|  |  |
| --- | --- |
| 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 №4  
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

1. Разработать программу клиент, работающий на сокетах на выбранном Вами языке программирования. Данный клиент должен хранить лог файл соединений. В лог файле хранится время подключения и адрес сервера, время отправки сообщения и само сообщение, время получения сообщения и само сообщение. При подключении к серверу клиент через определенный промежуток времени (задается разработчиком клиента) отправляет сообщение серверу: ФИО и группа студента, а также ждет от сервера. Данные о сервере необходимо вводить через файл конфигурации. В отчете привести код клиента и один тестовый лог файл.
2. Разработать программу сервер, которая ожидает строку от клиента. Через промежуток времени (моделирование работы сервера) он отправляет в ответ эту же строку, только зеркально отраженную и добавляет к сообщению ФИО студента, написавшего сервер. (пример ответа: Чиволвап Йирю Вотит. Сервер написан Титовым Ю.П. М3О-1ХХБВ-ХХ).

Сервер должен эмулировать работу путем временной задержки. Отключение клиентов от сервера совершать через определенное время работы клиента. Сервер и клиент должны быть написаны на различных языках программирования.

Сервер ведёт лог-файл, в котором содержится:

* Время запуска сервера,
* Время подсоединения клиента,
* Адрес клиента,
* Время получения сообщения и само сообщение,
* Время отправки сообщения и само сообщение, Время отключения клиента от сервера.

Лог-файл не очищать.

1. Доработать программу сервер для выполнения асинхронных запросов от клиентов. Чётные варианты реализуют многопоточный асинхронный сервер, нечетные – многопроцессный. Номер варианта определяется номером по списку.

При доработке сервера необходимо создать новый проект, т.е. в результате должно получиться две программы сервера.

1. Запустить 5 клиентов в сети кафедры и 1 сервер (5 студентов). В отчете указать настройки для подключения и лог файлы клиентов и серверов. Проводить тестирование одновременного подключения, т.е. второй, третий и т.д. пользователь подключается к серверу пока первый пользователь еще не получил ответ. Желательно провести тестирование как на однопоточном, так и на асинхронном сервере, описать отличия.

# 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>

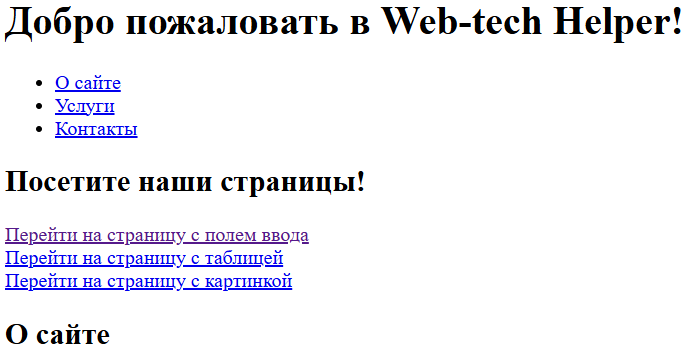
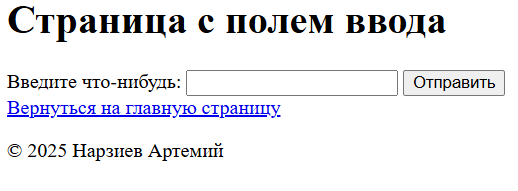
    </table>

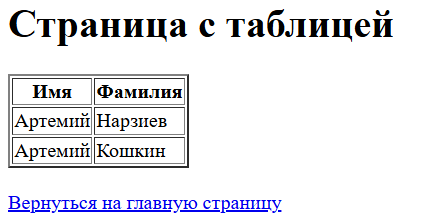
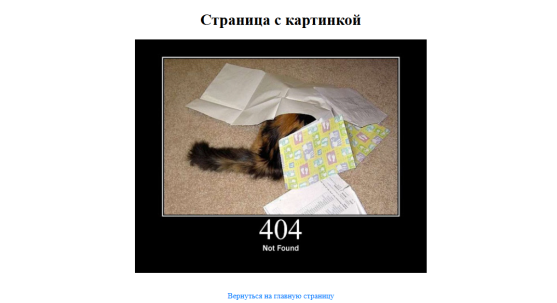
    <a href="/">Вернуться на главную страницу</a>

