

Министерство науки и высшего образования РФ  
Федеральное государственное автономное  
образовательное учреждение высшего образования  
**«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Институт космических и информационных технологий

---

институт

Программная инженерия

---

кафедра

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ**  
Обеспечение безопасности web-приложений

---

тема

Преподаватель

подпись, дата

Р. С. Шиманович

инициалы, фамилия

Студент КИ23-16/16, 032320521

номер группы, зачётной книжки

подпись, дата

А. С. Лысаковский

инициалы, фамилия

Красноярск 2025

## СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ .....	4
1.1 Цель работы .....	4
1.2 Задачи .....	4
1.3 Задание .....	4
2 ОСНОВНАЯ ЧАСТЬ.....	6
2.1 Описание программы .....	6
2.1.1 Назначение .....	6
2.1.2 Используемые технологии .....	6
2.1.3 Архитектура приложения .....	6
2.1.4 Структура проекта .....	7
2.2 Аутентификация .....	7
2.2.1 Регистрация пользователя .....	7
2.2.2 Вход в систему .....	8
2.2.3 Выход из системы .....	8
2.3 Управление задачами.....	8
2.3.1 Создание задачи .....	8
2.3.2 Просмотр задач .....	8
2.3.3 Редактирование задачи .....	8
2.3.4 Удаление задачи .....	9
2.4 Административные функции .....	9
2.4.1 Просмотр всех задач .....	9
2.4.2 Поиск задач .....	9
2.5 Описание разработанных мер по защите информации .....	9
2.5.1 Аутентификация .....	9
2.5.2 Защита от SQL Injection.....	10
2.5.3 Валидация .....	10
2.5.4 Ролевая модель доступа.....	10
2.5.5 Защита от кражи Cookie .....	11
2.6 Потоки данных.....	11
2.6.1 Регистрация.....	11
2.6.2 Вход .....	11

2.7 Контрольные примеры .....	12
2.7.1 Регистрация.....	12
2.7.2 Вход в систему .....	14
2.7.3 Выход из системы.....	15
2.7.4 Получение информации о текущем пользователе .....	15
2.7.5 Доступ к защищённому маршруту (обычный пользователь) .....	17
2.7.6 Попытка доступа без аутентификации .....	17
2.7.7 Доступ к административному маршруту (обычный пользователь) ....	18
2.7.8 Доступ к административному маршруту (администратор) .....	18
2.8 Листинг программы.....	20
3 ЗАКЛЮЧЕНИЕ .....	21
ПРИЛОЖЕНИЕ А .....	22
ПРИЛОЖЕНИЕ Б.....	36

# 1 ВВЕДЕНИЕ

## 1.1 Цель работы

Ознакомиться с базовыми механизмами обеспечения информационной безопасности в web-приложениях.

## 1.2 Задачи

В рамках данной практической работы необходимо выполнить следующие задачи:

- 1 Ознакомиться с основами механизмами аутентификации в web-приложениях;
- 2 Ознакомиться с базовыми моделями доступа в задачах защиты от несанкционированного доступа;
- 3 изучить криптографический протокол Диффи-Хеллмана и особенности его применения в задачах разработки механизмов аутентификации;
- 4 Ознакомиться с областью применения хеш-функций в задачах обеспечения информационной безопасности web-приложений;
- 5 Получить навыки разработки web-приложений с учетом аспектов их информационной безопасности.

## 1.3 Задание

Разработайте безопасное web-приложение с любой бизнес-логикой (функциональностью), нативно реализующее механизмы аутентификации, авторизации и управления доступом (посредством ролевой модели управления доступом) согласно схеме взаимодействия клиент-сервер, представленной на рис. 4.7. Безопасность приложения заключается в защитных мерах, реализованных непосредственно в приложении, способных нивелировать угрозы со стороны злоумышленника.

Особенности и требования к web-приложению следующие:

- приложение взаимодействует с пользователем посредством браузера и открытого канала связи, основанному на протоколе http;
- на стороне web-сервера допускается использование любых языков программирования. На стороне клиента допускается применение скриптов, в том числе фреймворков;
- не допускается как на стороне сервера, так и на стороне клиента использование готовых фреймворков, библиотек и/или алгоритмов, непосредственно реализующих элементы задания практической работы (за исключением библиотек с криптографическими функциями);
- web-приложение может функционировать под управлением любой операционной системы, а также может использовать любую систему управления базами данных;

- потенциал нарушителя позволяет ему свободно слушать открытый канал связи между клиентом и сервером приложения, а также осуществлять взаимодействие с сервером (отправлять ему любые запросы). Кроме того, потенциал нарушителя позволяет ему украсть cookies пользователя (не важно каким именно способом);

## **2 ОСНОВНАЯ ЧАСТЬ**

### **2.1 Описание программы**

#### **2.1.1 Назначение**

Было разработано веб-приложение «Система управления задачами». Приложение реализует полный цикл управления задачами (TODO-лист) с механизмами аутентификации, авторизации и управления доступом на основе ролевой модели.

#### **2.1.2 Используемые технологии**

Серверная часть:

- Node.js (версия 14+) – среда выполнения JavaScript;
- Express (версия 4.18.2) – минималистичный веб-фреймворк;
- SQLite (sql.js версия 1.10.3) – встраиваемая база данных;
- bcrypt (версия 5.1.1) – библиотека для хеширования паролей;
- cookie-parser (версия 1.4.6) – парсинг HTTP cookies.

Клиентская часть:

- HTML5 – разметка страниц;
- CSS3 – стилизация интерфейса;
- Vanilla JavaScript – клиентская логика без фреймворков.

Криптографические библиотеки:

- crypto (встроенный модуль Node.js) - для HMAC подписи;
- bcrypt - для хеширования паролей.

#### **2.1.3 Архитектура приложения**

Архитектура продемонстрирована на рисунке 1.

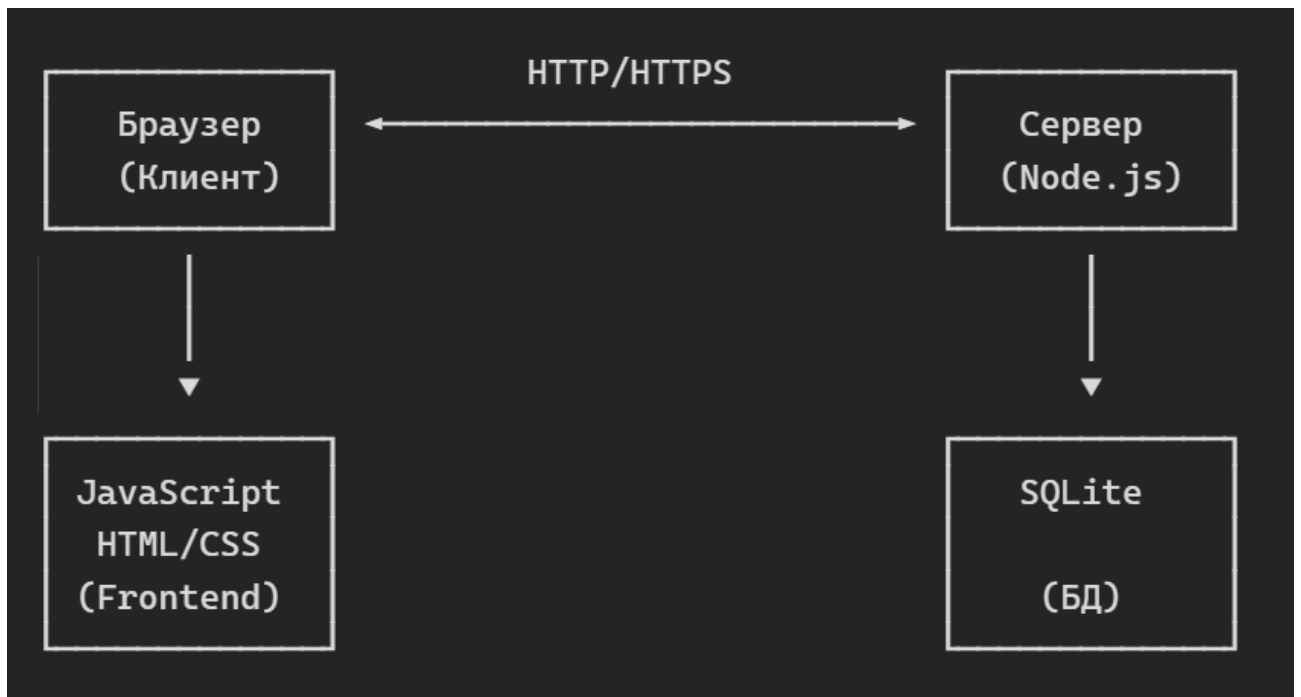


Рисунок 1 – Архитектура приложения

#### 2.1.4 Структура проекта

Структура проекта представлена ниже:

— server.js	# Главный файл сервера
— database.js	# Инициализация и работа с БД
— package.json	# Зависимости проекта
— create-admin.js	# Скрипт создания администратора
— routes/	
— auth.js	# Маршруты аутентификации
— tasks.js	# Маршруты для работы с задачами
— security/	
— session.js	# Управление сессиями и HMAC
— csrf.js	# CSRF защита
— validation.js	# Валидация и санитизация данных
— public/	
— index.html	# Главная страница
— styles.css	# Стили
— app.js	# Клиентский JavaScript

### 2.2 Аутентификация

#### 2.2.1 Регистрация пользователя

Регистрация происходит по следующему сценарию:

- Ввод имени пользователя (3-20 символов, только буквы, цифры, подчёркивание);

- Ввод пароля (минимум 6 символов);
- Ввод подтверждение пароля;
- Автоматический вход после успешной регистрации;
- Выдача сессионной cookie и CSRF токена.

### **2.2.2 Вход в систему**

Регистрация происходит по следующему сценарию:

- Ввод имени пользователя и пароля;
- Проверка учётных данных;
- Выдача сессионной cookie и CSRF токена;
- Автоматическое обновление времени жизни сессии при активности.

### **2.2.3 Выход из системы**

Выход из системы происходит по следующему сценарию:

- Нажатие кнопки «Выход»;
- Удаление сессии из базы данных;
- Удаление всех CSRF токенов сессии;
- Очистка cookie на клиенте.

## **2.3 Управление задачами**

### **2.3.1 Создание задачи**

Создание задачи происходит по следующему сценарию:

- Ввод названия задачи (до 200 символов);
- Ввод описания задачи (до 1000 символов, опционально);
- Установка статуса выполнения (выполнено/не выполнено);
- Автоматическое сохранение времени создания.

### **2.3.2 Просмотр задач**

Просмотр задач реализован как:

- Отображение всех задач текущего пользователя;
- Отображение статуса выполнения;
- Отображение даты создания и обновления;
- Для администраторов: просмотр всех задач всех пользователей с указанием автора.

### **2.3.3 Редактирование задачи**

Редактирование задачи происходит по следующему сценарию:

- Нажатие на кнопку «Редактировать»;
- Изменение названия и описания;

- Изменение статуса выполнения;
- Автоматическое обновление времени изменения.

#### **2.3.4 Удаление задачи**

Удаление задачи реализовано следующим образом:

- Нажатие кнопки «Удалить»;
- Удаление задачи с подтверждением;
- Обычные пользователи могут удалять только свои задачи;
- Администраторы могут удалять любые задачи.

### **2.4 Административные функции**

#### **2.4.1 Просмотр всех задач**

Просмотр задач реализован со следующими особенностями:

- Доступно только администраторам
- Отображение задач всех пользователей
- Отображение имени пользователя-автора каждой задачи

#### **2.4.2 Поиск задач**

Поиск задач позволяет делать следующее:

- Поиск по имени пользователя;
- Поиск по названию задачи;
- Комбинированный поиск (оба фильтра одновременно).

