их при генерации контента для клиента.
6. Настроить логирование доступа и ошибок на сервере Node.js с использованием модуля morgan. Проанализировать логи для выявления информации о запросах, статусах ответов и возможных проблемах.
7. Реализовать аутентификацию пользователей на сервере Node.js с использованием Passport.js и стратегии локальной аутентификации. Обеспечить защиту доступа к определенным страницам или действиям с помощью аутентификации по паролю.
8. Использовать модуль Express.js для упрощения создания маршрутов на сервере Node.js. Создать REST API для работы с данными (например, CRUD операции) и протестировать его работу с помощью Postman или другого инструмента.
9. Реализовать кэширование ресурсов (изображения, CSS, JS) на сервере Node.js с помощью модуля node-cache или других методов. Оценить влияние кэширования на скорость загрузки страниц и снижение нагрузки на сервер.

# Решение

## Разработка сервера HTTP

### Сервер – server.js:

// Импортируем необходимые модули

const http = require('http');// Модуль для создания HTTP сервера

const fs = require('fs');    // Модуль для работы с файловой системой

const path = require('path');// Модуль для работы с путями файловой системы

const PORT = 8080; // Порт, на котором будет работать сервер

// Создаем HTTP сервер

const server = http.createServer((req, res) => {

    let filePath = ''; // Путь к файлу

    // Определяем, какой файл нужно вернуть в зависимости от URL запроса

    switch (req.url) {

        // Главная страница

        case '/':

            filePath = path.join(\_\_dirname, 'public', 'index.html');

            break;

        // Страница ввода

        case '/input':

            filePath = path.join(\_\_dirname, 'public', 'input.html');

            break;

        // Страница таблицы

        case '/table':

            filePath = path.join(\_\_dirname, 'public', 'table.html');

            break;

        // Страница с изображением

        case '/image':

            filePath = path.join(\_\_dirname, 'public', 'image.html');

            break;

        case '/404-image.jpg':

            // Изображение для 404 ошибки

            filePath = path.join(\_\_dirname, 'public', '404-image.jpg');

            break;

        default:

            // Если запрашиваемая страница не найдена, возвращаем 404 ошибку

            res.writeHead(404, { 'Content-Type': 'text/html' });

            res.end('<h1>404 - Страница не найдена</h1><a href="/">Вернуться на главную страницу</a>');

            return;

    }

    // Читаем файл по указанному пути

    fs.readFile(filePath, (err, content) => {

        if (err) {

            // Если произошла ошибка при чтении файла, возвращаем 500 ошибку

            res.writeHead(500, { 'Content-Type': 'text/html' });

            res.end('<h1>500 - Ошибка сервера</h1>');

            return;

        }

        // Если файл успешно прочитан, возвращаем его содержимое с кодом 200

        res.writeHead(200, { 'Content-Type': 'text/html' });

        res.end(content); // Отправляем содержимое файла

    });

});

// Обработчик ошибок

server.on('error', (err) => {

    console.error('Ошибка сервера:', err);

});

// Запуск сервера

server.listen(PORT, '172.17.254.41', () => {

    console.log(`Сервер запущен на http://172.17.254.41:${PORT}/`);

});

### Страница с текстом – index.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Главная страница</title>

</head>

<body>

    <header>

        <h1>Добро пожаловать на главную страницу!</h1>

        <nav>

            <ul>

                <li><a href="#about">О сайте</a></li>

                <li><a href="#services">Услуги</a></li>

                <li><a href="#contact">Контакты</a></li>

            </ul>

        </nav>

    </header><main>

        <section id="about">

            <h2>О сайте</h2>

            <p>Это сайт для представления информации и материалов

                по всему курсу предмета Web-технологии</p>

        </section>

        <section id="services">

            <h2>Услуги</h2>

            <p>Здесь мы собрали конспекты всех лекций, практических

                занятий, лабораторных работ и материалов для

                подготовки к экзамену.</p>

        </section>

        <section id="contact">

            <h2>Контакты</h2>

            <p>Свяжитесь с нами!<br><br>

                <a href="https://github.com/ZeroD1vision/Web-tech">Наш репозиторий Git</a><br>

                <a href="https://t.me/+Yku9N\_IK9mEzOTZi">Наш Telegram-канал</a><br>

                <a href="mailto:anatolypozd739@gmail.com">Написать письмо администратору.</a><br>

            </p>

        </section>

    </main>

    <a href="/input">Перейти на страницу с полем ввода</a><br>

    <a href="/table">Перейти на страницу с таблицей</a><br>

    <a href="/image">Перейти на страницу с картинкой</a>

    <footer>

        <p>© 2025 Нарзиев Артемий</p>

    </footer>

</body>

</html>

### Страница с полем ввода – input.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Страница с полем ввода</title>

</head>

<body>

    <h1>Страница с полем ввода</h1>

    <form>

        <label for="input">Введите что-нибудь:</label>

        <input type="text" id="input" name="input">

        <input type="submit" value="Отправить">

    </form>

    <a href="/">Вернуться на главную страницу</a>

    </main>

    <footer>

        <p>© 2025 Нарзиев Артемий</p>

    </footer>

</body>

</html>

### Страница с картинкой – image.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Страница с картинкой</title>

</head>

<body>

    <h1>Страница с картинкой</h1>

    <img src="/404-image.jpg" alt="Ошибка 404" style="width:300px;">

    <a href="/">Вернуться на главную страницу</a>

</body>

</html>

### Страница с таблицей – table.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Страница с таблицей</title>

</head>

<body>

    <h1>Страница с таблицей</h1>

    <table border="2">

        <tr>

            <th>Имя</th>

            <th>Фамилия</th>

        </tr>

        <tr>

            <td>Артемий</td>

            <td>Нарзиев</td>

        </tr>

        <tr>

            <td>Артемий</td>

            <td>Кошкин</td>

        </tr>

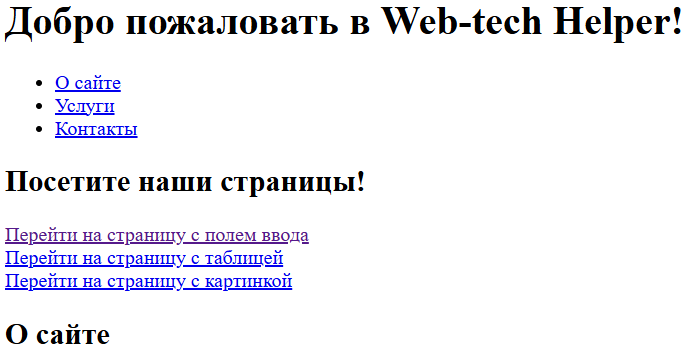
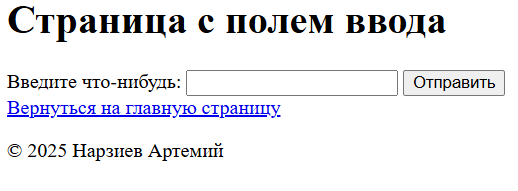
    </table>

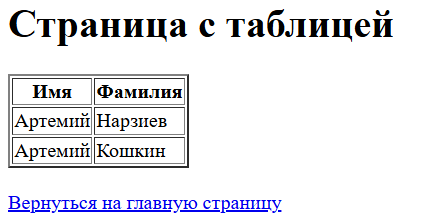
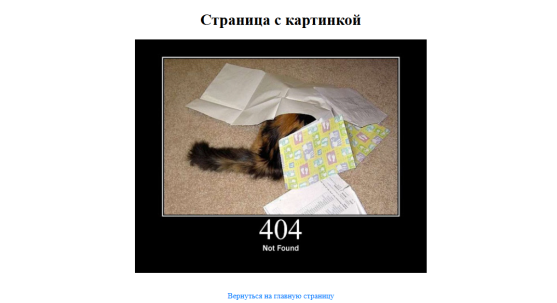
    <a href="/">Вернуться на главную страницу</a>