</body>

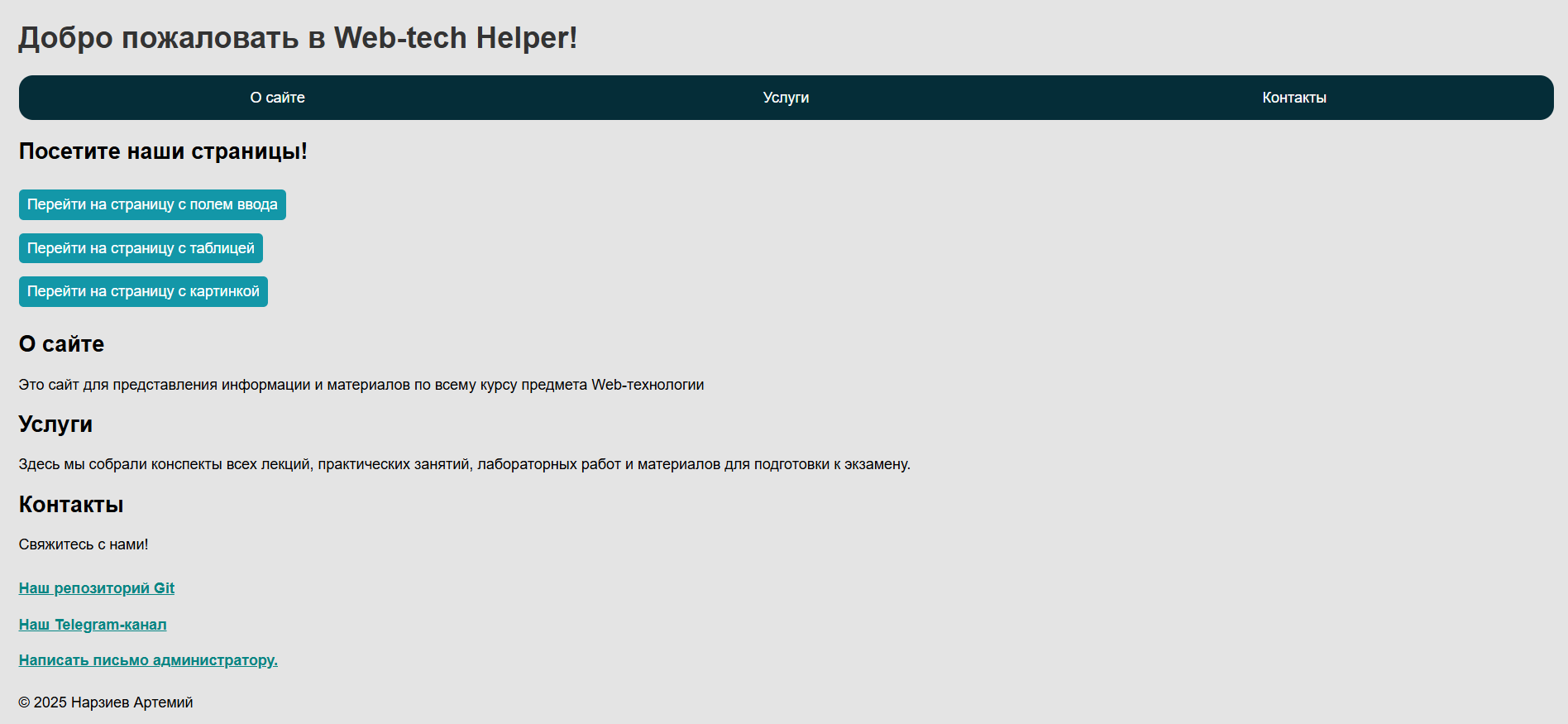
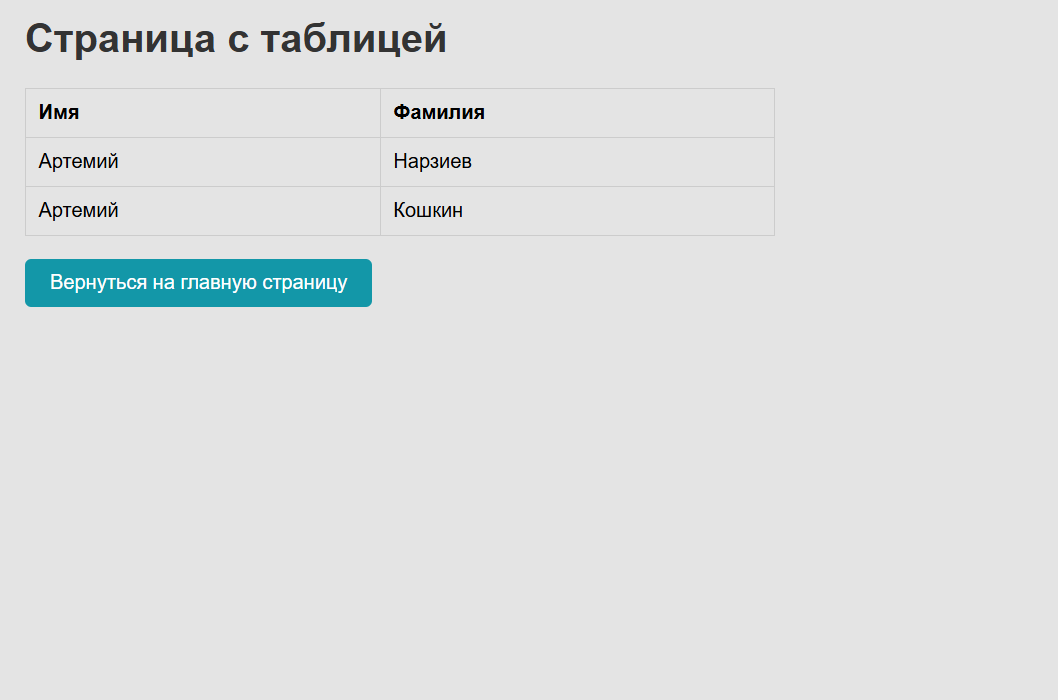
</html>