### **2.5 Описание разработанных мер по защите информации**

#### **2.5.1 Аутентификация**

Пароли хешируются при регистрации. Реализовано с применением библиотеки «bcrypt». У каждого пароля уникальная соль. Защищает пароли при утечки базы данных (БД), потребует знать особенности хеширования на бэкенде.

Авторизация работает на сессиях с HMAC-подписью. Сессия живёт 30 минут. При активности за эти 30 минут сессия продлевается ещё на 30 с момента продления. Проверка подписи защищает от кражи и/или подделки cookie.

Используются «безопасные» куки, настроенные на работу в httpOnly режиме. Javascript не может получить доступ к таким куки. Параметр «strict» у опции «samesite» запрещает отправлять cookie при cross-site запросах. В демо-настройках параметр «secure» был поставлен на false. В продакшене необходимо ставить его на true для работы https (шифрование). Таким образом,

мы защищаемся от XSS, CSRF атак и проблем долгоживущих сессий. Старые сессии автоматически удаляются.

Для защиты от CSRF атаки генерируется CSRF-токен. Он случайный и привязан к сессии. Использовать его можно лишь 1 раз и в течение 1 часа с момента получения.

### **2.5.2 Защита от SQL Injection**

В проекте в эндпоинтах используются параметризованные SQL-запросы. Для защиты использовался встроенный механизм экранирования. Если переданная строка содержит нечто вроде «; SELECT \* FROM users;», то данное выражение будет корректно обработано, как текст или иной тип данных параметра в эндпоинте.

На стороне клиента при обработке все небезопасные HTML-символы экранируются, как часть работы по защите от XSS. Запросы содержат повышающие общую безопасность запросов заголовки:

- «X-Content-Type-Options: nosniff» – предотвращение MIME-sniffing;
- «X-Frame-Options: DENY» – защита от clickjacking;
- «X-XSS-Protection: 1; mode=block» – защита от XSS.

### **2.5.3 Валидация**

Данные форм валидируются. Имя пользователя может иметь длину от 3 до 20 символов, содержать только латинские буквы, цифры или подчёркивания. Проверяется через регулярное выражение. Пароль также валидируется. Может иметь длину от 6 до 100 символов. В работе с задачами поля названия и описания могут иметь длину текста до 200 и 1000 символов соответственно. Защищает от переполнения полей БД и XSS через названия/описания задач.

Реализовано ограничение числа запросов за единицу времени. С одного IP можно делать не более 100 запросов за 15 минут. Защищает от атак грубой силы, DoS атак, злоупотребления API-запросами.

### **2.5.4 Ролевая модель доступа**

Реализованы две роли: «user», «admin». Первая присваивается по умолчанию при регистрации. Вторая – в результате выполнения специального скрипта создания администратора на стороне сервера. Имена ролей говорят сами за себя. Первая роль – это обычный пользователь системы, может создавать, просматривать, редактировать, удалять свои задачи. Вторая роль – администраторская, которая позволяет просматривать, редактировать, удалять все задачи всех пользователей, искать нужные по имени пользователя и паролю. Также можно вести и свой личный TODO-список.

Проверки прав доступа реализованы через middlewares «requireAuth», «requireRole». Первая проверяет аутентификацию, вторая – роль. Также

реализована проверка прав на уровне данных. Ролевая модель защищает от неавторизованного доступа.

### 2.5.5 Защита от кражи Cookie

Реализация:

- «sameSite: strict» – cookie не отправляется при cross-site запросах;
- «httpOnly: true» – JavaScript не может прочитать cookie;
- HMAC подпись – невозможно подделать без секретного ключа;
- Ограниченное время жизни - автоматическое истечение.

## 2.6 Потоки данных

### 2.6.1 Регистрация

Поток регистрации представлен на рисунке 2.

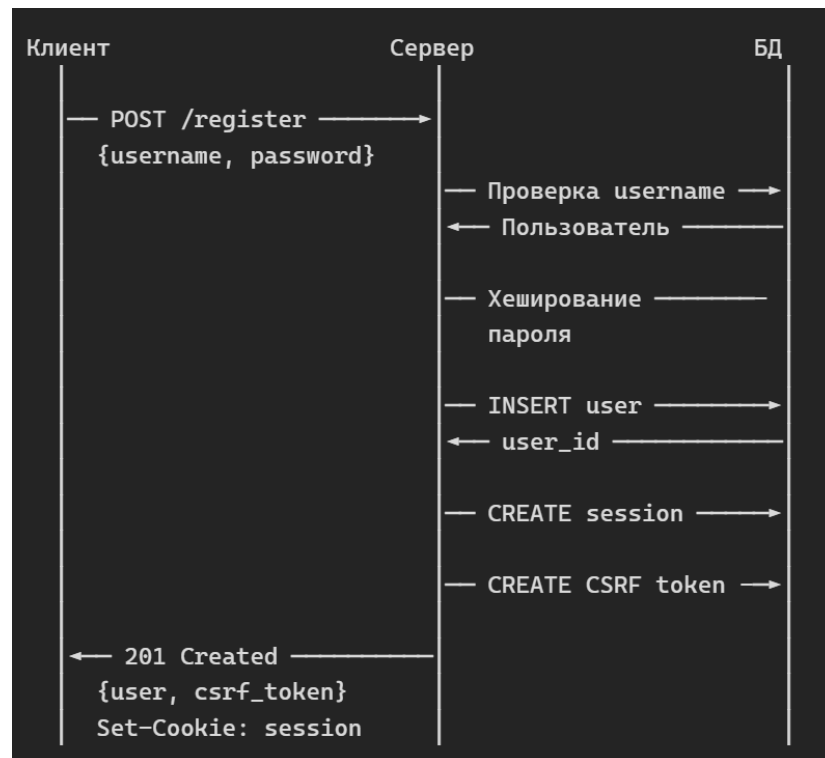


Рисунок 2 – Поток регистрации

### 2.6.2 Вход

Поток входа демонстрируется на рисунке 3.

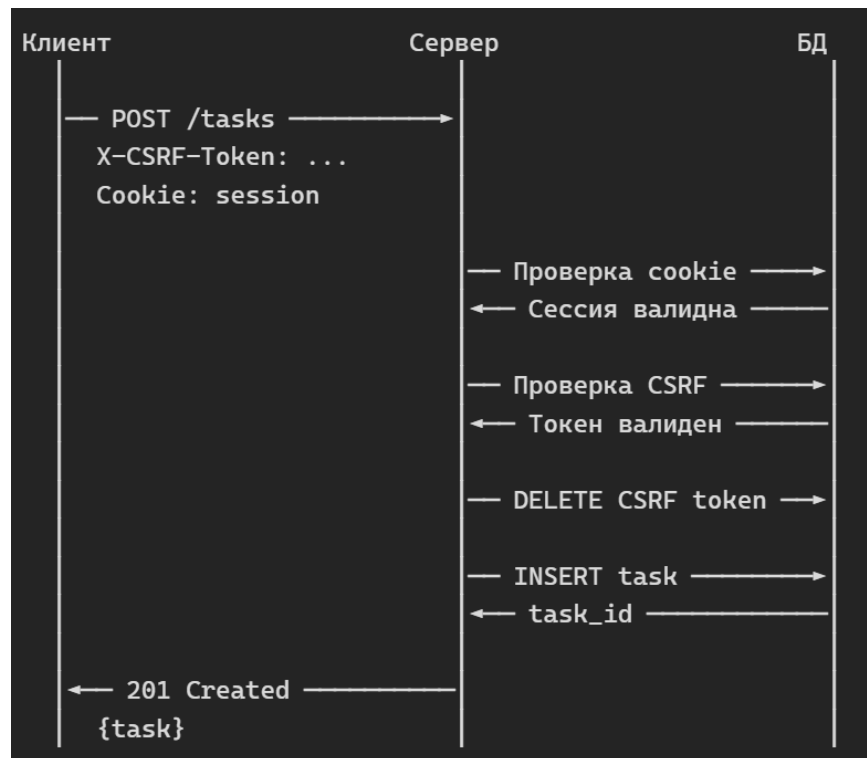


Рисунок 3 – Поток входа

## 2.7 Контрольные примеры

### 2.7.1 Регистрация

Демонстрируется на рисунках 4-5.