</body>

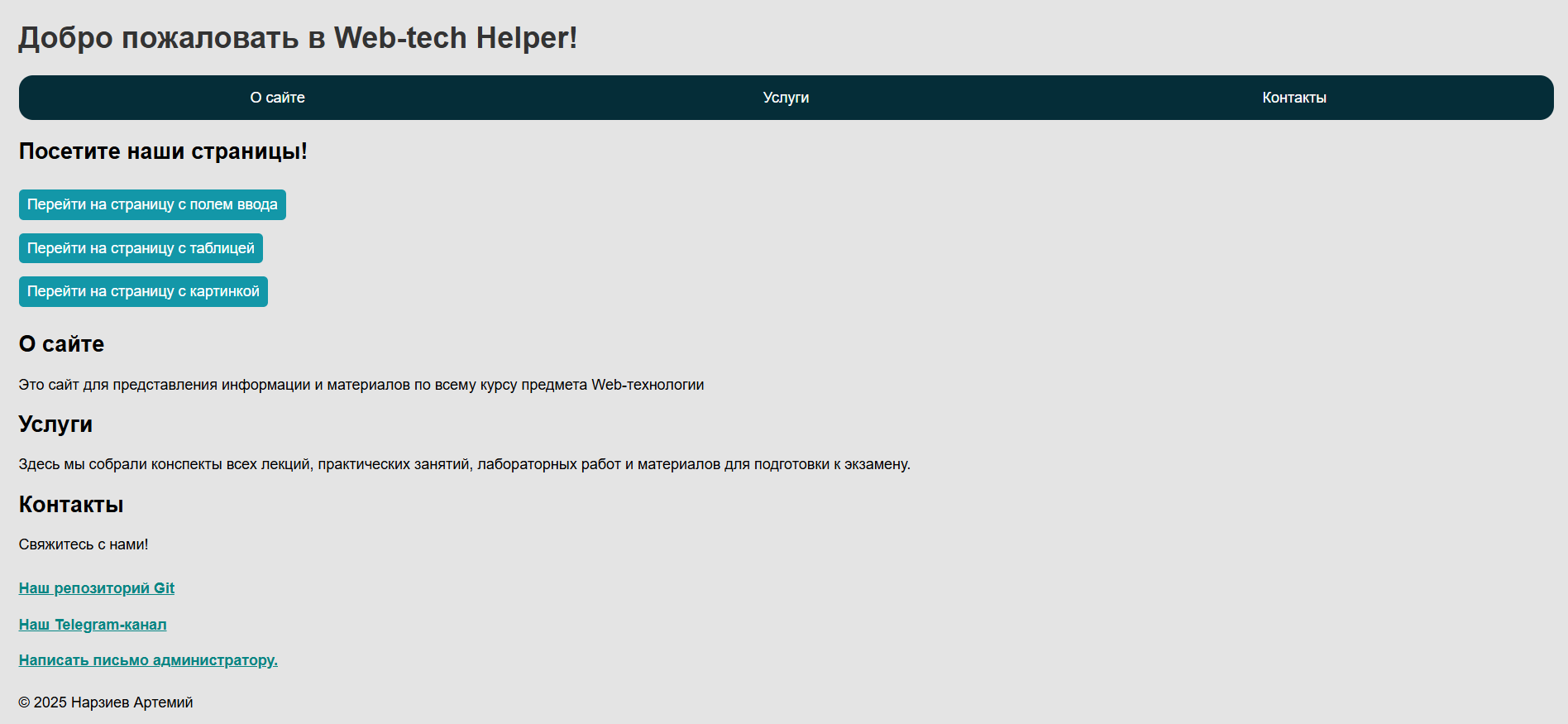
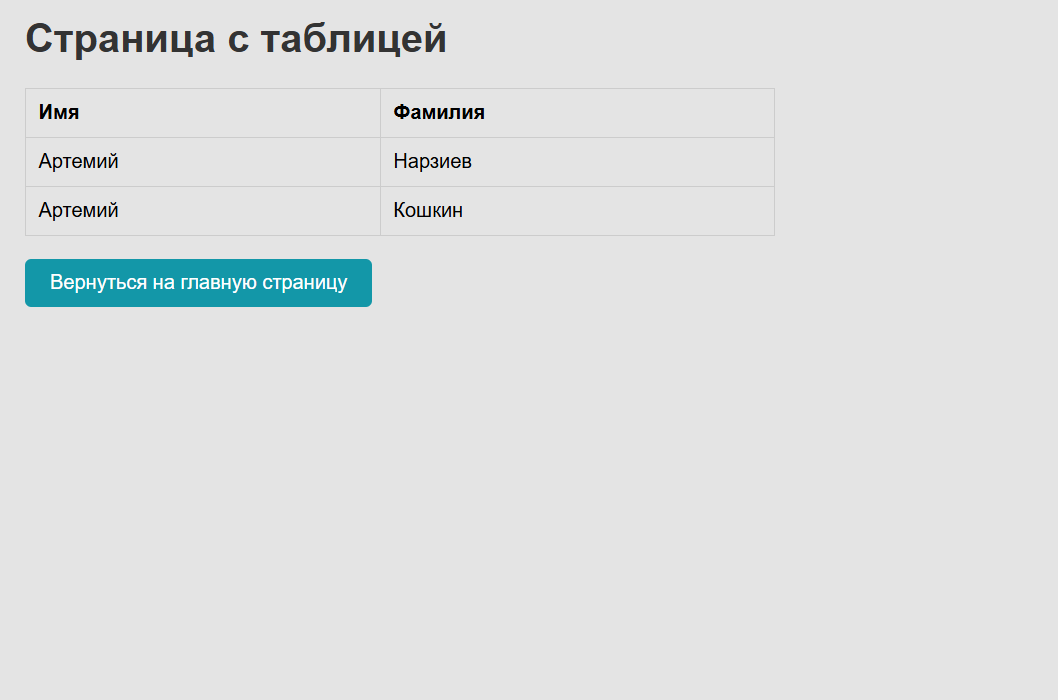
</html>

### Пример работы сервера:

## Разработка стилей

### Файл стилей – styles.css:

body {

    font-family: Arial, sans-serif;

    background-color: #e4e4e4;

    margin: 0; /\* Убирает внешние отступы \*/

    padding: 20px; /\* Добавляет внутренние отступы по 20 пикселей \*/

}

h1 {

    color: #333;

}

a {

    display: inline-block;

    color: white;

    text-decoration: none;

    border-radius: 5px;

}

table {

    width: 100%;

    border-collapse: collapse;

    margin-top: 20px;

}

table, th, td {

    border: 1px solid #ccc;

}

th, td {

    padding: 10px;

    text-align: left;

}

/\* Стили для навигационного меню \*/

nav {

    background-color: #052d38; /\* Темно-серый фон \*/

    padding: 5px; /\* Отступы вокруг меню \*/

    border-radius: 15px; /\* Скругленные углы \*/

}

nav ul {

    list-style-type: none; /\* Без маркеров списка\*/

    margin: 0; /\* Без маргинов \*/

    padding: 0; /\* Без отступов \*/

    display: flex; /\* В строку \*/

    justify-content: space-around; /\* Расположение элементов равномерно \*/

}

nav li {

    flex: 20; /\* Позволяет элементам списка занимать равное пространство \*/

    margin: 0 10px; /\* Отступы между пунктами меню \*/

}

nav a {

    display: flex;

    justify-content:center; /\*Центрируем содержимое по горизонтали \*/

    align-items: center; /\* Центрируем содержимое по вертикали \*/

    color: white;

    text-decoration: none;

    padding: 10px 0px;

    border-radius: 10px;

    transition: background-color 1s;

    height: 50%; /\* Задаем высоту 100% для заполнения области \*/

}

nav a:hover {

    background-color: #10171e;

    color: white;

}

/\* Стили для ссылок на другие страницы сайта \*/

.link-page {

    display: inline-block;

    margin: 7px 0;

    padding: 7px 9px;

    background-color: #1397a8; /\* Синий фон \*/

    color: white; /\* Белый текст \*/

    text-decoration: none; /\* Убираем подчеркивание \*/

    border-radius: 5px; /\* Скругленные углы \*/

    transition: background-color 0.3s; /\* Плавный переход цвета фона \*/

}

.link-page:hover {

    background-color: #053c49; /\* Темно-синий фон при наведении \*/

}

/\* Стили для ссылок на контакты \*/

.link-contact {

    color: #068684; /\* Зеленый текст \*/

    text-decoration: underline; /\* Подчеркивание \*/

    font-weight: bold; /\* Жирный текст \*/

    margin: 10px 0;

}

.link-contact:hover {

    color: #074442; /\* Темно-зеленый текст при наведении \*/

}

/\* Стили для кнопки "Вернуться на главную страницу" \*/

a.button {

    display: inline-block; /\* Делаем ссылку блочным элементом \*/

    background-color: #1397a8; /\* Цвет фона \*/

    color: white; /\* Цвет текста \*/

    padding: 10px 20px; /\* Отступы \*/

    text-decoration: none; /\* Убираем подчеркивание \*/

    border-radius: 5px; /\* Закругление углов \*/

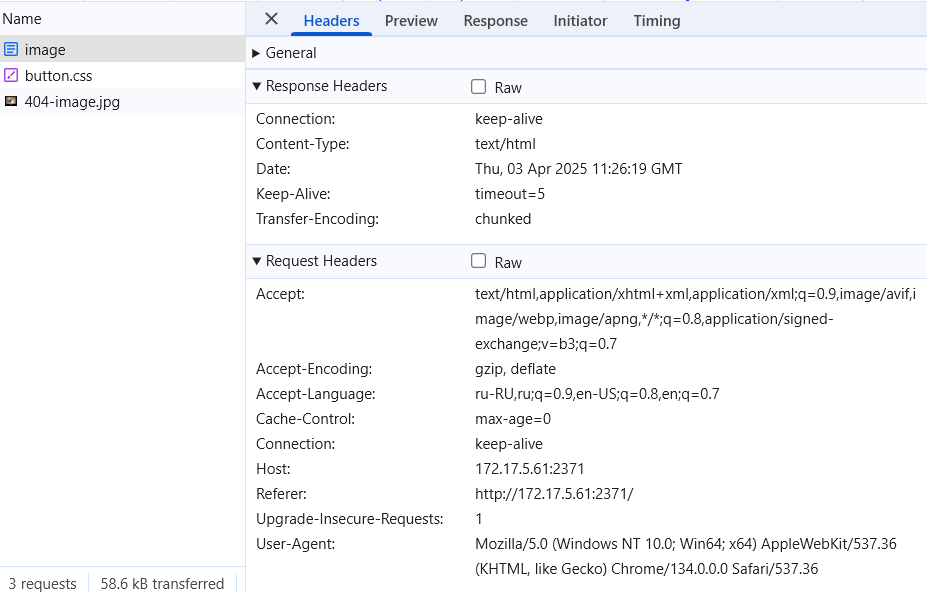
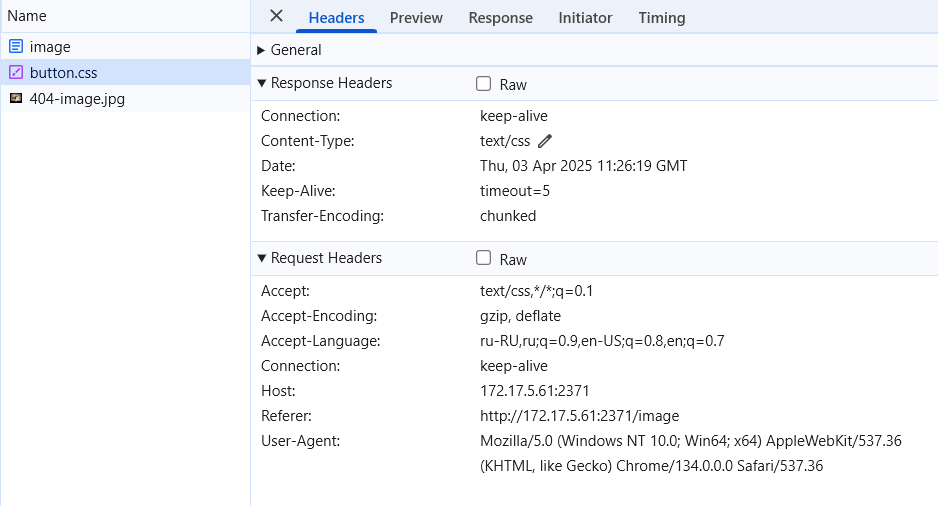
    transition: background-color 0.3s; /\* Плавный переход при наведении \*/