### 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 \*/

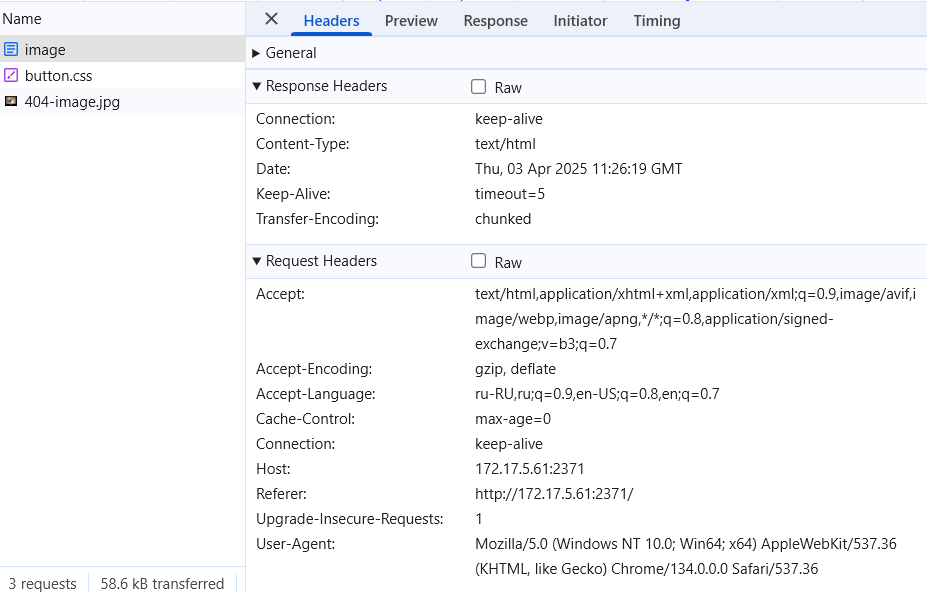
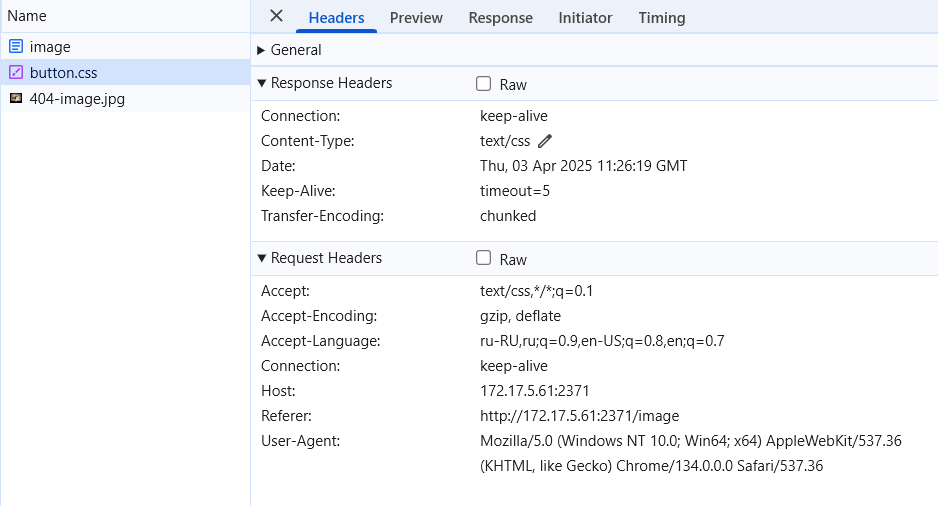
    transition: background-color 0.3s; /\* Плавный переход при наведении \*/