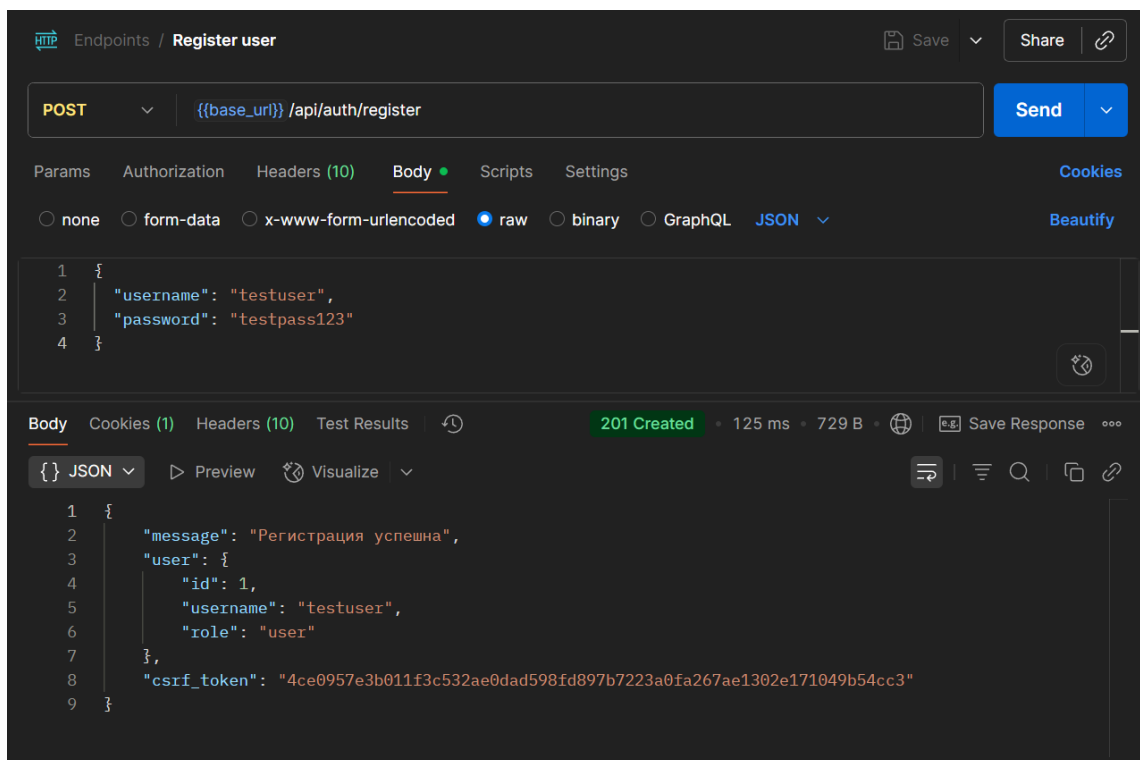


Рисунок 4 – Регистрация пользователей, пример 1

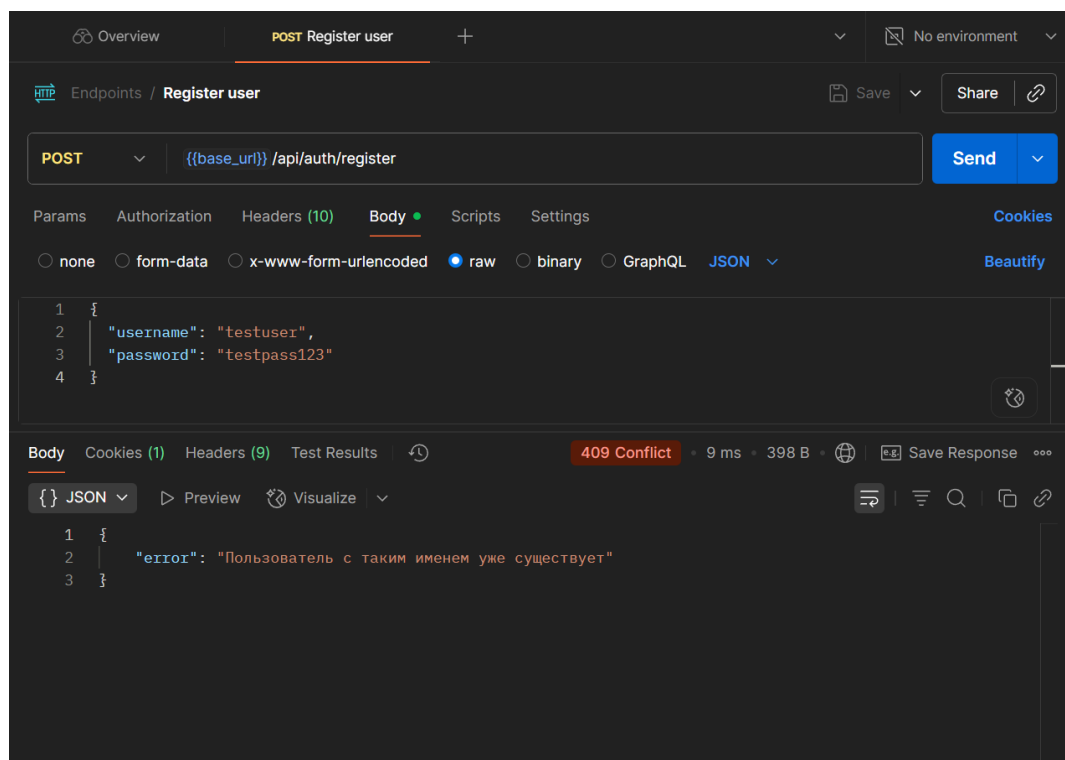


Рисунок 5 – Регистрация пользователей, пример 2

Регистрация пользователя оканчивает демонстрироваться на рисунке 6.

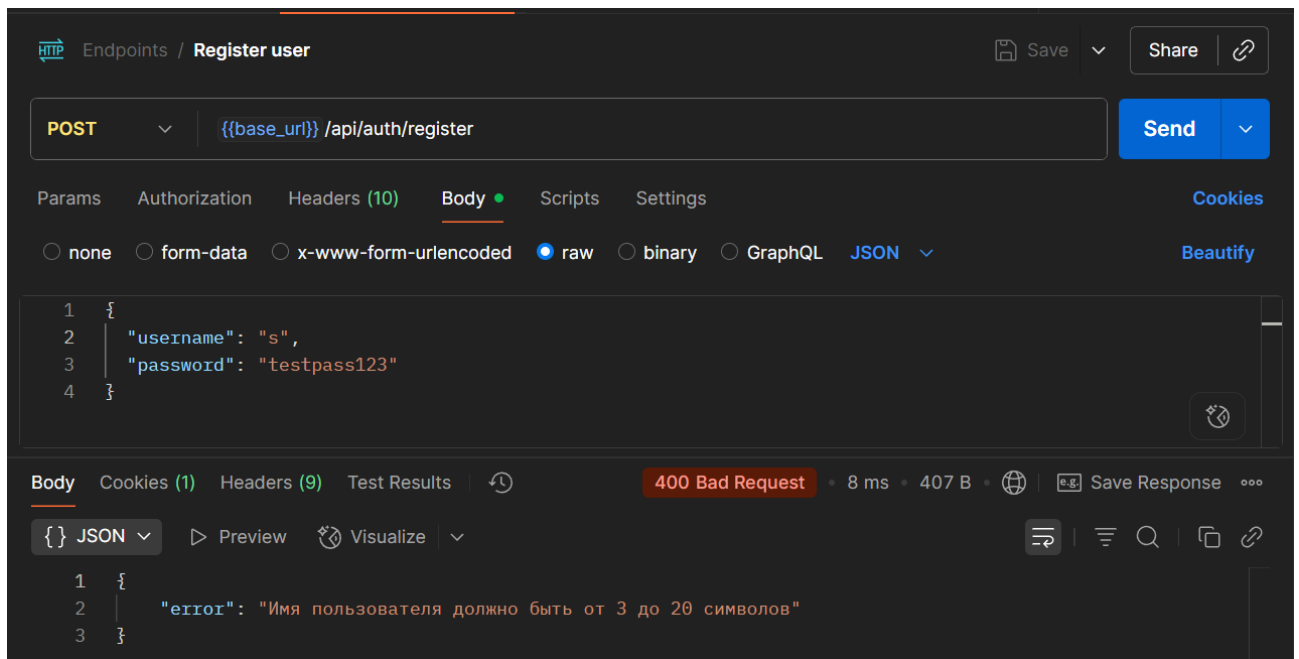


Рисунок 6 – Регистрация пользователей, пример 2

## 2.7.2 Вход в систему

Демонстрируется на рисунках 7-8.

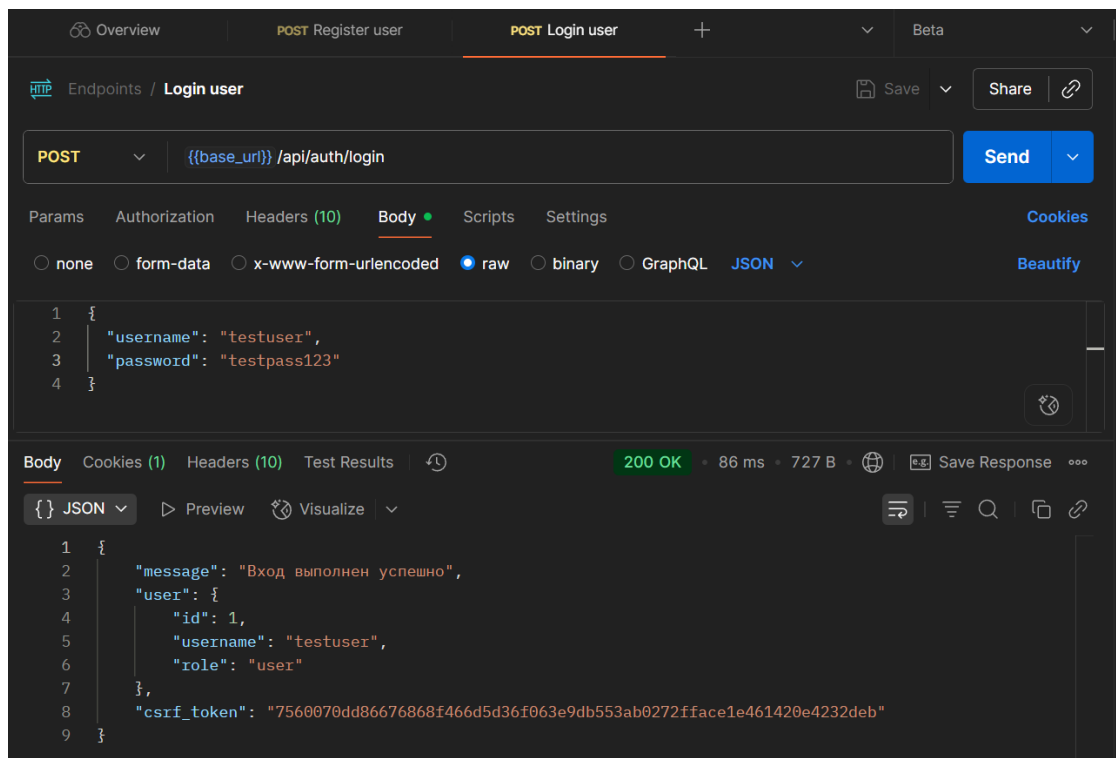


Рисунок 7 – Вход в систему, пример 1

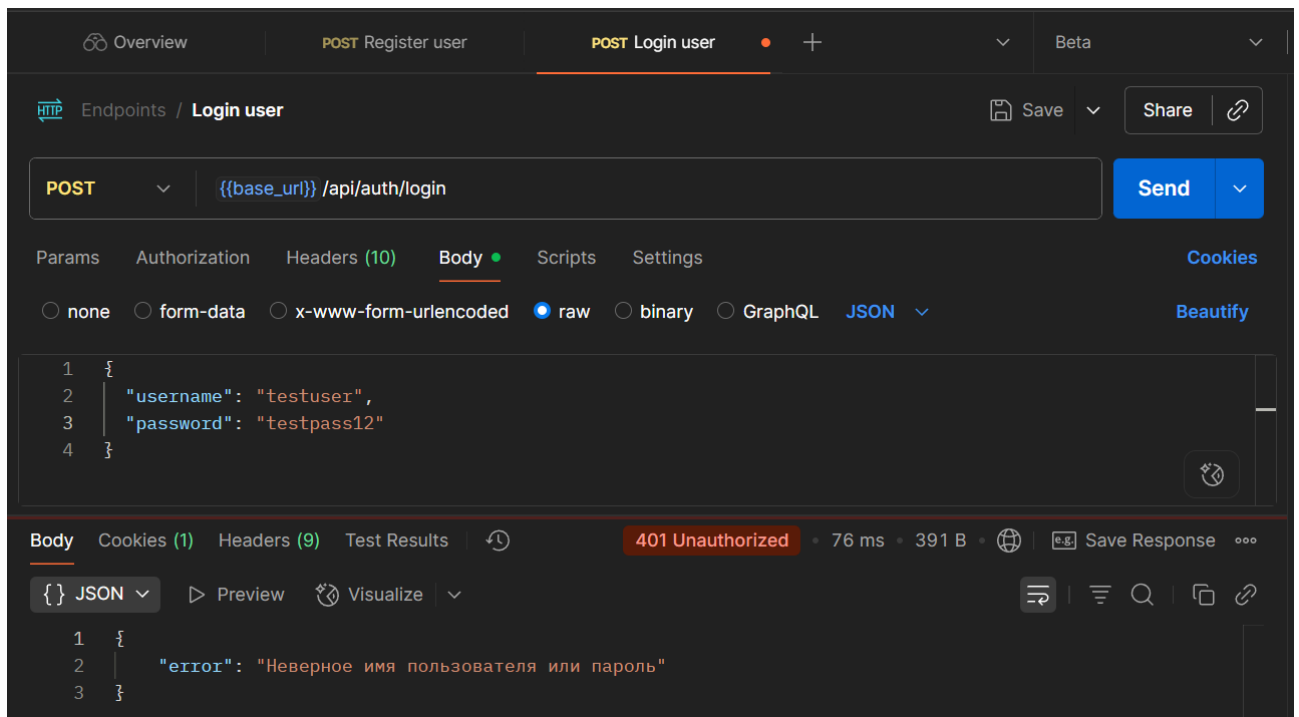


Рисунок 8 – Вход в систему, пример 2

### 2.7.3 Выход из системы

Демонстрируется на рисунке 9.