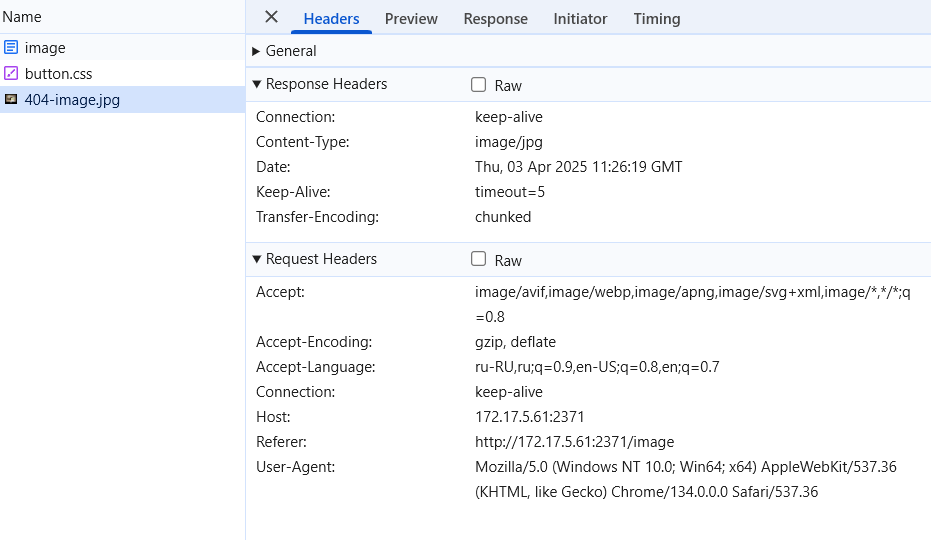
}

a.button:hover {

    background-color: #066462; /\* Цвет фона при наведении \*/}

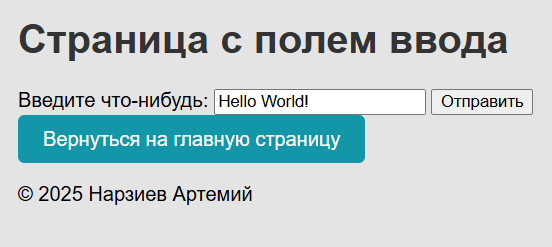
* 1. *Запросы и ответы (Network):*

********

**

## Обработка POST запросов

### Пример ввода

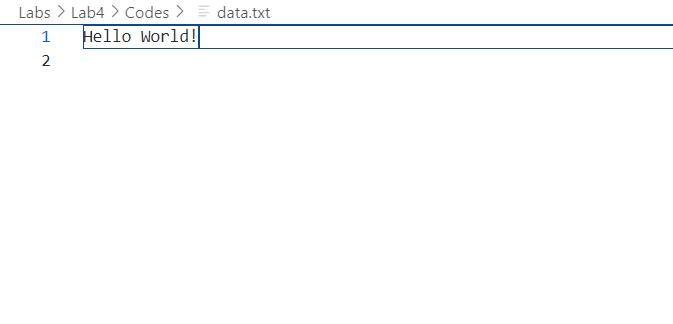


### Результат ввода

*Консоль:*



*Файл data.txt:*



## Разработка шаблонов

Разработка шаблонов производилась на примере нескольких новых страниц с товарами для студентов разных курсов, использующих один шаблон, но с заменой некоторых данных, конкретнее, самих товаров. Далее приведены примеры использования шаблонов в работе сервера и отображении некоторых страниц.

### Пример файла- шаблона – production.ejs:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title><%= title %></title>

    <meta name="description" content="<%= description %>">

    <link rel="stylesheet" href="./css/styles.css">

</head>

<body>

    <header>

        <h1><%= title %></h1>

    </header>

    <main>

        <section>

            <h2>Описание</h2>

            <p><%= description %></p>

        </section>

        <section>

            <h2>Наши товары</h2>

            <ul>

                <% products.forEach(product => { %>

                    <li>

                        <strong><%= product.name %></strong>: <%= product.price %>

                    </li>

                <% }); %>

            </ul>

        </section>

    </main>

    <footer>

        <p>© 2025 Нарзиев Артемий</p>

    </footer>

</body>

</html>

### Пример части файла server-templates.js:

Эта часть отвечает за формирование двух страниц по одному шаблону.

. . .

// Страница "Для первокурсников"

            case '/production1':

                const production1Data = {

                    title: 'Продукция для первокурсников',

                    description: 'Мы предлагаем разнообразные товары для первокурсников.',

                    products: [

                        { name: 'Учебник по программированию', price: '500₽' },

                        { name: 'Блокнот для записей', price: '150₽' },

                        { name: 'Ручка', price: '50₽' }

                    ]

                };

                renderTemplate('production', production1Data, res); // Используем шаблон index.ejs с данными для страницы "О нас"

                break;

            // Страница "Для второкурсников"

            case '/production2':

                const production2Data = {

                    title: 'Продукция для второкурсников',

                    description: 'Мы предлагаем разнообразные товары для второкурсников.',

                    products: [

                        { name: 'Учебник по алгоритмам', price: '600₽' },

                        { name: 'Курс по веб-разработке', price: '2000₽' },

                        { name: 'Математика для программистов', price: '700₽' }

                    ]

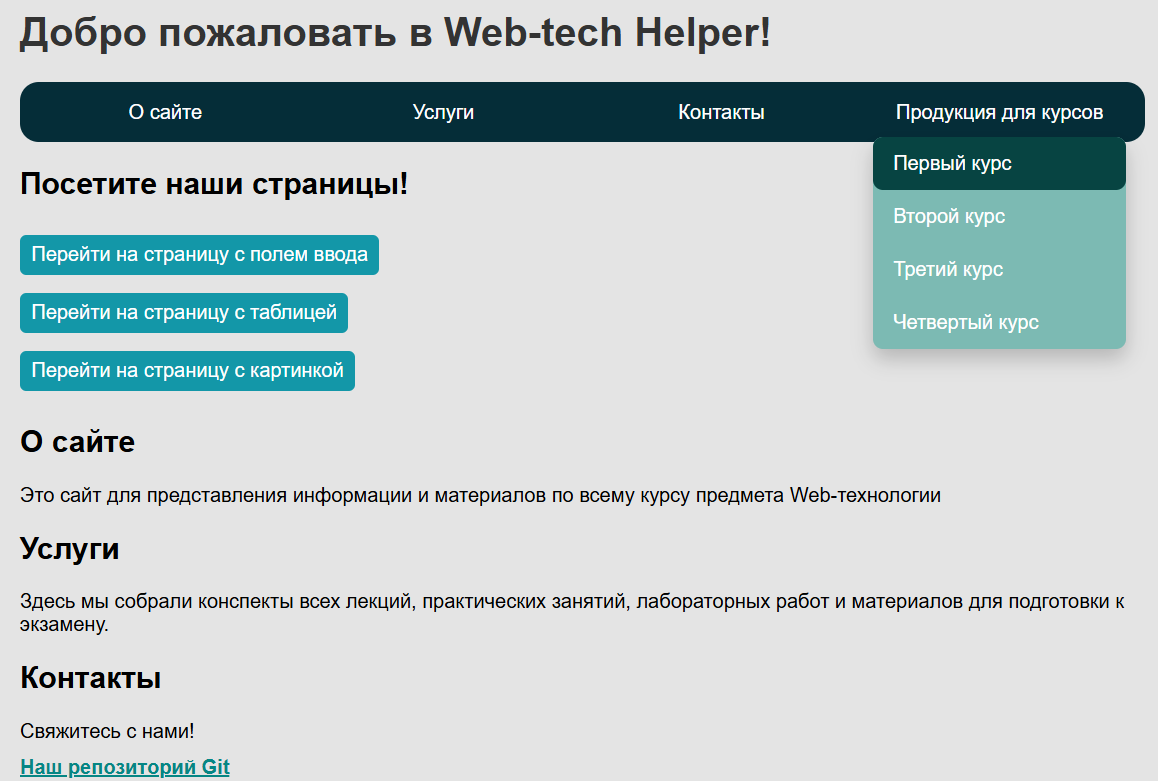
                };