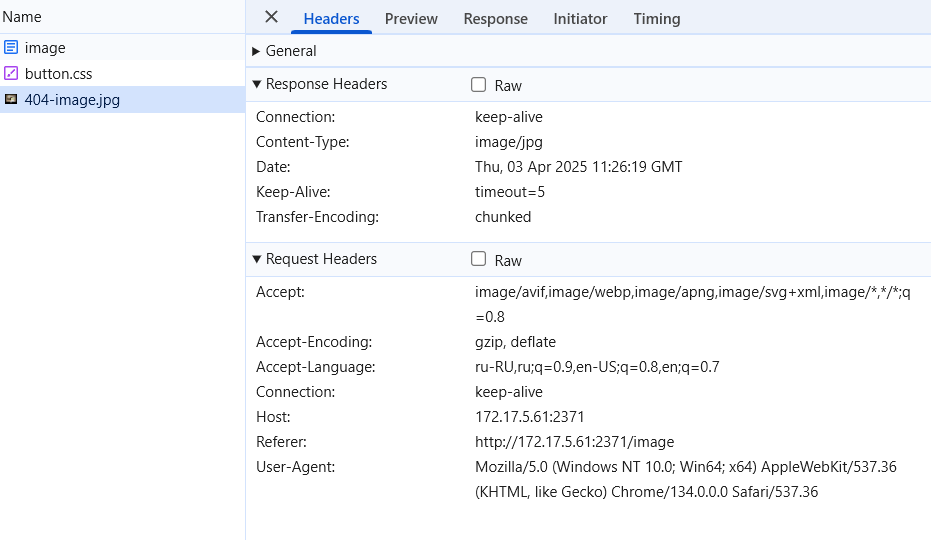
}

a.button:hover {

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

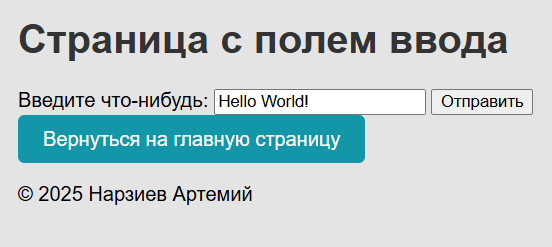
* 1. *Запросы и ответы (Network):*

********

**

## Обработка POST запросов

### Пример ввода

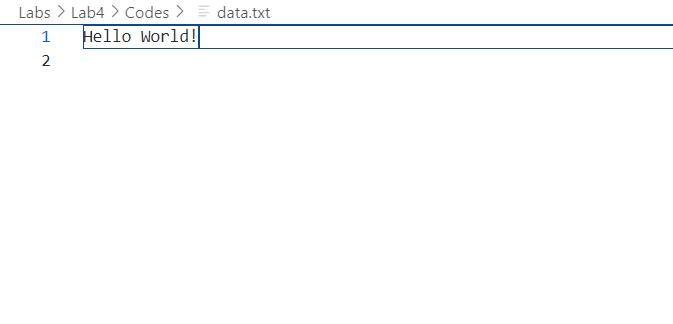


### Результат ввода

*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₽' }

                    ]