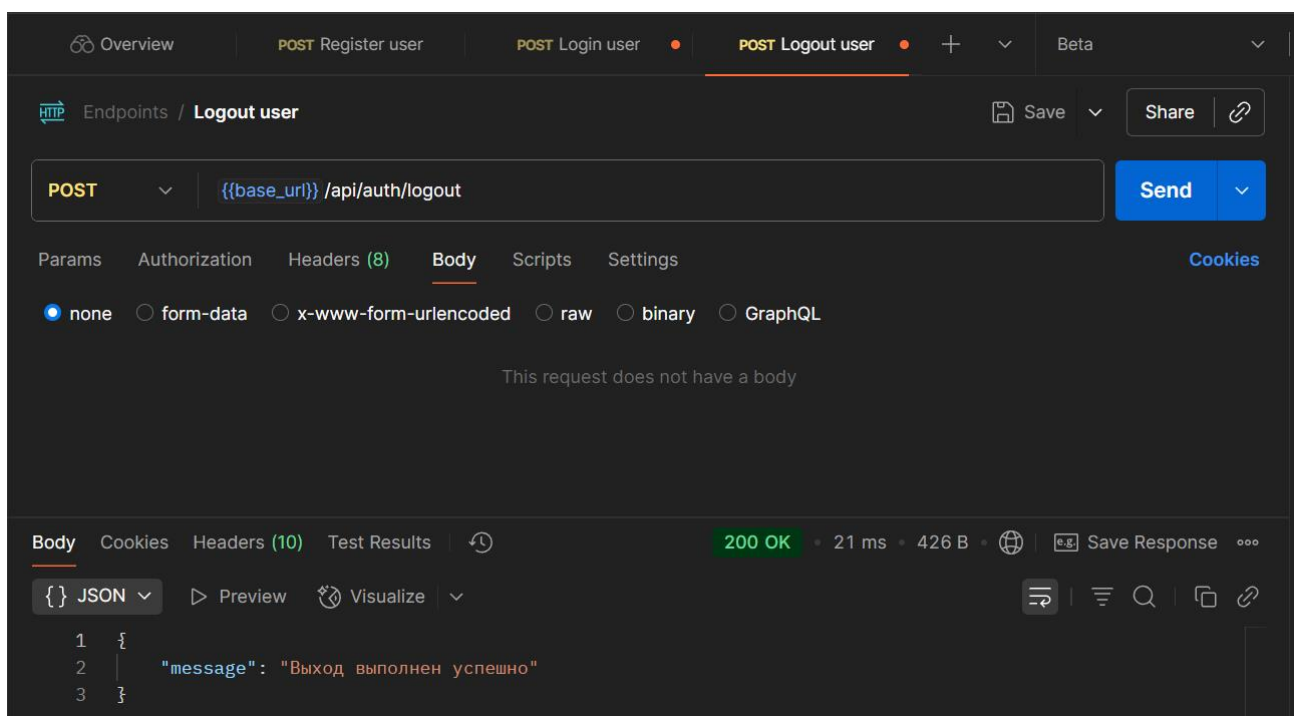


Рисунок 9 – Выход из системы

### 2.7.4 Получение информации о текущем пользователе

Демонстрируется на рисунках 10-11.

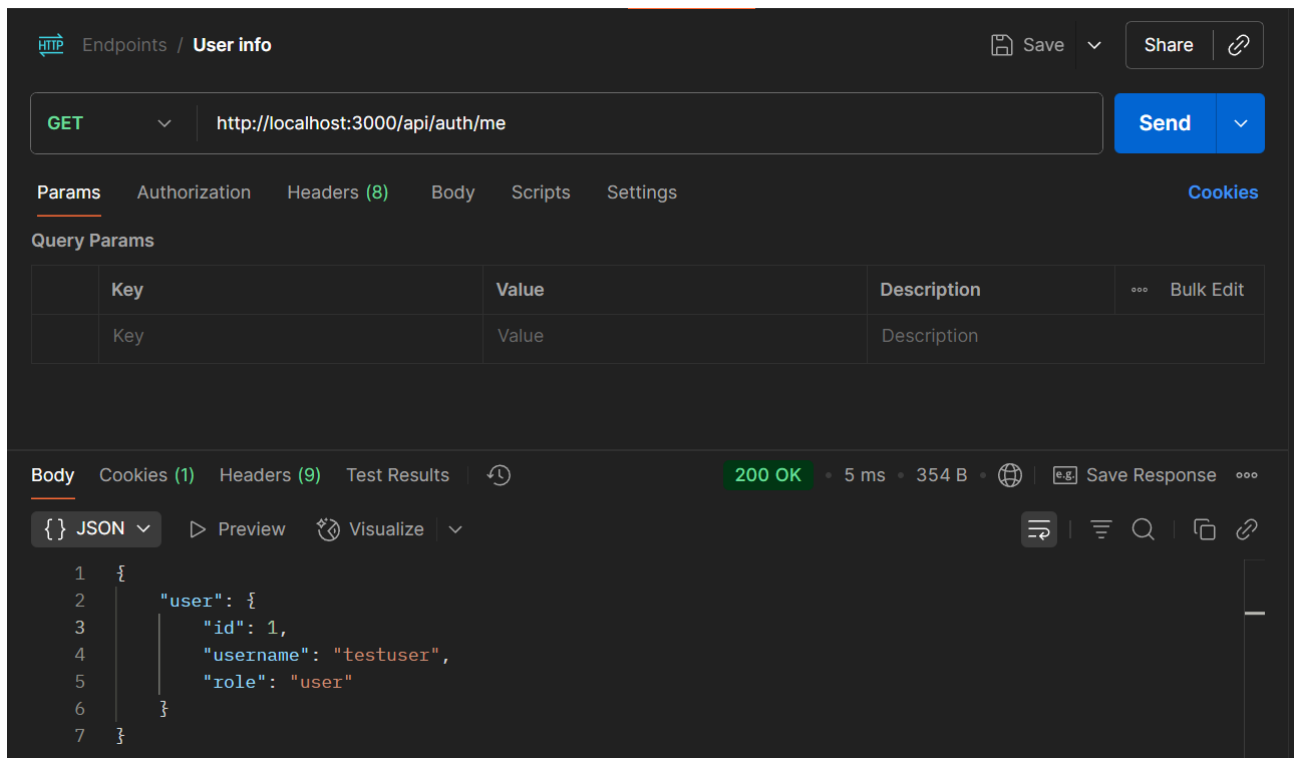


Рисунок 10 – Получение информации о текущем пользователе, часть 1

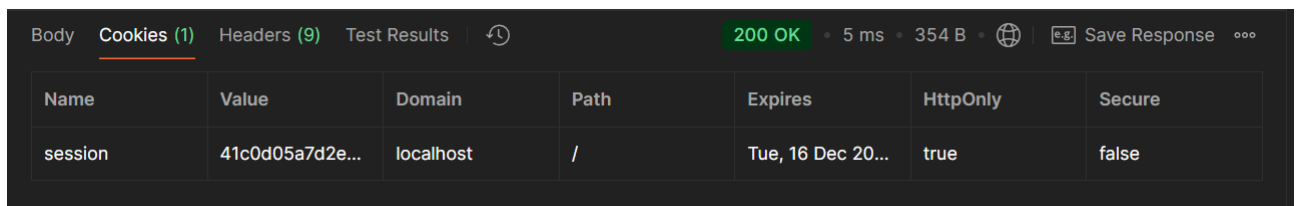


Рисунок 11 – Получение информации о текущем пользователе, часть 2

Демонстрация заканчивается на рисунке 12.

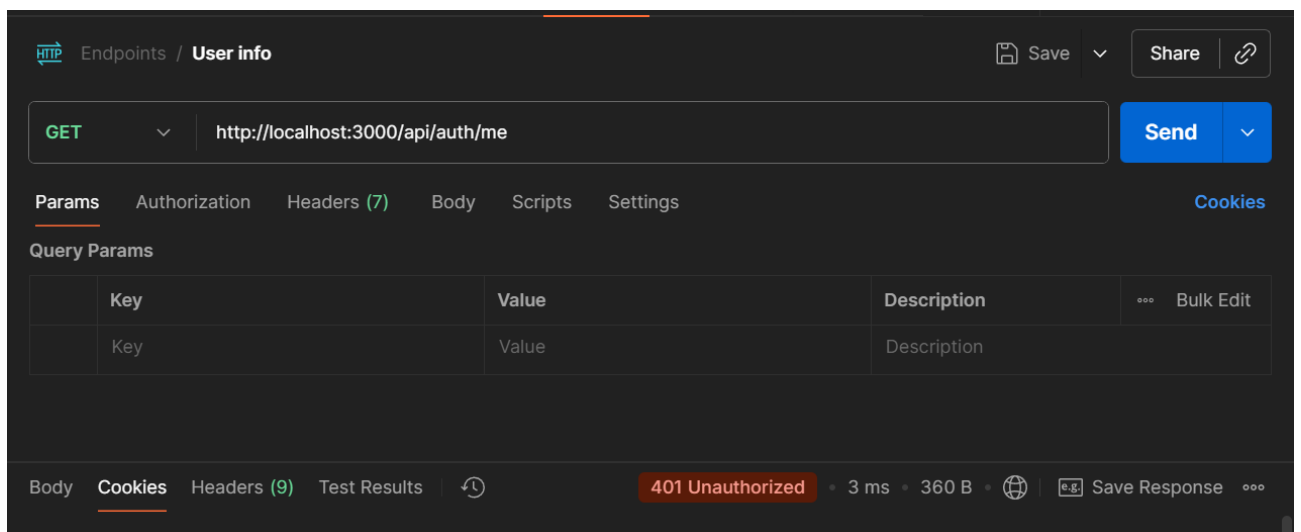


Рисунок 12 – Получение информации о текущем пользователе, часть 3

## 2.7.5 Доступ к защищённому маршруту (обычный пользователь)

Демонстрируется на рисунках 13-14.

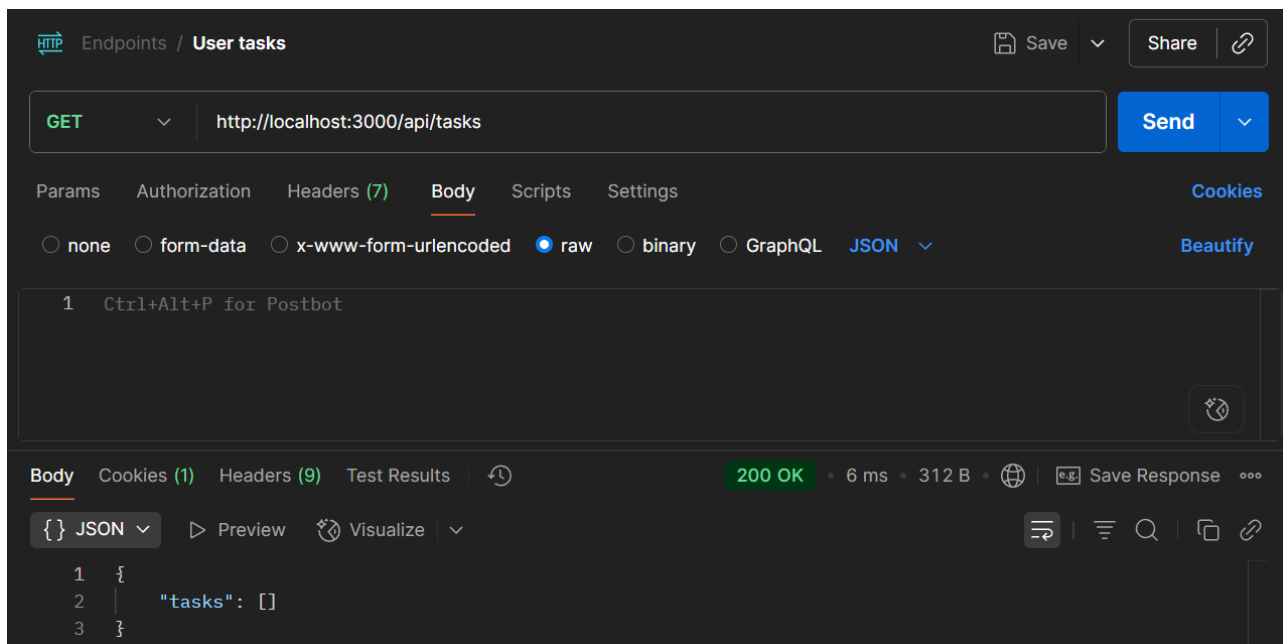


Рисунок 13 – Доступ к защищённому маршруту, часть 1

Name	Value	Domain	Path	Expires	HttpOnly	Secure
session	80fb64ed9f914...	localhost	/	Tue, 16 Dec 20...	true	false

Рисунок 14 – Доступ к защищённому маршруту, часть 2

## 2.7.6 Попытка доступа без аутентификации

Демонстрируется на рисунке 15.