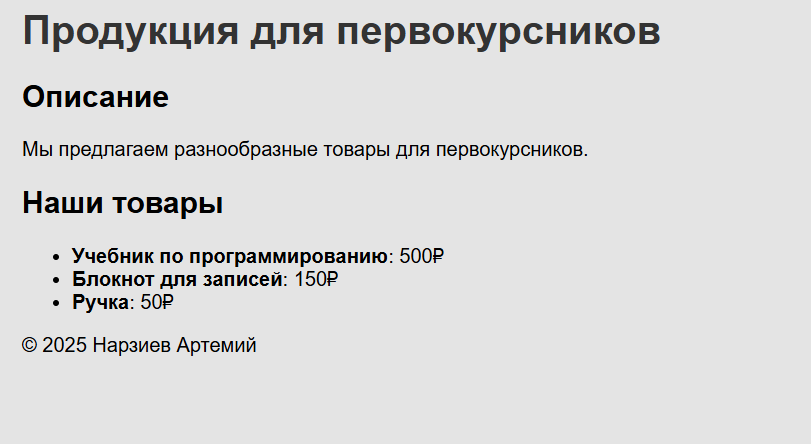
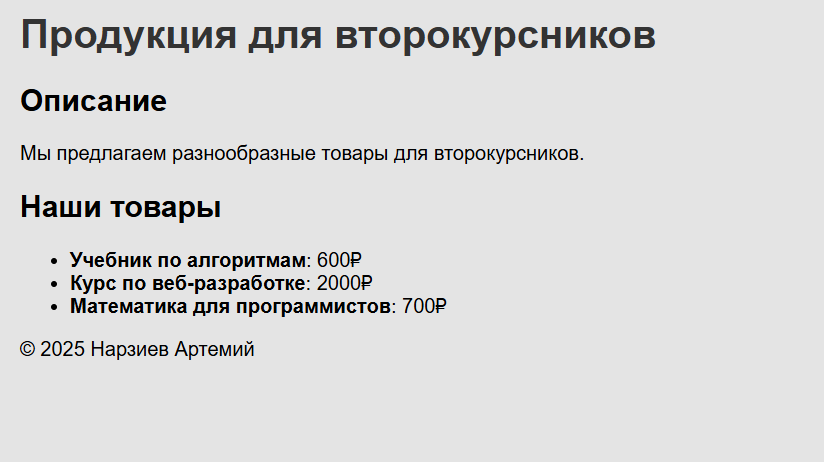
                renderTemplate('production', production2Data, res); // Используем шаблон index.ejs с данными для страницы "О нас"

                break;

. . .

### Примеры работы шаблонов:



## Логирование

### Файл для логирования действий - access.log:

## 

### Анализ лог-файла:

**Общая информация**

1. **IP-адрес**: Все запросы поступают от одного и того же IP-адреса 172.17.5.61, что может указывать на локальный тест или использование одного устройства.
2. **Временные метки**: Все запросы происходят 6 апреля 2025 года, в пределах одной минуты, что также указывает на тестирование или использование приложения в короткий промежуток времени.

**Статусы ответов**

* **200 OK**: Большинство запросов (10 из 12) возвращают статус 200, что означает успешное выполнение запросов.
* **404 Not Found**: Два запроса возвращают статус 404, что указывает на то, что запрашиваемый ресурс не найден (специально было заменено название файла для теста).

**Подробный анализ запросов**

1. **Запросы на корневой путь и статические файлы**:
   * GET / — успешный запрос к корневой странице.
   * GET /css/styles.css — успешный запрос к стилям.
   * GET /css/droplist.css — успешный запрос к стилям.
   * GET /production1 — успешный запрос к странице продукта.
2. **Ошибки 404**:
   * GET /css/single-styles.css — возвращает 404. Это может указывать на то, что файл стилей не существует по указанному пути. Возможно, стоит проверить наличие этого файла на сервере.
   * GET /css/single-styles.css (второй раз) также возвращает 404, что подтверждает, что проблема сохраняется.
3. **POST-запросы**:
   * POST /submit — два успешных запроса с кодом 200, что указывает на успешное выполнение операций, вероятно, связанных с отправкой данных формы.

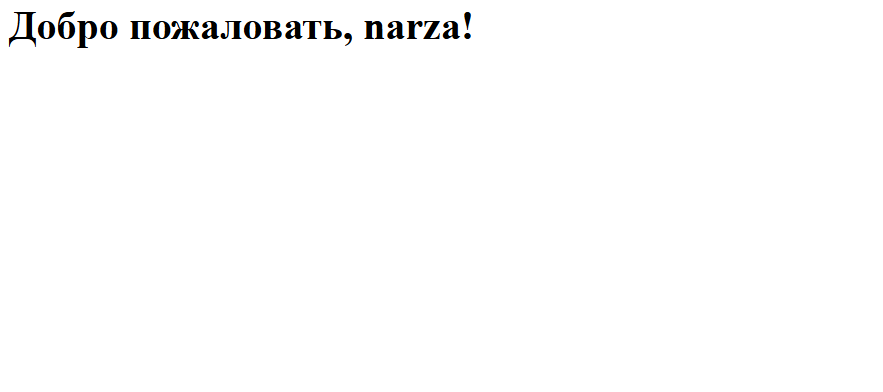
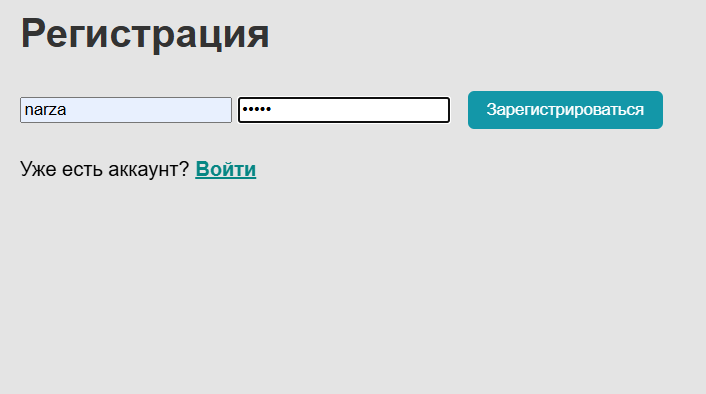
**Возможные проблемы**

1. **Отсутствие файлов**:
   * Ошибки 404 указывают на отсутствие файла single-styles.css. Необходимо проверить, существует ли этот файл на сервере и правильно ли указан путь к нему в HTML-коде (специально было заменено название файла для теста).
2. **Потенциальные повторные запросы**:
   * В логах видно, что некоторые файлы, такие как styles.css и button.css, запрашиваются несколько раз. Это может указывать на то, что браузер не кэширует эти файлы должным образом, или они не загружаются из кэша.
3. **Скорость отклика**:
   * Все запросы возвращают статус 200, но если пользователь заметил задержки, стоит рассмотреть возможность оптимизации серверных процессов или кэширования статических ресурсов.

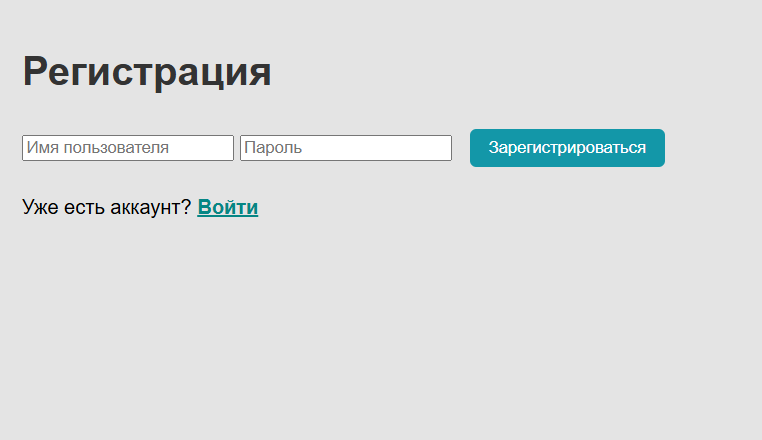
## Аутентификация пользователей

В качестве страницы, недоступной до полной аутентификации была взята страница профиля – /profile

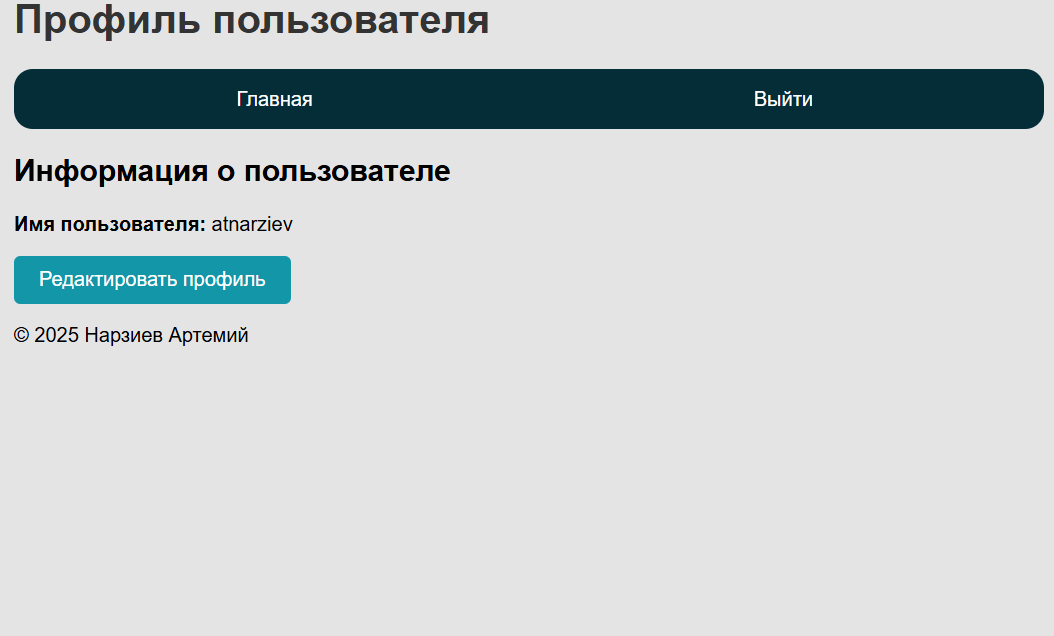
### Пример аутентификации:



### При переходе на страницу профиля – profile.ejs без аутентификации:



### При переходе на страницу профиля после утентификации:



### Часть кода отвечающая за контроль над сессиями:

Также защищена страница – profile.ejs до успешной авторизации мы на нее не попадем.