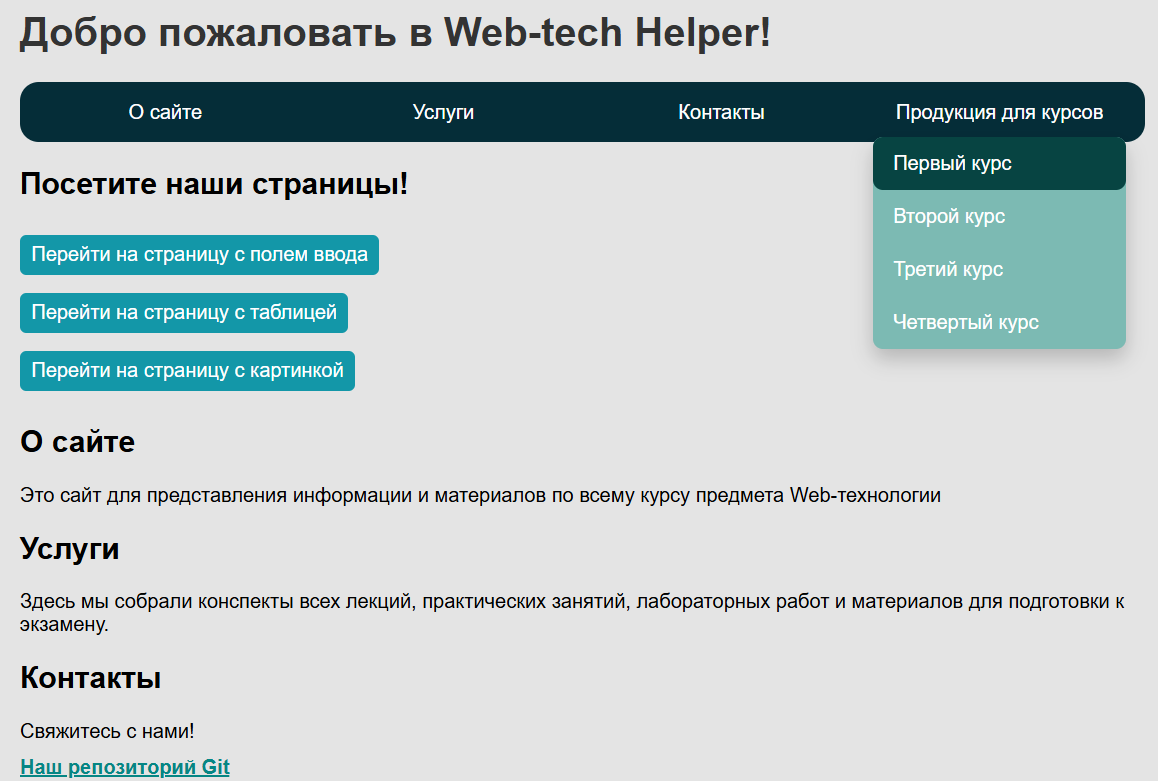
                };

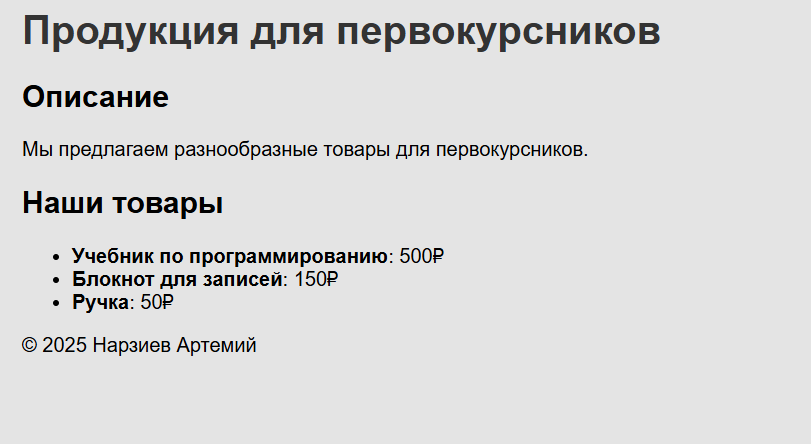
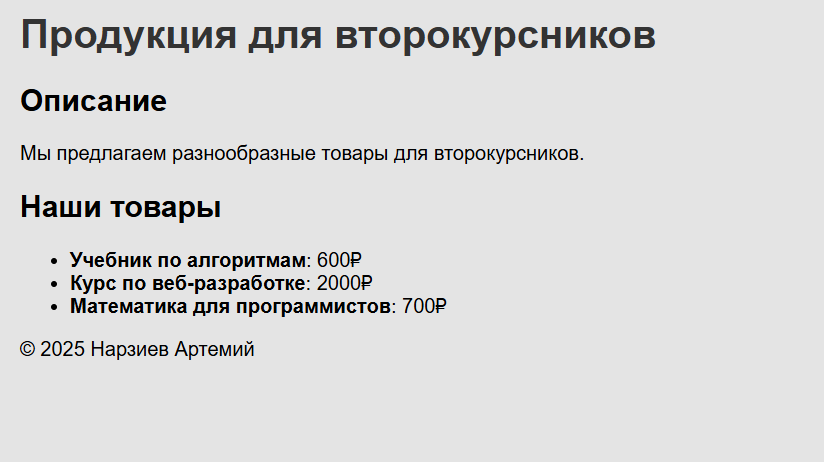
                renderTemplate('production', production2Data, res); // Используем шаблон index.ejs с данными для страницы "О нас"