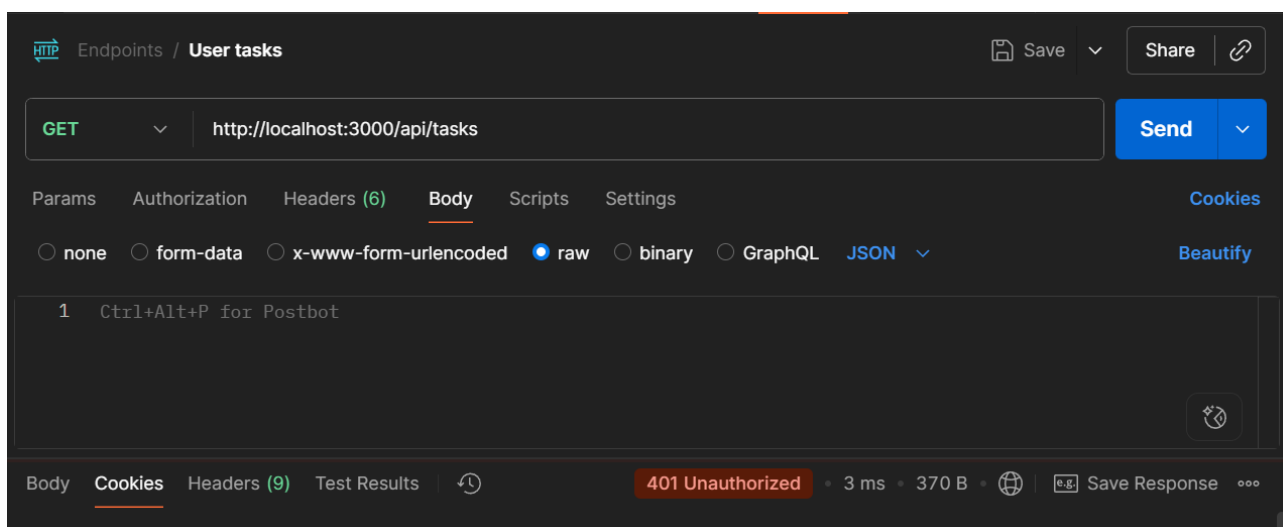


Рисунок 15 – Попытка доступа без аутентификации

### 2.7.7 Доступ к административному маршруту (обычный пользователь)

Демонстрируется на рисунке 16.

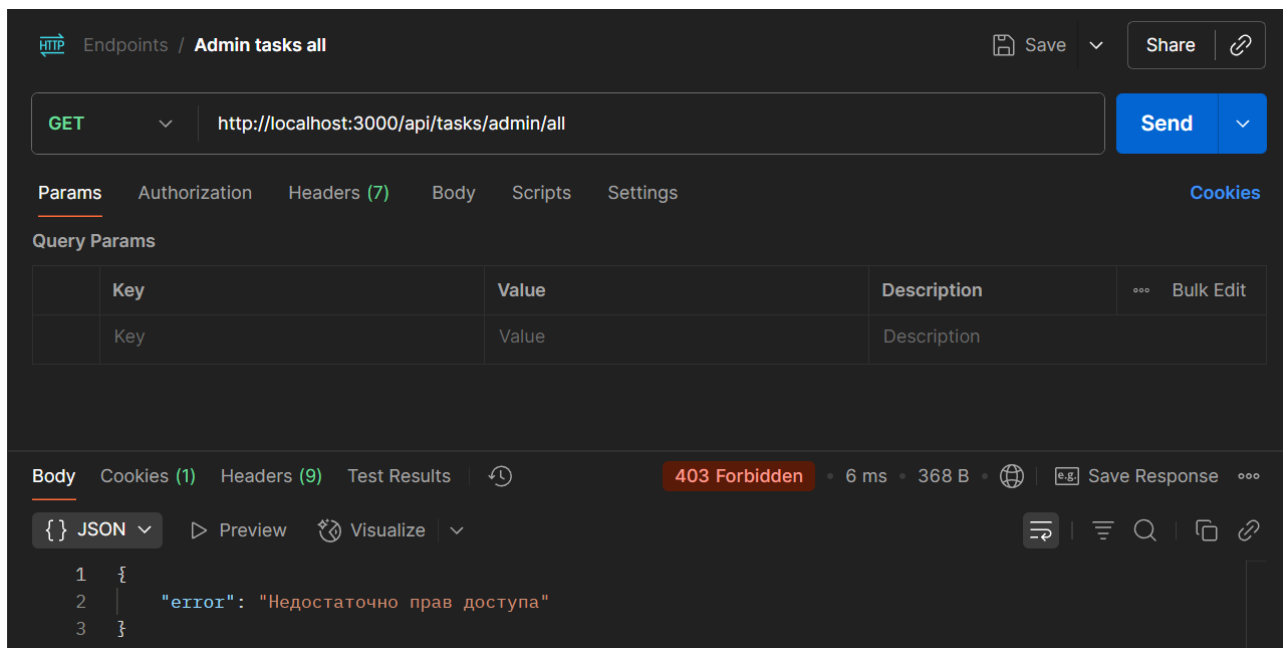


Рисунок 16 – Доступ к административному маршруту, часть 1

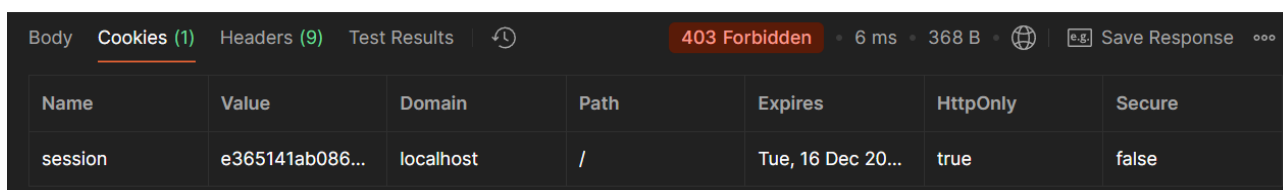


Рисунок 17 – Доступ к административному маршруту, часть 2

### 2.7.8 Доступ к административному маршруту (администратор)

Демонстрируется на рисунках 18-19.

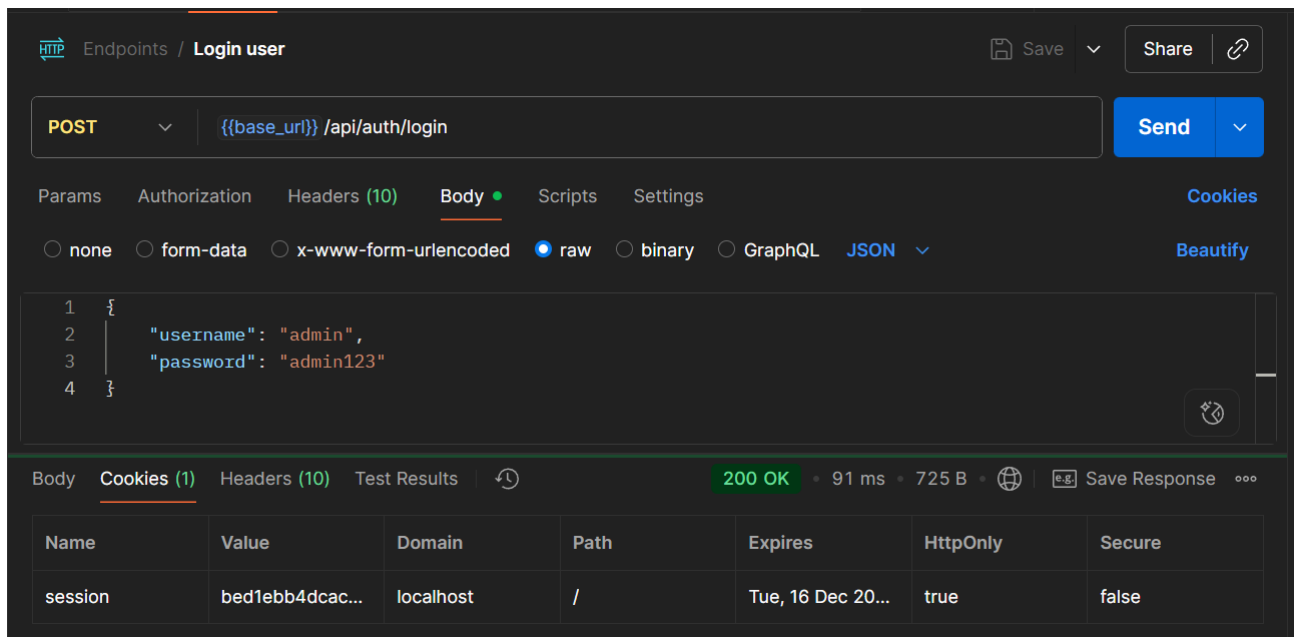


Рисунок 18 – Доступ к административному маршруту, часть 1

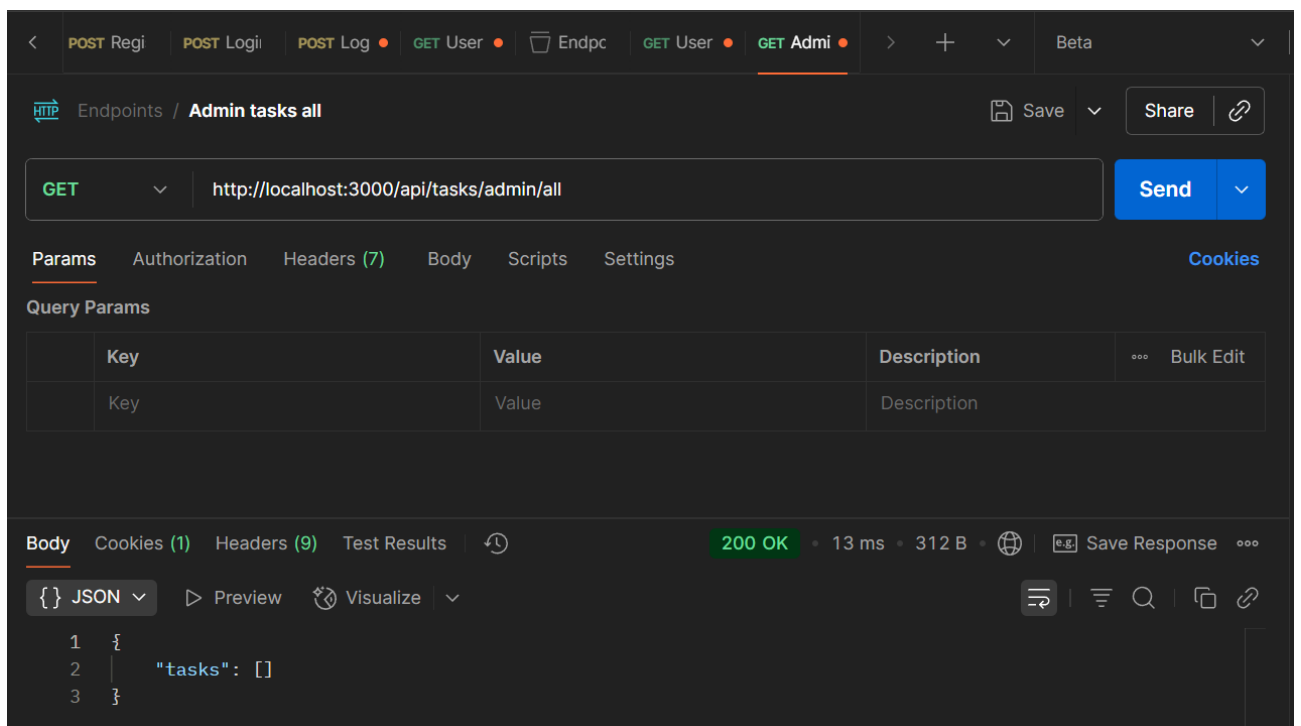


Рисунок 19 – Доступ к административному маршруту, часть 2

Демонстрация заканчивается на рисунке 20.