/\* . . . \*/

let sessions = {}; // Объект для хранения сессий

/\* . . . \*/

// Обработка регистрации

            if (req.url === '/register' && req.method === 'POST') {

                console.log('Получен POST-запрос на регистрацию'); // Отладочное сообщение

                let body = '';

                req.on('data', chunk => {

                    body += chunk.toString();

                });

                req.on('end', () => {

                    const { username, password } = querystring.parse(body);

                    // Проверка на существование пользователя

                    const existingUser  = users.find(u => u.username === username);

                    if (existingUser ) {

                        res.writeHead(400, { 'Content-Type': 'text/plain; charset=utf-8' });

                        return res.end('Пользователь с таким именем уже существует.');

                    }

                    // Хешируем пароль перед сохранением

                    bcrypt.hash(password, 10, (err, hashedPassword) => {

                        if (err) {

                            res.writeHead(500, { 'Content-Type': 'text/plain; charset=utf-8' });

                            return res.end('Ошибка хеширования пароля');

                        }

                        // Добавление нового пользователя

                        const newUser  = { username, password: hashedPassword };

                        users.push(newUser );

                        // Создание сессии

                        const sessionId = new Date().getTime(); // простой способ генерации ID сессии

                        sessions[sessionId] = newUser ;

                        // Установка cookie с идентификатором сессии

                        res.writeHead(302, {

                            Location: '/auth/profile',

                            'Set-Cookie': `sessionId=${sessionId}; HttpOnly`

                        });

                        res.end();

                    });

                });

                return;

            }

// Защищенный маршрут для отображения профиля

            if (req.url === '/auth/profile') {

                const cookies = parseCookies(req.headers.cookie);

                const sessionId = cookies.sessionId;

                if (sessionId && sessions[sessionId]) {

                    const user = sessions[sessionId];

                    res.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8' });

                    res.end(`<h1>Добро пожаловать, ${user.username}!</h1>`);

                } else {

                    res.writeHead(302, { Location: '/register' });

                    res.end();

                }

                return;

            }

### Вывод в консоль после аутентификации (регистрации):

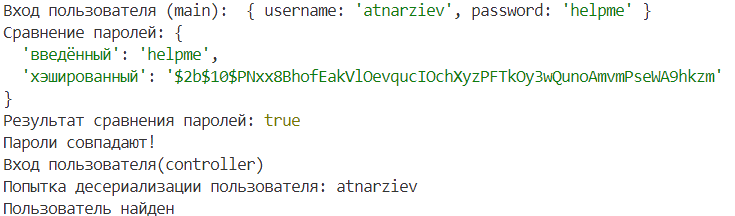
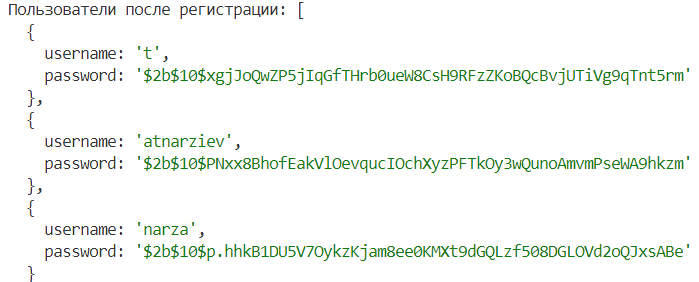
### 

### Вывод в консоль после аутентификации (входа):

### 

### Регистрация с одинаковым именем:

Вывод в консоль при попытке регистрации аккаунта с именем, уже имеющимся в списке зарегестрированных:

******

Вывод в HTML документе, автоматически открытом при переадресации при ошибке:

## 

## Упрощение маршрутизации c Express.js

Пункт тесно связан с предыдущим, по этой причине здесь включены некоторые подпункты, которые являются частью аутентификационного процесса.

### Основной файл сервера - server-express.js:

const express = require('express');

const path = require('path');

const fs = require('fs');

const LocalStrategy = require('passport-local').Strategy;

const morgan = require('morgan');

const session = require('express-session');

const cookieParser = require('cookie-parser');

const passport = require('passport');

const authRoutes = require('./routes/authRoutes');

const authController = require('./controllers/authController');

const { getIp } = require('./utils/network.js');

const users = require('./models/userModel');

const app = express();

const PORT = 4000; // Порт, на котором будет работать сервер'

// Middleware

app.use(express.urlencoded({ extended: true }));

app.use(express.json());

app.use(cookieParser());

app.use(morgan('combined', { stream: fs.createWriteStream(path.join(\_\_dirname, 'logs', 'access.log'), { flags: 'a' }) }));

// Passport middleware

app.use(session({

    secret: 'your\_secret\_key',

    resave: false,

    saveUninitialized: true }));

// Инициализация Passport

app.use(passport.initialize());

app.use(passport.session());

// Сериализация и десериализация пользователя

passport.serializeUser ((user, done) => {

    done(null, user.username); // Сохраняем только имя пользователя в сессии

});

passport.deserializeUser ((username, done) => {

    console.log('Попытка десериализации пользователя:', username);

    const user = users.find(u => u.username === username);

    if (user) {

        console.log('Пользователь найден');

        done(null, user); // Если пользователь найден, передаем его

    } else {

        done(new Error('Пользователь не найден')); // Если не найден, передаем ошибку

    }

});

// Подключаем маршруты

app.use('/', authRoutes);

// Главная страница

app.get('/', (req, res) => {

    const homeData = {

        title: 'Главная страница',

        description: 'Добро пожаловать на наш сайт, где вы найдете информацию о веб-технологиях и наши услуги.'

    };

    res.render('index', homeData);

});

// Страница "Для первокурсников"

app.get('/production1', (req, res) => {

    const production1Data = {

        title: 'Продукция для первокурсников',

        description: 'Мы предлагаем разнообразные товары для первокурсников.',

        products: [

            { name: 'Учебник по программированию', price: '500₽' },

            { name: 'Блокнот для записей', price: '150₽' },

            { name: 'Ручка', price: '50₽' }

        ]

    };

    res.render('production', production1Data);

});

// Страница "Для второкурсников"

app.get('/production2', (req, res) => {

    const production2Data = {

        title: 'Продукция для второкурсников',

        description: 'Мы предлагаем разнообразные товары для второкурсников.',

        products: [

            { name: 'Учебник по алгоритмам', price: '600₽' },

            { name: 'Курс по веб-разработке', price: '2000₽' },

            { name: 'Математика для программистов', price: '700₽' }

        ]

    };

    res.render('production', production2Data);

});

// Страница ввода

app.get('/input', (req, res) => {

    res.render('input');

});

// Страница таблицы

app.get('/table', (req, res) => {

    res.render('table');

});

// Страница с изображением

app.get('/image', (req, res) => {

    res.render('image');

});

// Страница входа

app.get('/login', (req, res) => {

    res.render('auth/auth', {

        title: 'Вход',

        action: '/login',

        buttonText: 'Войти',

        linkRegLogText: 'Нет аккаунта?',

        linkHrefRegLog: '/register',

        linkHrefRegLogLabel: 'Зарегистрироваться'

    });

});

// Страница регистрации

app.get('/register', (req, res) => {

    res.render('auth/auth', {

        title: 'Регистрация',

        action: '/register',

        buttonText: 'Зарегистрироваться',

        linkRegLogText: 'Уже есть аккаунт?',

        linkHrefRegLog: '/login',

        linkHrefRegLogLabel: 'Войти'

    });

});

// Статические файлы

app.use(express.static(path.join(\_\_dirname, 'public')));

// Шаблонизатор

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'ejs');

// Запуск сервера

getIp((ip) => {

    app.listen(PORT, ip, () => {

        console.log(`Сервер запущен на http://${ip}:${PORT}/`);

    });

});

### Файл сетевых функций – network.js:

const { exec } = require('child\_process');

const os = require('os');

function getIp(callback) {

    let command;

    const platform = os.platform();

    if (platform === 'darwin') {

        command = 'ipconfig getifaddr en0';

    } else if (platform === 'win32') {

        command = 'ipconfig | findstr /i "IPv4"';

    } else if (platform === 'linux') {

        command = 'hostname -I | awk \'{print \\$1}\'';

    } else {

        console.error('Unknown platform: ' + platform);

        return;

    }

    exec(command, (error, stdout) => {

        if (error) {

            console.error(`Error executing command: ${error}`);

            return;

        }

        // Используем регулярное выражение для извлечения IP-адреса

        let ipAddress;

        if (platform === 'win32') {

            const match = stdout.match(/(\d{1,3}\.){3}\d{1,3}/);

            ipAddress = match ? match[0] : null;

        } else {

            ipAddress = stdout.trim();

        }

        if (ipAddress) {

            console.log(`IP: ${ipAddress}`);

            callback(ipAddress);

        } else {

            console.error('Не удалось извлечь IP-адрес.');

        }

    });

}

module.exports = { getIp };

Так как функция добавлена не по заданию, далее приведено её обьяснение:

1. **Определение платформы**: Проверяет, какая операционная система используется (macOS, Windows или Linux) и задает соответствующую команду для получения IP-адреса.
2. **Выполнение команды**: Выполняет команду через exec и обрабатывает вывод.
3. **Извлечение IP-адреса**: Использует регулярное выражение для извлечения IP-адреса из вывода команды (для Windows) или просто обрезает вывод (для macOS и Linux).