                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.

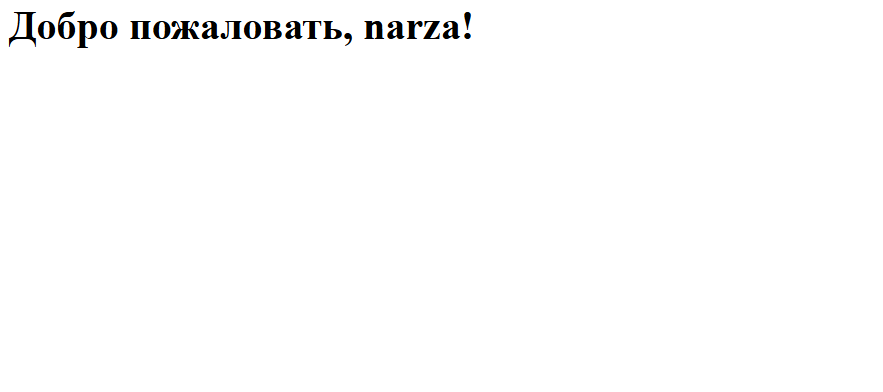
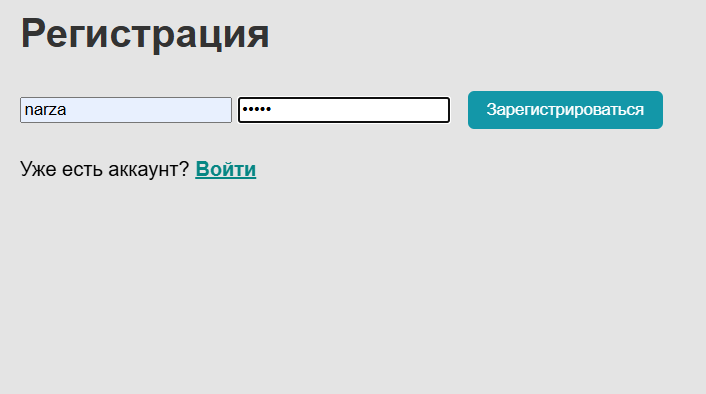
**Возможные проблемы**

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.