Name	Value	Domain	Path	Expires	HttpOnly	Secure
session	bed1ebb4dcac...	localhost	/	Tue, 16 Dec 20...	true	false

Рисунок 20 – Доступ к административному маршруту, часть 3

## **2.8 Листинг программы**

Листинг клиентской и серверной частей представлен в приложении А и Б соответственно.

### **3 ЗАКЛЮЧЕНИЕ**

По результатам работы был изучен теоретический материал по теме «Обеспечение безопасности web-приложений». Все поставленные цели и задачи были выполнены. Задания были выполнены и помогли лучше усвоить пройденный материал.

## ПРИЛОЖЕНИЕ А

### Листинг программы, клиентская часть

```
// app.js
// Глобальное состояние приложения
let currentUser = null;
let csrfToken = null;
let isAdminView = false;

// Инициализация при загрузке страницы
document.addEventListener('DOMContentLoaded', () => {
  initAuthTabs();
  initAuthForms();
  initTaskModal();
  initAdminSearch();
  checkAuth();
});

// Переключение между вкладками входа и регистрации
function initAuthTabs() {
  const tabs = document.querySelectorAll('.tab-btn');
  const loginForm = document.getElementById('login-form');
  const registerForm = document.getElementById('register-form');

  tabs.forEach(tab => {
    tab.addEventListener('click', () => {
      tabs.forEach(t => t.classList.remove('active'));
      tab.classList.add('active');

      if (tab.dataset.tab === 'login') {
        loginForm.style.display = 'block';
        registerForm.style.display = 'none';
      } else {
        loginForm.style.display = 'none';
        registerForm.style.display = 'block';
      }
    });
  });
}

// Инициализация форм аутентификации
function initAuthForms() {
  const loginForm = document.getElementById('login-form');
  const registerForm = document.getElementById('register-form');

  loginForm.addEventListener('submit', async (e) => {
    e.preventDefault();
    const errorDiv = document.getElementById('login-error');
    errorDiv.textContent = '';

    const formData = {
      username: document.getElementById('login-username').value.trim(),
      password: document.getElementById('login-password').value
    };

    try {
      console.log('Отправка запроса входа:', { username: formData.username,
password: '***' });
    }
  });
}
```

```

const response = await fetch('/api/auth/login', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(formData)
});

console.log('Ответ входа получен:', response.status,
response.statusText);

const data = await response.json();

if (response.ok) {
  csrfToken = data.csrf_token;
  currentUser = data.user;
  showApp();
  showMessage('Вход выполнен успешно', 'success');
  loginForm.reset();
} else {
  errorDiv.textContent = data.error || 'Ошибка входа';
}
} catch (error) {
  errorDiv.textContent = 'Ошибка соединения с сервером';
  console.error('Ошибка входа:', error);
}
});

registerForm.addEventListener('submit', async (e) => {
  e.preventDefault();
  const errorDiv = document.getElementById('register-error');
  errorDiv.textContent = '';

  const password = document.getElementById('register-password').value;
  const passwordConfirm = document.getElementById('register-password-
confirm').value;

  // Проверка совпадения паролей
  if (password !== passwordConfirm) {
    errorDiv.textContent = 'Пароли не совпадают';
    return;
  }

  const formData = {
    username: document.getElementById('register-username').value.trim(),
    password: password
  };

  try {
    // Логируем без пароля для безопасности
    console.log('Отправка запроса регистрации:', {
      username: formData.username,
      password: '***'
    });

    const response = await fetch('/api/auth/register', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(formData)
    });

```

```

    });

    console.log('Ответ регистрации получен:', response.status,
response.statusText);

    const data = await response.json();

    if (response.ok) {
        csrfToken = data.csrf_token;
        currentUser = data.user;
        showApp();
        showMessage('Регистрация успешна', 'success');
        registerForm.reset();
    } else {
        errorDiv.textContent = data.error || 'Ошибка регистрации';
    }
} catch (error) {
    errorDiv.textContent = 'Ошибка соединения с сервером';
    console.error('Ошибка регистрации:', error);
}
});

// Кнопка выхода
document.getElementById('logout-btn').addEventListener('click', async () => {
    try {
        await fetch('/api/auth/logout', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
                'X-CSRF-Token': csrfToken
            }
        });

        currentUser = null;
        csrfToken = null;
        showAuth();
        showMessage('Выход выполнен', 'success');
    } catch (error) {
        console.error('Ошибка выхода:', error);
    }
});
}

// Проверка текущей аутентификации
async function checkAuth() {
    try {
        const response = await fetch('/api/auth/me');
        const data = await response.json();

        if (response.ok) {
            currentUser = data.user;
            await getCsrfToken();
            showApp();
        } else {
            showAuth();
        }
    } catch (error) {
        showAuth();
    }
}
}

```

```

// Получение CSRF токена
async function getCsrftoken() {
  try {
    const response = await fetch('/api/auth/csrf-token');
    const data = await response.json();
    if (response.ok) {
      csrfToken = data.csrf_token;
    }
  } catch (error) {
    console.error('Ошибка получения CSRF токена:', error);
  }
}

// Показать форму аутентификации
function showAuth() {
  document.getElementById('auth-section').style.display = 'block';
  document.getElementById('app-section').style.display = 'none';
  document.getElementById('user-info').style.display = 'none';
}

// Показать основное приложение
function showApp() {
  document.getElementById('auth-section').style.display = 'none';
  document.getElementById('app-section').style.display = 'block';
  document.getElementById('user-info').style.display = 'flex';

  document.getElementById('username-display').textContent =
  currentUser.username;
  const roleBadge = document.getElementById('role-badge');
  roleBadge.textContent = currentUser.role === 'admin' ? 'Администратор' :
  'Пользователь';
  roleBadge.classList.toggle('admin', currentUser.role === 'admin');

  const adminBtn = document.getElementById('admin-view-btn');
  const adminSearch = document.getElementById('admin-search');
  adminBtn.style.display = currentUser.role === 'admin' ? 'block' : 'none';
  adminSearch.style.display = currentUser.role === 'admin' && isAdminView ?
  'block' : 'none';

  loadTasks();
}

// Загрузка задач
async function loadTasks() {
  try {
    let endpoint = isAdminView && currentUser.role === 'admin'
      ? '/api/tasks/admin/all'
      : '/api/tasks';

    // Добавляем параметры поиска для админки
    if (isAdminView && currentUser.role === 'admin') {
      const username = document.getElementById('search-
username').value.trim() || '';
      const taskTitle = document.getElementById('search-
task').value.trim() || '';

      const params = new URLSearchParams();
      if (username) params.append('username', username);
      if (taskTitle) params.append('task', taskTitle);

      if (params.toString()) {

```

```

        endpoint += '?' + params.toString();
    }
}

const response = await fetch(endpoint, {
    headers: {
        'X-CSRF-Token': csrfToken
    }
});

const data = await response.json();

if (response.ok) {
    displayTasks(data.tasks || []);
} else {
    if (response.status === 401) {
        showAuth();
    } else {
        showMessage(data.error || 'Ошибка загрузки задач', 'error');
    }
}
} catch (error) {
    showMessage('Ошибка соединения с сервером', 'error');
    console.error('Ошибка загрузки задач:', error);
}
}

// Отображение задач
function displayTasks(tasks) {
    const container = document.getElementById('tasks-container');

    if (tasks.length === 0) {
        container.innerHTML = `
            <div class="empty-state">
                <h3>Нет задач</h3>
                <p>Создайте свою первую задачу!</p>
            </div>
        `;
        return;
    }

    container.innerHTML = tasks.map(task => `
        <div class="task-card ${task.completed ? 'completed' : ''}">
            <div class="task-title">${escapeHtml(task.title)}</div>
            ${task.description ? `<div class="task-description">${escapeHtml(task.description)}</div>` : ''}
            <div class="task-meta">
                Создано: ${new Date(task.created_at).toLocaleString('ru-RU')}
                ${task.username ? ` | Пользователь: ${escapeHtml(task.username)}` : ''}
            </div>
            <div class="task-actions">
                <button class="btn btn-edit"
                    onclick="editTask(${task.id})">Редактировать</button>
                <button class="btn btn-delete"
                    onclick="deleteTask(${task.id})">Удалить</button>
            </div>
        </div>
    `).join('');
}

```

```

// Защита от XSS
function escapeHtml(text) {
  const div = document.createElement('div');
  div.textContent = text;
  return div.innerHTML;
}

// Инициализация модального окна
function initTaskModal() {
  const modal = document.getElementById('task-modal');
  const closeBtn = document.querySelector('.close');
  const cancelBtn = document.getElementById('cancel-btn');
  const form = document.getElementById('task-form');

  closeBtn.addEventListener('click', () => {
    modal.style.display = 'none';
  });

  cancelBtn.addEventListener('click', () => {
    modal.style.display = 'none';
  });

  window.addEventListener('click', (e) => {
    if (e.target === modal) {
      modal.style.display = 'none';
    }
  });

  form.addEventListener('submit', async (e) => {
    e.preventDefault();
    await saveTask();
  });

  document.getElementById('add-task-btn').addEventListener('click', () => {
    openTaskModal();
  });

  document.getElementById('admin-view-btn').addEventListener('click', () => {
    isAdminView = !isAdminView;
    const adminBtn = document.getElementById('admin-view-btn');
    const adminSearch = document.getElementById('admin-search');
    const tasksTitle = document.getElementById('tasks-title');

    adminBtn.textContent = isAdminView ? 'Мои задачи' : 'Все задачи (админ)';
    adminSearch.style.display = isAdminView && currentUser?.role === 'admin'
      ? 'block' : 'none';
    tasksTitle.textContent = isAdminView ? 'Все задачи (админ)' : 'Мои задачи';

    // Очищаем поиск при переключении
    if (!isAdminView) {
      document.getElementById('search-username').value = '';
      document.getElementById('search-task').value = '';
    }

    loadTasks();
  });
}

// Инициализация поиска в админке
function initAdminSearch() {

```

```

const searchBtn = document.getElementById('search-btn');
const clearSearchBtn = document.getElementById('clear-search-btn');
const searchUsername = document.getElementById('search-username');
const searchTask = document.getElementById('search-task');

if (searchBtn) {
  searchBtn.addEventListener('click', () => {
    loadTasks();
  });
}

if (clearSearchBtn) {
  clearSearchBtn.addEventListener('click', () => {
    if (searchUsername) searchUsername.value = '';
    if (searchTask) searchTask.value = '';
    loadTasks();
  });
}

// Поиск при нажатии Enter
if (searchUsername) {
  searchUsername.addEventListener('keypress', (e) => {
    if (e.key === 'Enter') {
      loadTasks();
    }
  });
}

if (searchTask) {
  searchTask.addEventListener('keypress', (e) => {
    if (e.key === 'Enter') {
      loadTasks();
    }
  });
}
}

// Открыть модальное окно для создания/редактирования задачи
function openTaskModal(task = null) {
  const modal = document.getElementById('task-modal');
  const title = document.getElementById('modal-title');
  const form = document.getElementById('task-form');
  const errorDiv = document.getElementById('task-error');

  errorDiv.textContent = '';

  if (task) {
    title.textContent = 'Редактировать задачу';
    document.getElementById('task-id').value = task.id;
    document.getElementById('task-title').value = task.title;
    document.getElementById('task-description').value = task.description ||
    '';
    document.getElementById('task-completed').checked = task.completed === 1;
  } else {
    title.textContent = 'Добавить задачу';
    form.reset();
    document.getElementById('task-id').value = '';
  }

  modal.style.display = 'block';
}