### Файл для путей – authRoutes.js:

const express = require('express');

const router = express.Router();

const authController = require('../controllers/authController');

// Регистрация

router.post('/register', authController.registerUser);

// Вход

router.post('/login', authController.loginUser);

// Профиль

router.get('/profile', authController.profile);

// Выход

router.get('/logout', authController.logout);

module.exports = router;

### Файл моделей - userModel :

const users = []; // Хранилище пользователей

module.exports = users;

// Функция для добавления пользователя

exports.addUser  = (user) => {

    users.push(user);

};

// Функция для поиска пользователя по имени

exports.find\_user\_by\_username = (username) => {

    return users.find(user => user.username === username);

};

// Функция для получения всех пользователей (если потребуется)

exports.getAllUsers = () => {

    return users;

};

### Файл контроллеров – authController.js:

const bcrypt = require('bcrypt');

const passport = require('passport');

const { pool, addUserToDB, findUserByUsernameInDB, nicknameExists } = require('../config/db');

const NodeCache = require('node-cache');

const cache = new NodeCache({ stdTTL: 3600 }); // Кэш на 1 час

// Регистрация пользователя

exports.registerUser  = async (req, res) => {

    console.log('Регистрация пользователя:', req.body);

    const { username, password } = req.body;

    // Генерируем уникальный никнейм для нового пользователя

    const randomname = generateGuestNickname();

    const existingUser  = await findUserByUsernameInDB(username);

    if (existingUser ) {

        return res.status(400).send('Пользователь с таким именем уже существует.');

    }

    bcrypt.hash(password, 10, async (err, hashedPassword) => {

        console.log('Please help');

        if (err) {

            return res.status(500).send('Ошибка хеширования пароля');

        }

        const newUser  = { username,

                           nickname: randomname,

                           password: hashedPassword };

        await addUserToDB(newUser );

        console.log('Пользователи после регистрации:', users); // Логируем массив пользователей

        req.login(newUser , (err) => {

            if (err) return res.status(500).send('Ошибка входа');

            console.log('Переходим на страницу профиля');

            return res.redirect('/profile/profile');

        });

    });

};

// Функция для генерации уникального никнейма

const generateGuestNickname = () => {

    let baseNickname = 'guest';

    let number = 1000;

    // Проверяем существующие никнеймы

    //while (users.some(user => user.nickname === `${baseNickname}${number}`)) {

    while (nicknameExists(`${baseNickname}${number}`)) {

        number++;

    }

    return `${baseNickname}${number}`;

};

// Проверка уникальности никнейма

exports.checkNickname = async (req, res) => {

    const { nickname, currentNickname } = req.body;

    // Если новый никнейм совпадает с текущим, возвращаем, что он уникален

    if (nickname === currentNickname) {

        return res.json({ isUnique: true });

    }

    // Проверяем кэш на наличие результата

    const cacheKey = `nicknameExists:${nickname}`;

    const cachedExists = cache.get(cacheKey);

    if (cachedExists !== undefined) {

        return res.json({ isUnique: !cachedExists });

    }

    try {

        const exists = await nicknameExists(nickname); // Проверяем уникальность

        cache.set(cacheKey, exists); // Кэшируем результат

        res.json({ isUnique: !exists });

    } catch (error) {

        res.status(500).json({ error: 'Ошибка при проверке никнейма.' });

    }

};

// Вход пользователя

exports.loginUser  = (req, res, next) => {

    passport.authenticate('local', async (err, user, info) => { // Обработка результата newLocalStrategy

        console.log('Вход пользователя(controller)');

        if (err) {

            return res.status(500).json({ error: 'Ошибка аутентификации' });

        }

        if (!user) {

            return res.status(401).json({ error: 'Неверное имя пользователя или пароль' });

        }

        req.logIn(user, (err) => {

            if (err) {

                return res.status(500).json({ error: 'Ошибка входа' });

            }

            return res.status(200).json({ message: 'Успешный вход!' }); // Отправка успешного сообщения

        });

    })(req, res, next);

};

// Профиль пользователя

exports.profile = (req, res) => {

    if (req.isAuthenticated()) {

        return res.render('auth/profile', { user: req.user });

    } else {

        return res.redirect('/profile/login');

    }

};

// Выход пользователя

exports.logout = (req, res) => {

    req.logout((err) => {

        if (err) {

            return res.status(500).send('Ошибка при выходе из системы');

        }

        req.session.destroy((err) => {

            if (err) {

                return res.status(500).send('Ошибка при уничтожении сессии');

            }

            res.redirect('/profile/login');

        });

    });

};

// Обновление профиля

exports.updateProfile = async (req, res) => {

    const userId = req.user.id; // Используем ID пользователя из сессии

    const { nickname, email, birthDate } = req.body;

    try {

        const cacheKey = `userProfile:${userId}`;

        const cachedUser  = cache.get(cacheKey);

        let currentUser;

        if (cachedUser ) {

            currentUser = cachedUser ;

        } else {

            const result = await pool.query('SELECT nickname FROM users WHERE id = \$1', [userId]);

            console.log(userId);

            if (result.rows.length === 0) {

                return res.status(404).json({ error: 'Пользователь не найден.' });

            }

            currentUser  = result.rows[0];

            cache.set(cacheKey, currentUser ); // Кэшируем данные пользователя

        }

        const currentNickname = currentUser.nickname;

        // Если никнейм не изменился, просто обновляем другие поля

        if (nickname === currentNickname) {

            await pool.query(

                'UPDATE users SET email = \$1, birthDate = \$2 WHERE id = \$3',

                [email, new Date(birthDate), userId]

            );

            return res.json({ success: 'Ваш профиль успешно обновлен!' });

        }

        // Если никнейм изменился проверяем уникальность

        const exists = await nicknameExists(nickname);

        if (exists) {

            return res.status(400).json({ error: 'Никнейм уже существует.' });

        }

        // Обновляем никнейм и другие поля

        await pool.query(

            'UPDATE users SET nickname = \$1, email = \$2, birthDate = \$3 WHERE id = \$4',

            [nickname, email, new Date(birthDate), userId]

        );

        // Обновляем кэш с новыми данными

        cache.set(cacheKey, { ...currentUser , nickname });

        // Отправляем сообщение об успехе

        console.log('Отправляемые данные:', { nickname, email, birthDate });

        return res.json({ success: 'Ваш профиль успешно обновлен!' });

    } catch (err) {

        console.error(err);

        // Сообщение об ошибке

        return res.status(500).json({ error: 'Произошла ошибка при обновлении профиля. Попробуйте еще раз.' });

    }

};

// Редактирование профиля

exports.editProfile = (req, res) => {

    if (req.isAuthenticated()) {

        return res.render('auth/edit', {

            user: req.user,

            successMessage: '', // Успешные сообщения теперь будут передаваться через JSON

            errorMessage: '' // Ошибки тоже

        });

    } else {

        return res.redirect('profile/login');

    }

};

### CRUD-операции:

#### До изменений:

### 

#### Редактирование профиля:

### 

#### После изменений:

### 

### 

Далее была добавлена база данных с таблицами users (таблица пользователей) и movies (таблица для хранения фильмов для страницы).

### CRUD с базой данных

#### Файл для работы с базой данных – db.js:

Часть, отвечающая за добавление пользователя в базу данных:

. . .

// Функция для добавления пользователя в базу данных

const addUserToDB = async (user) => {

    const { username, nickname, password } = user;

    // Проверяем, существует ли никнейм

    if (await nicknameExists(nickname)) {

        throw new Error('Никнейм уже существует. Пожалуйста, выберите другой.');

    }

    const query = 'INSERT INTO users (username, nickname, password) VALUES (\$1, \$2, \$3) RETURNING \*';

    const values = [user.username, user.nickname, user.password];

    try {

        const result = await pool.query(query, values);

        // Логируем результат запроса

        console.log('Результат вставки:', result.rows[0]);

        console.log('Пользователь успешно добавлен в базу данных.');

        // Возвращаем созданного пользователя с его id

        return result.rows[0];

    } catch (error) {

        console.error('Ошибка при добавлении пользователя в базу данных:', error);

        throw error; // Перебрасываем ошибку для дальнейшей обработки

    }

};

const findUserById = async (id) => {

    const result = await pool.query('SELECT \* FROM users WHERE id = \$1', [id]);

    return result.rows[0]; // Возвращаем первого найденного пользователя или undefined

};

. . .

А также часть, отвечающая за создание и обновление иформации о фильме:

. . .

const updateMovie = async (id, updatedData) => {

    const { title, description, image } = updatedData;

    const query = `

        UPDATE movies

        SET title = \$1, description = \$2, image = \$3

        WHERE id = \$4

    `;

    try {

        const result = await pool.query(query, [title, description, image, id]);

        return result.rowCount > 0; // Возвращаем true, если обновление прошло успешно

    } catch (error) {

        console.error('Ошибка обновления фильма:', error);

        throw error; // Пробрасываем ошибку дальше для обработки в контроллере

    }

};

. . .