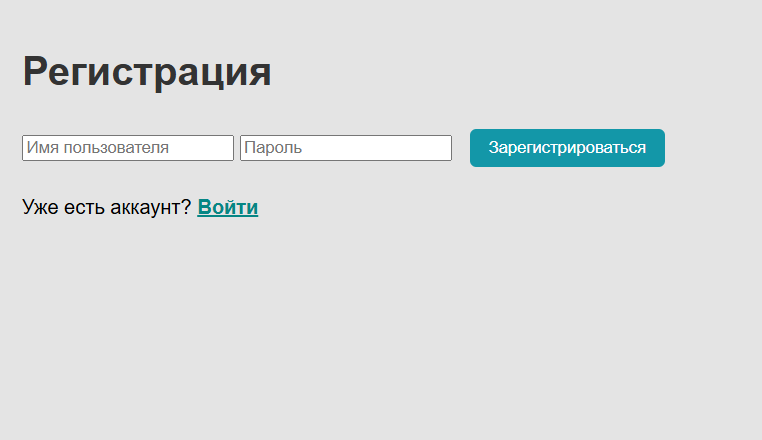
## Аутентификация пользователей

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

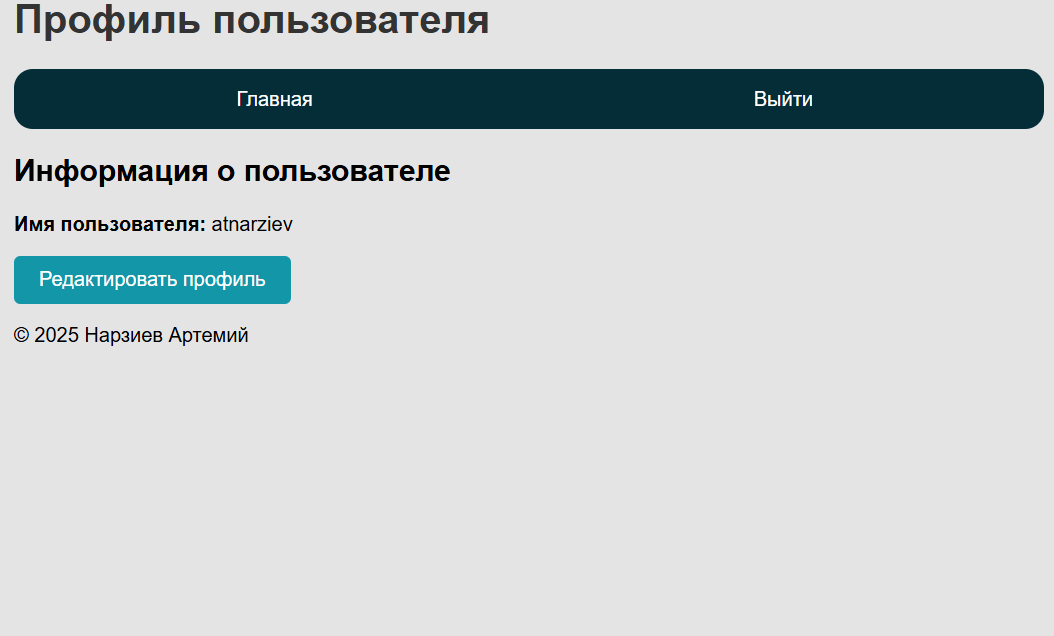
### Пример аутентификации:



### 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):

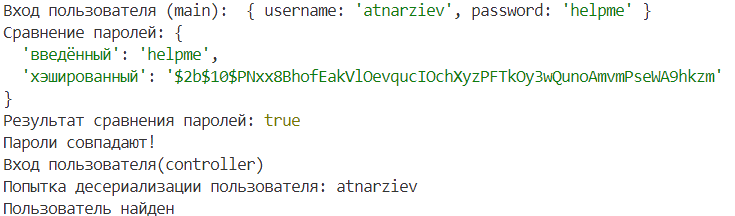
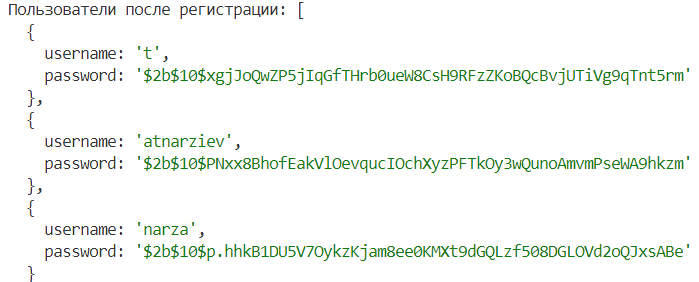
### 

### 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 users = require('../models/userModel'); // Модель пользователя, если используем базу данных

const bcrypt = require('bcrypt');

const passport = require('passport');

// Регистрация пользователя

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

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

    const { username, password } = req.body;

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

    if (existingUser ) {

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

    }

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

        console.log('Please help');

        if (err) {

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

        }

        const newUser  = { username, password: hashedPassword };

        users.push(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');

        });

    });

    console.log(users); // Выводит массив всех пользователей в консоль

};

// Вход пользователя

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

    console.log('heeeelp');

    passport.authenticate('local', (err, user, info) => {

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

        if (!user) return res.status(401).send('Неверное имя пользователя или пароль');

        req.logIn(user, (err) => {

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

            return res.redirect('/profile'); // Перенаправление на профиль после успешного входа

        });

    })(req, res, next);

};

// Профиль пользователя

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

    if (req.isAuthenticated()) {

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

    } else {

        return res.redirect('/login');

    }

};

// Выход пользователя

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

    req.logout();

    res.redirect('/login');

};

### CRUD-операции:

#### До изменений:

### 

#### Редактирование профиля:

### 

#### После изменений:

### 

### 

## Кэширование ресурсов

# Вывод

Разработка программы завершена на том основании, что:

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