```

```

// Сохранение задачи
async function saveTask() {
  const errorDiv = document.getElementById('task-error');
  errorDiv.textContent = '';

  const taskId = document.getElementById('task-id').value;
  const taskData = {
    title: document.getElementById('task-title').value.trim(),
    description: document.getElementById('task-description').value.trim(),
    completed: document.getElementById('task-completed').checked,
    csrf_token: csrfToken
  };

  try {
    const url = taskId ? `/api/tasks/${taskId}` : '/api/tasks';
    const method = taskId ? 'PUT' : 'POST';

    // Логим без CSRF токена для безопасности
    const logData = { ...taskData };
    delete logData.csrf_token;
    console.log('Отправка запроса:', method, url, { ...logData, csrf_token:
'****' });

    const response = await fetch(url, {
      method: method,
      headers: {
        'Content-Type': 'application/json',
        'X-CSRF-Token': csrfToken
      },
      body: JSON.stringify(taskData)
    });

    console.log('Ответ получен:', response.status, response.statusText);

    const data = await response.json();

    if (response.ok) {
      document.getElementById('task-modal').style.display = 'none';
      await getCsrfToken(); // Получаем новый токен после использования
      loadTasks();
      showMessage(taskId ? 'Задача обновлена' : 'Задача создана',
'success');
    } else {
      if (response.status === 401) {
        showAuth();
      } else {
        errorDiv.textContent = data.error || 'Ошибка сохранения задачи';
      }
    }
  } catch (error) {
    errorDiv.textContent = 'Ошибка соединения с сервером';
    console.error('Ошибка сохранения задачи:', error);
  }
}

// Редактирование задачи
async function editTask(id) {
  try {
    const response = await fetch(`/api/tasks/${id}`, {
      headers: {

```

```

        'X-CSRF-Token': csrfToken
    }
  });

  const data = await response.json();

  if (response.ok) {
    openTaskModal(data.task);
  } else {
    if (response.status === 401) {
      showAuth();
    } else {
      showMessage(data.error || 'Ошибка загрузки задачи', 'error');
    }
  }
} catch (error) {
  showMessage('Ошибка соединения с сервером', 'error');
  console.error('Ошибка загрузки задачи:', error);
}

}

// Удаление задачи
async function deleteTask(id) {
  if (!confirm('Вы уверены, что хотите удалить эту задачу?')) {
    return;
  }

  try {
    const response = await fetch(`/api/tasks/${id}`, {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
        'X-CSRF-Token': csrfToken
      }
    });

    const data = await response.json();

    if (response.ok) {
      await getCsrfToken(); // Получаем новый токен после использования
      loadTasks();
      showMessage('Задача удалена', 'success');
    } else {
      if (response.status === 401) {
        showAuth();
      } else {
        showMessage(data.error || 'Ошибка удаления задачи', 'error');
      }
    }
  } catch (error) {
    showMessage('Ошибка соединения с сервером', 'error');
    console.error('Ошибка удаления задачи:', error);
  }
}

// Показать сообщение
function showMessage(text, type = 'success') {
  const messageDiv = document.getElementById('message');
  messageDiv.textContent = text;
  messageDiv.className = `message ${type}`;
  messageDiv.style.display = 'block';
}

```

```

        setTimeout(() => {
            messageDiv.style.display = 'none';
        }, 3000);
    }

<!-- index.html -->
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Система управления задачами</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Система управления задачами</h1>
            <div id="user-info" class="user-info" style="display: none;">
                <span id="username-display"></span>
                <span id="role-badge" class="role-badge"></span>
                <button id="logout-btn" class="btn btn-secondary">Выход</button>
            </div>
        </header>

        <!-- Форма входа/регистрации -->
        <div id="auth-section" class="auth-section">
            <div class="auth-tabs">
                <button class="tab-btn active" data-tab="login">Вход</button>
                <button class="tab-btn" data-tab="register">Регистрация</button>
            </div>

            <form id="login-form" class="auth-form">
                <h2>Вход в систему</h2>
                <div class="form-group">
                    <label for="login-username">Имя пользователя:</label>
                    <input type="text" id="login-username" name="username"
required minlength="3" maxlength="20">
                </div>
                <div class="form-group">
                    <label for="login-password">Пароль:</label>
                    <input type="password" id="login-password" name="password"
required minlength="6">
                </div>
                <button type="submit" class="btn btn-primary">Войти</button>
                <div id="login-error" class="error-message"></div>
            </form>

            <form id="register-form" class="auth-form" style="display: none;">
                <h2>Регистрация</h2>
                <div class="form-group">
                    <label for="register-username">Имя пользователя:</label>
                    <input type="text" id="register-username" name="username"
required minlength="3" maxlength="20" pattern="[a-zA-Z0-9_]+">
                    <small>Только буквы, цифры и подчёркивание</small>
                </div>
                <div class="form-group">
                    <label for="register-password">Пароль:</label>
                    <input type="password" id="register-password" name="password"
required minlength="6">

```

```

        <small>Минимум 6 символов</small>
    </div>
    <div class="form-group">
        <label
пароля:</label>
            <input
                type="password"
                id="register-password-confirm"
name="password-confirm" required minlength="6">
                <small>Повторите пароль</small>
            </div>
            <button
                type="submit"
                class="btn
                btn-
primary">Зарегистрироваться</button>
            <div id="register-error" class="error-message"></div>
        </form>
    </div>

    <!-- Основное приложение -->
    <div id="app-section" class="app-section" style="display: none;">
        <div class="tasks-header">
            <h2 id="tasks-title">Мои задачи</h2>
            <button
                id="add-task-btn"
                class="btn
                btn-primary">Добавить
задачу</button>
            <button id="admin-view-btn" class="btn btn-admin" style="display:
none;">Все задачи (админ)</button>
        </div>

        <!-- Поля поиска для админки -->
        <div id="admin-search" class="admin-search" style="display: none;">
            <div class="search-fields">
                <div class="search-field">
                    <label
пользователя:</label>
                        <input
                            type="text"
                            id="search-username"
placeholder="Введите имя пользователя">
                        </div>
                <div class="search-field">
                    <label
задачи:</label>
                        <input type="text" id="search-task" placeholder="Введите
название задачи">
                    </div>
                <button
                    id="search-btn"
                    class="btn
                    btn-
primary">Поиск</button>
                <button
                    id="clear-search-btn"
                    class="btn
                    btn-
secondary">Очистить</button>
            </div>
        </div>

        <div id="tasks-container" class="tasks-container">
            <!-- Задачи будут добавлены через JavaScript -->
        </div>

        <!-- Модальное окно для добавления/редактирования задачи -->
        <div id="task-modal" class="modal" style="display: none;">
            <div class="modal-content">
                <span class="close">&times;</span>
                <h2 id="modal-title">Добавить задачу</h2>
                <form id="task-form">
                    <input type="hidden" id="task-id">
                    <div class="form-group">
                        <label for="task-title">Название:</label>

```

```

        <input type="text" id="task-title" required
maxlength="200">
    </div>
    <div class="form-group">
        <label for="task-description">Описание:</label>
        <textarea id="task-description" rows="4"
maxlength="1000"></textarea>
    </div>
    <div class="form-group">
        <label for="task-completed">
            Выполнено
            <input type="checkbox" id="task-completed">
        </label>
    </div>
    <div class="modal-actions">
        <button type="submit" class="btn btn-
primary">Сохранить</button>
        <button type="button" class="btn btn-secondary"
id="cancel-btn">Отмена</button>
    </div>
    <div id="task-error" class="error-message"></div>
</form>
</div>
</div>
</div>

    <div id="message" class="message"></div>
</div>

    <script src="app.js"></script>
</body>
</html>

<!-- index.html -->
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Система управления задачами</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>Система управления задачами</h1>
            <div id="user-info" class="user-info" style="display: none;">
                <span id="username-display"></span>
                <span id="role-badge" class="role-badge"></span>
                <button id="logout-btn" class="btn btn-secondary">Выход</button>
            </div>
        </header>

        <!-- Форма входа/регистрации -->
        <div id="auth-section" class="auth-section">
            <div class="auth-tabs">
                <button class="tab-btn active" data-tab="login">Вход</button>
                <button class="tab-btn" data-tab="register">Регистрация</button>
            </div>

            <form id="login-form" class="auth-form">

```

```

        <h2>Вход в систему</h2>
        <div class="form-group">
            <label for="login-username">Имя пользователя:</label>
            <input type="text" id="login-username" name="username"
required minlength="3" maxlength="20">
        </div>
        <div class="form-group">
            <label for="login-password">Пароль:</label>
            <input type="password" id="login-password" name="password"
required minlength="6">
        </div>
        <button type="submit" class="btn btn-primary">Войти</button>
        <div id="login-error" class="error-message"></div>
    </form>

    <form id="register-form" class="auth-form" style="display: none;">
        <h2>Регистрация</h2>
        <div class="form-group">
            <label for="register-username">Имя пользователя:</label>
            <input type="text" id="register-username" name="username"
required minlength="3" maxlength="20" pattern="[a-zA-Z0-9_]+">
            <small>Только буквы, цифры и подчёркивание</small>
        </div>
        <div class="form-group">
            <label for="register-password">Пароль:</label>
            <input type="password" id="register-password" name="password"
required minlength="6">
            <small>Минимум 6 символов</small>
        </div>
        <div class="form-group">
            <label
пароля:</label>
            <input type="password" id="register-password-confirm"
name="password-confirm" required minlength="6">
            <small>Повторите пароль</small>
        </div>
        <button type="submit" class="btn btn-
primary">Зарегистрироваться</button>
        <div id="register-error" class="error-message"></div>
    </form>
</div>

<!-- Основное приложение -->
<div id="app-section" class="app-section" style="display: none;">
    <div class="tasks-header">
        <h2 id="tasks-title">Мои задачи</h2>
        <button id="add-task-btn" class="btn btn-primary">Добавить
задачу</button>
        <button id="admin-view-btn" class="btn btn-admin" style="display:
none;">Все задачи (админ)</button>
    </div>

    <!-- Поля поиска для админки -->
    <div id="admin-search" class="admin-search" style="display: none;">
        <div class="search-fields">
            <div class="search-field">
                <label for="search-username">Поиск по имени
пользователя:</label>
                <input type="text" id="search-username"
placeholder="Введите имя пользователя">
            </div>