#### Файл контроллерова – movieController.js:

. . .

const editList = async (req, res) => {

    try {

        const movies = await getAllMovies(); // Функция для получения всех фильмов из базы данных

        res.render('movies/editList', { movies }); // Отправляем список фильмов в шаблон

    } catch (error) {

        console.error(error);

        res.status(500).send('Ошибка сервера'); // Обработка ошибок сервера

    }

};

const editData = async (req, res) => {

    // Преобразуем ID из параметров URL в целое число

    const movieId = parseInt(req.params.id, 10);

    try {

        const movie = await getMovieById(movieId); // Получаем фильм по ID

        if (!movie) {

            return res.status(404).send('Фильм не найден'); // Обработка случая, когда фильм не найден

        }

        res.render('movies/editData', { movie }); // Отправляем данные о фильме в шаблон редактирования

    } catch (error) {

        console.error(error);

        res.status(500).send('Ошибка сервера'); // Обработка ошибок сервера

    }

};

. . .

. . .

const updateData = async (req, res) => {

    const movieId = parseInt(req.params.id, 10); // Преобразование id в число

    if (isNaN(movieId)) {

        return res.status(400).send('Неверный идентификатор фильма');

    }

    const updatedData = {

        title: req.body.title,

        description: req.body.description,

        image: req.body.image,

    };

    try {

        const updatedMovie = await updateMovie(movieId, updatedData);

        if (updatedMovie) {

            res.redirect(`/movies/${movieId}`); // Перенаправление на страницу фильма после успешного обновления

        } else {

            res.status(404).send('Фильм не найден');

        }

    } catch (error) {

        console.error(error);

        res.status(500).send('Ошибка обновления фильма');

    }

}

// Экспортируем все функции контроллера в одном объекте

module.exports = {

    getMovies,

    editList,

    editData,

    updateData,

    getMovieInfo,

};

#### Файл маршрутов – movieController.js:

const express = require('express');

const movieController = require('../controllers/movieController');

const { isAdmin } = require('../middleware/adminMiddleware');

const router = express.Router();

// Маршрут для страницы списка фильмов

router.get('/movies', isAdmin, movieController.getMovies);

// Страница редактирования листа

router.get('/movies/edit', isAdmin, movieController.editList);

// Страница информации о фильме

router.get('/movies/:id',  isAdmin, movieController.getMovieInfo);

// Страница редактирования фильма

router.get('/movies/:id/edit', isAdmin, movieController.editData);

// Обработчик для сохранения изменений фильма

router.post('/movies/:id/edit', movieController.updateData);

module.exports = router;

#### Файл шаблона страницы фильмов – movieList.ejs:

Была добавлена кнопка, доступная для администраторов, нажимая на которую мы переносимся на страницу редактирвоания списка фильмов:

<!-- Кнопка редактирования, отображается только для администраторов -->

    <% if (isAdmin) { %>

        <button onclick="window.location.href='/movies/edit'" class="edit-button">Редактировать список фильмов</button>

    <% } %>

    <!--  -->

#### Файл – шаблон страницы реадктироания списка фильмов – editList.ejs:

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Редактирование списка фильмов</title>

</head>

<body>

    <h1>Редактирование списка фильмов</h1>

    <table>

        <thead>

            <tr>

                <th>Название</th>

                <th>Действия</th>

            </tr>

        </thead>

        <tbody>

            <% movies.forEach(movie => { %>

                <tr>

                    <td><%= movie.title %></td>

                    <td>

                        <a href="/movies/<%= movie.id %>/edit">Редактировать</a>

                    </td>

                </tr>

            <% }) %>

        </tbody>

    </table>

    <a href="/movies">Назад к списку фильмов</a>

</body>

</html>

#### Файл – шаблон страницы редактирования конкретного фильма - editData.ejs:

<!DOCTYPE html>

<html lang="ru">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Редактировать фильм</title>

</head>

<body>

    <h1>Редактировать фильм: <%= movie.title %></h1>

    <form action="/movies/<%= movie.id %>/edit" method="POST">

        <label for="title">Название:</label>

        <input type="text" id="title" name="title" value="<%= movie.title %>" required>

        <label for="description">Описание:</label>

        <textarea id="description" name="description" required><%= movie.description %></textarea>

        <label for="image">URL изображения:</label>

        <input type="text" id="image" name="image" value="<%= movie.image %>" required>

        <button type="submit">Сохранить изменения</button>

    </form>

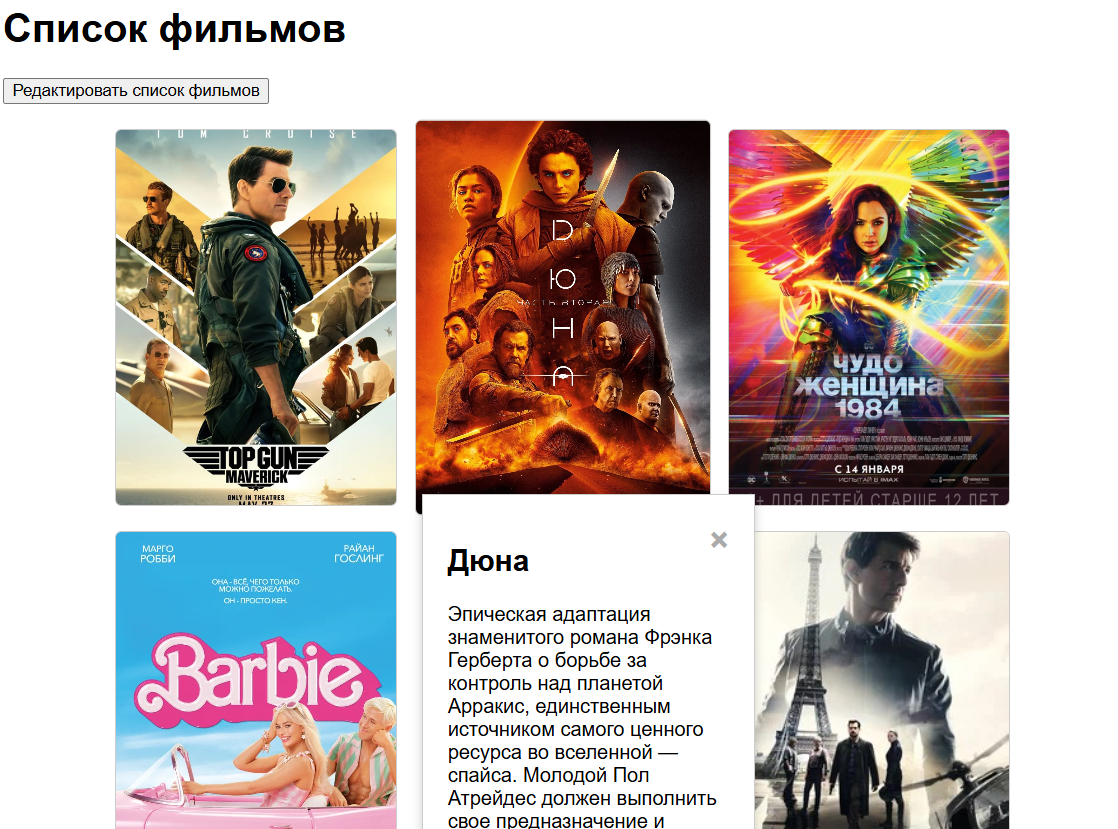
    <a href="/movies">Назад к списку фильмов</a>

</body>

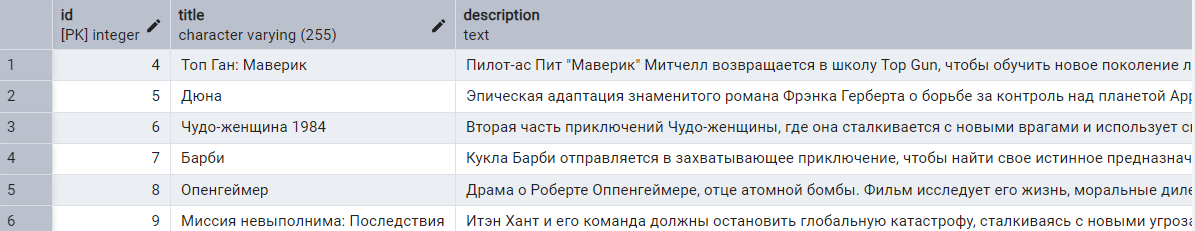
</html>

#### Пример работы с CRUD для фильмов

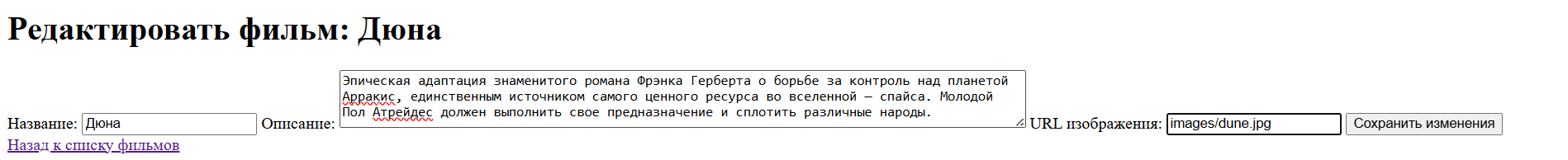
Создана страница фильмов в которой только администраторам доступна кнопка редактирования как списка фильмов, так и информации о конкретном фильме. Работа с CRUD API показана на примере фильма “Дюна”.

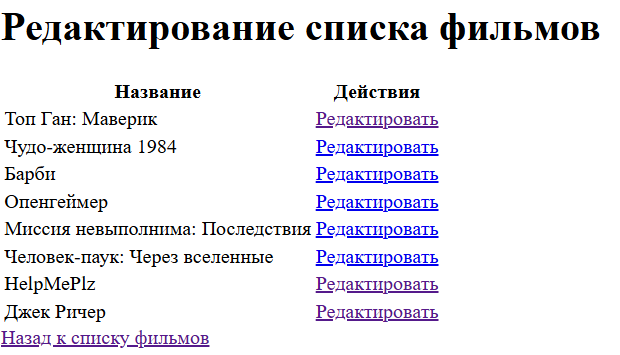


1. **В базе данных имеем:**



1. **Изменим название и описание:**

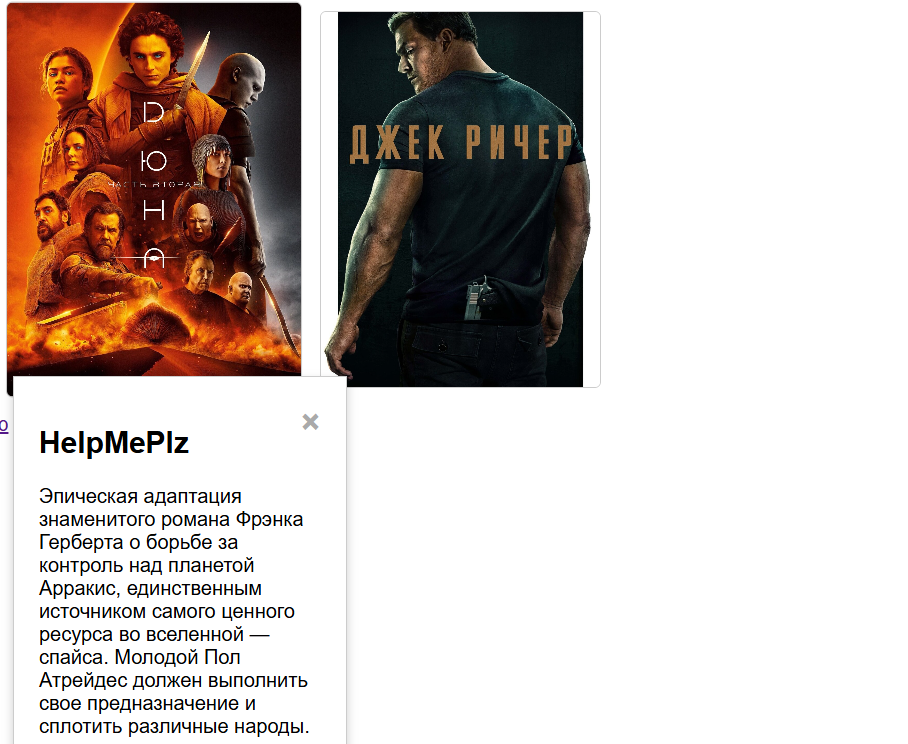




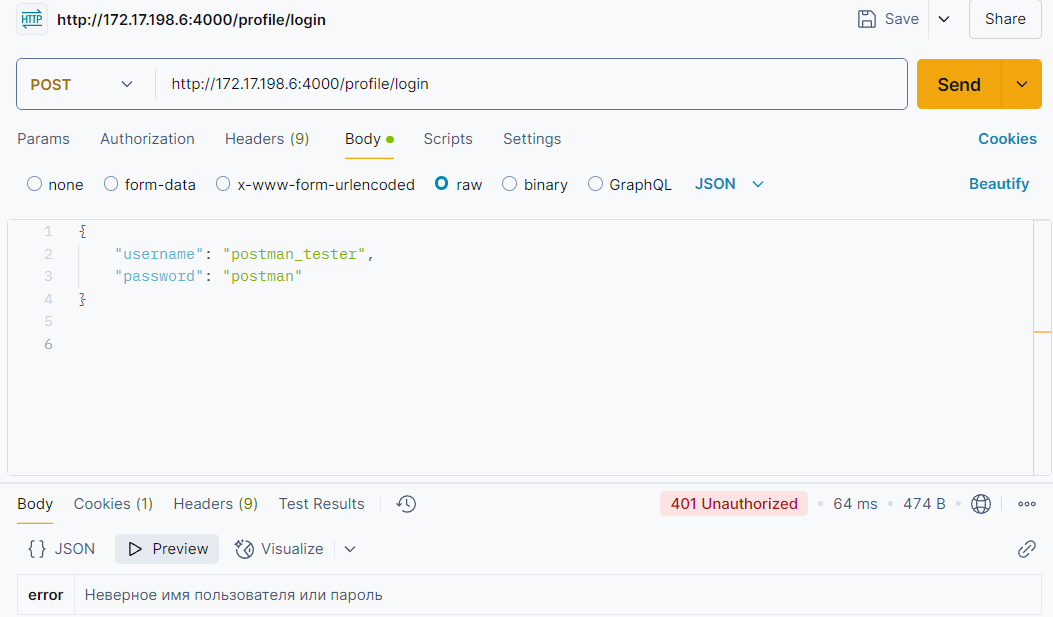
1. **В базе данных имеем:**



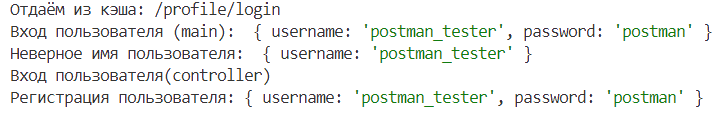
1. **В приложении имеем:**



### Тестирование с помощью Postman:

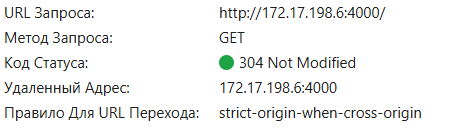


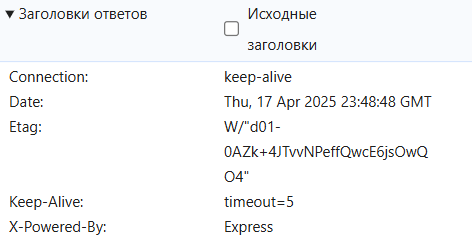
Вывод в консоли:



Выдает ошибку так как мы не зарегестрированы а запрос направлен на url регистрации.

## Кэширование ресурсов







### Etag

ETag (Entity Tag) — это HTTP-заголовок, который используется для идентификации версии ресурса на сервере. Он позволяет кэшировать ресурсы и оптимизировать взаимодействие между клиентом и сервером.

1. **Идентификация версии**: ETag представляет собой уникальный идентификатор, связанный с конкретной версией ресурса. Когда ресурс изменяется (например, файл, изображение или страница), его ETag также изменяется.
2. **Кэширование**: Когда клиент (например, браузер) запрашивает ресурс, сервер может отправить ETag в заголовке ответа. Клиент может сохранить этот ETag и использовать его в последующих запросах для проверки, изменился ли ресурс.
3. **Оптимизация запросов**: При следующем запросе клиент может отправить заголовок If-None-Match с сохраненным ETag. Если ресурс не изменился, сервер может ответить с кодом состояния 304 Not Modified, что позволяет избежать передачи данных, которые не изменились. Это экономит пропускную способность и ускоряет загрузку.
4. **Первый запрос**:
   * Клиент делает запрос на ресурс.
   * Сервер отвечает с ресурсом и заголовком ETag: W/"d01-0AZk+4JTvvNPeffQwcE6jsOwQO4".

HTTP/1.1 200 OK

ETag: W/"d01-0AZk+4JTvvNPeffQwcE6jsOwQO4"

Content-Type: application/json

Content-Length: 123

1. **Следующий запрос**:
   * Клиент делает запрос на тот же ресурс, добавляя заголовок If-None-Match с предыдущим ETag.

GET /resource HTTP/1.1

If-None-Match: W/"d01-0AZk+4JTvvNPeffQwcE6jsOwQO4"

1. **Ответ сервера**:
   * Если ресурс не изменился, сервер отвечает с кодом 304 Not Modified.

HTTP/1.1 304 Not Modified

### Cache-control

* **Cache-Control: max-age=0**

Мы видим **Cache-Control**: **max-age=0**, это означает, что кэшированный ресурс считается устаревшим сразу после его получения. Это может привести к некоторой путанице, так как кэш все еще может использоваться, но браузер должен проверять сервер для получения актуальной версии ресурса.

Далее приведено несколько причин, почему мы видим кэширование, но с max-age=0:

1. **Настройки кэширования**: Заголовок max-age=0 говорит браузеру, что он должен проверять сервер на наличие обновлений, прежде чем использовать кэшированные данные. Это полезно, если вы хотите, чтобы браузер всегда имел возможность получать свежие данные, но при этом не отключаете кэширование полностью.
2. **Поведение браузера**: Браузеры могут интерпретировать max-age=0 как указание на то, что кэшированные данные могут быть использованы, но они должны быть проверены на сервере. Это означает, что браузер может использовать кэшированные данные, но будет отправлять запрос на сервер, чтобы убедиться, что данные актуальны.

### If-none-match

* **if-none-match: W/"d01-**

**If-None-Match**: Этот заголовок используется для условного запроса к серверу. Он позволяет клиенту сообщить серверу, что он имеет кэшированную версию ресурса с указанным ETag (в данном случае W/"d01-"). Если ресурс на сервере не изменился, сервер может ответить статусом 304 Not Modified, что указывает на то, что клиент может использовать свою кэшированную версию.

# Вывод

Разработка программы завершена на том основании, что:

1. Полученные результаты совпали с ожидаемыми
2. Считаю набор тестов полным