```

```

        <div class="search-field">
            <label        for="search-task">Поиск        по        названию
задачи:</label>
            <input type="text" id="search-task" placeholder="Введите
название задачи">
        </div>
        <button        id="search-btn"        class="btn        btn-
primary">Поиск</button>
        <button        id="clear-search-btn"        class="btn        btn-
secondary">Очистить</button>
    </div>
</div>

<div id="tasks-container" class="tasks-container">
    <!-- Задачи будут добавлены через JavaScript -->
</div>

<!-- Модальное окно для добавления/редактирования задачи -->
<div id="task-modal" class="modal" style="display: none;">
    <div class="modal-content">
        <span class="close">&times;</span>
        <h2 id="modal-title">Добавить задачу</h2>
        <form id="task-form">
            <input type="hidden" id="task-id">
            <div class="form-group">
                <label for="task-title">Название:</label>
                <input        type="text"        id="task-title"        required
maxlength="200">
            </div>
            <div class="form-group">
                <label for="task-description">Описание:</label>
                <textarea        id="task-description"        rows="4"
maxlength="1000"></textarea>
            </div>
            <div class="form-group">
                <label for="task-completed">
                    Выполнено
                    <input type="checkbox" id="task-completed">
                </label>
            </div>
            <div class="modal-actions">
                <button        type="submit"        class="btn        btn-
primary">Сохранить</button>
                <button        type="button"        class="btn        btn-secondary"
id="cancel-btn">Отмена</button>
            </div>
            <div id="task-error" class="error-message"></div>
        </form>
    </div>
</div>

<div id="message" class="message"></div>
</div>

<script src="app.js"></script>
</body>
</html>

```

## ПРИЛОЖЕНИЕ Б

### Листинг программы, серверная часть

```
// server.js
const express = require('express');
const cookieParser = require('cookie-parser');
const path = require('path');
const { initDatabase } = require('./database');
const authRoutes = require('./routes/auth');
const taskRoutes = require('./routes/tasks');

const app = express();
const PORT = process.env.PORT || 3000;

// Флаг готовности БД
let dbReady = false;

// Middleware
app.use(express.json({ limit: '10mb' })); // Ограничение размера тела запроса
app.use(express.urlencoded({ extended: true, limit: '10mb' }));
app.use(cookieParser());

// Защита от некоторых атак через заголовки
app.use((req, res, next) => {
  // Защита от XSS
  res.setHeader('X-Content-Type-Options', 'nosniff');
  res.setHeader('X-Frame-Options', 'DENY');
  res.setHeader('X-XSS-Protection', '1; mode=block');

  // Удаление информации о сервере
  res.removeHeader('X-Powered-By');

  next();
});

// Простой rate limiting (базовая защита от брутфорса)
const rateLimitMap = new Map();
const RATE_LIMIT_WINDOW = 15 * 60 * 1000; // 15 минут
const RATE_LIMIT_MAX_REQUESTS = 100; // максимум запросов

app.use((req, res, next) => {
  const ip = req.ip || req.connection.remoteAddress;
  const now = Date.now();

  if (!rateLimitMap.has(ip)) {
    rateLimitMap.set(ip, { count: 1, resetTime: now + RATE_LIMIT_WINDOW });
    return next();
  }

  const limit = rateLimitMap.get(ip);

  if (now > limit.resetTime) {
    limit.count = 1;
    limit.resetTime = now + RATE_LIMIT_WINDOW;
    return next();
  }

  if (limit.count >= RATE_LIMIT_MAX_REQUESTS) {

```

```

        return res.status(429).json({ error: 'Слишком много запросов. Попробуйте
        позже.' });
    }

    limit.count++;
    next();
});

// Очистка старых записей rate limiting
setInterval(() => {
    const now = Date.now();
    for (const [ip, limit] of rateLimitMap.entries()) {
        if (now > limit.resetTime) {
            rateLimitMap.delete(ip);
        }
    }
}, RATE_LIMIT_WINDOW);

// Статические файлы (клиентская часть)
app.use(express.static(path.join(__dirname, 'public')));

// Middleware для проверки готовности БД
// Должен быть ПЕРЕД регистрацией API маршрутов, чтобы защитить их во время запуска
app.use((req, res, next) => {
    if (!dbReady && req.path.startsWith('/api')) {
        return res.status(503).json({ error: 'База данных не готова. Подождите...' });
    }
    next();
});

// API маршруты
app.use('/api/auth', authRoutes);
app.use('/api/tasks', taskRoutes);

// Главная страница
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'index.html'));
});

// Обработка 404
app.use((req, res) => {
    res.status(404).json({ error: 'Страница не найдена' });
});

// Обработка ошибок
app.use((err, req, res, next) => {
    console.error('Ошибка сервера:', err);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
});

// Инициализация базы данных и запуск сервера
initDatabase().then(() => {
    dbReady = true;
    console.log('База данных инициализирована');

    // Запуск сервера только после инициализации БД
    app.listen(PORT, () => {
        console.log(`Сервер запущен на http://localhost:${PORT}`);
    });
}).catch((error) => {

```

```

    console.error('Ошибка инициализации БД:', error);
    process.exit(1);
  });

const initSqlJs = require('sql.js');
const fs = require('fs-extra');
const path = require('path');

let db = null;
let dbFile = path.join(__dirname, 'app.db');

// Инициализация базы данных
async function initDatabase() {
  // Загружаем SQL.js
  const SQL = await initSqlJs();

  // Загружаем существующую БД или создаём новую
  let buffer;
  try {
    buffer = await fs.readFile(dbFile);
    db = new SQL.Database(buffer);
  } catch (error) {
    // Файл не существует, создаём новую БД
    db = new SQL.Database();
  }

  // Включаем foreign keys
  db.run('PRAGMA foreign_keys = ON');

  // Инициализация таблиц
  initTables();

  // Сохраняем БД
  saveDatabase();

  // Очистка устаревших сессий и токенов
  cleanupExpiredSessions();

  // Периодическое сохранение (каждые 5 минут)
  setInterval(() => {
    saveDatabase();
    cleanupExpiredSessions();
  }, 5 * 60 * 1000);
}

// Инициализация таблиц
function initTables() {
  // Таблица пользователей
  db.run(`
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      username TEXT UNIQUE NOT NULL,
      password_hash TEXT NOT NULL,
      role TEXT NOT NULL DEFAULT 'user',
      created_at DATETIME DEFAULT CURRENT_TIMESTAMP
    )
  `);

  // Таблица сессий
  db.run(`
    CREATE TABLE IF NOT EXISTS sessions (

```

```

        id TEXT PRIMARY KEY,
        user_id INTEGER NOT NULL,
        expires_at DATETIME NOT NULL,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
`);

// Таблица CSRF токенов
db.run(`
    CREATE TABLE IF NOT EXISTS csrf_tokens (
        token TEXT PRIMARY KEY,
        session_id TEXT NOT NULL,
        expires_at DATETIME NOT NULL,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (session_id) REFERENCES sessions(id) ON DELETE CASCADE
    )
`);

// Таблица задач
db.run(`
    CREATE TABLE IF NOT EXISTS tasks (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        title TEXT NOT NULL,
        description TEXT,
        completed INTEGER DEFAULT 0,
        created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
`);

// Создаём индексы
try {
    db.run('CREATE INDEX IF NOT EXISTS idx_sessions_user_id ON
sessions(user_id)');
    db.run('CREATE INDEX IF NOT EXISTS idx_sessions_expires_at ON
sessions(expires_at)');
    db.run('CREATE INDEX IF NOT EXISTS idx_csrf_session_id ON
csrf_tokens(session_id)');
    db.run('CREATE INDEX IF NOT EXISTS idx_tasks_user_id ON tasks(user_id)');
} catch (e) {
    // Индексы могут уже существовать
}

// Сохранение базы данных в файл
function saveDatabase() {
    if (!db) return;
    try {
        const data = db.export();
        const buffer = Buffer.from(data);
        fs.writeFileSync(dbFile, buffer);
    } catch (error) {
        console.error('Ошибка сохранения БД:', error);
    }
}

// Очистка устаревших сессий и токенов
function cleanupExpiredSessions() {

```

```

if (!db) return;
const now = new Date().toISOString();
try {
  const stmt1 = db.prepare('DELETE FROM csrf_tokens WHERE expires_at < ?');
  stmt1.bind([now]);
  stmt1.step();
  stmt1.free();

  const stmt2 = db.prepare('DELETE FROM sessions WHERE expires_at < ?');
  stmt2.bind([now]);
  stmt2.step();
  stmt2.free();

  saveDatabase();
} catch (e) {
  // Игнорируем ошибки при очистке
  console.error('Ошибка очистки сессий:', e);
}

// Обёртка для совместимости с better-sqlite3 API
const dbWrapper = {
  prepare: (sql) => {
    if (!db) {
      throw new Error('База данных не инициализирована');
    }

    const stmt = db.prepare(sql);

    return {
      run: (...params) => {
        try {
          if (params.length > 0) {
            // Фильтруем undefined значения
            const filteredParams = params.filter(p => p !== undefined);
            if (filteredParams.length > 0) {
              stmt.bind(filteredParams);
            }
          }
          stmt.step();

          // Получаем lastInsertRowid сразу после выполнения
          let lastInsertRowid = 0;
          try {
            const lastIdStmt = db.prepare('SELECT last_insert_rowid() as id');
            if (lastIdStmt.step()) {
              const row = lastIdStmt.getAsObject({});
              lastInsertRowid = parseInt(row.id) || 0;
            }
            lastIdStmt.free();
          } catch (e) {
            // Игнорируем ошибки получения lastInsertRowid
            console.error('Ошибка получения lastInsertRowid:', e);
          }

          stmt.reset();
          saveDatabase();

          return {
            lastInsertRowid: lastInsertRowid,
            changes: 1 // sql.js не возвращает точное количество изменений
          };
        }
      }
    };
  }
};

```

```

    };
  } catch (error) {
    stmt.reset();
    console.error('Ошибка выполнения запроса:', error, 'SQL:', sql,
'Params:', params);
    throw error;
  }
},
get: (...params) => {
  try {
    if (params.length > 0) {
      // Фильтруем undefined значения
      const filteredParams = params.filter(p => p !== undefined);
      if (filteredParams.length > 0) {
        stmt.bind(filteredParams);
      }
    }
    const hasRow = stmt.step();
    if (!hasRow) {
      stmt.reset();
      return null;
    }

    // Получаем имена колонок
    const columnNames = stmt.getColumnNames();
    // Получаем значения
    const values = stmt.get();

    // Создаём объект из колонок и значений
    const result = {};
    for (let i = 0; i < columnNames.length; i++) {
      result[columnNames[i]] = values[i];
    }

    stmt.reset();
    return result;
  } catch (error) {
    stmt.reset();
    console.error('Ошибка получения данных:', error, 'SQL:', sql,
'Params:', params);
    throw error;
  }
},
all: (...params) => {
  try {
    if (params.length > 0) {
      // Фильтруем undefined значения
      const filteredParams = params.filter(p => p !== undefined);
      if (filteredParams.length > 0) {
        stmt.bind(filteredParams);
      }
    }

    // Получаем имена колонок один раз
    const columnNames = stmt.getColumnNames();
    const results = [];

    while (stmt.step()) {
      // Получаем значения
      const values = stmt.get();
      // Создаём объект из колонок и значений

```

```

        const row = {};
        for (let i = 0; i < columnNames.length; i++) {
            row[columnNames[i]] = values[i];
        }
        results.push(row);
    }

    stmt.reset();
    return results;
} catch (error) {
    stmt.reset();
    console.error('Ошибка получения всех данных:', error, 'SQL:', sql,
'Params:', params);
    throw error;
}
}
};

},
exec: (sql) => {
    if (!db) {
        throw new Error('База данных не инициализирована');
    }
    db.run(sql);
    saveDatabase();
}
};

// Экспортируем обёртку и функцию инициализации
module.exports = {
    get db() {
        return dbWrapper;
    },
    initDatabase,
    saveDatabase
};

const initSqlJs = require('sql.js');
const fs = require('fs-extra');
const path = require('path');

let db = null;
let dbFile = path.join(__dirname, 'app.db');

// Инициализация базы данных
async function initDatabase() {
    // Загружаем SQL.js
    const SQL = await initSqlJs();

    // Загружаем существующую БД или создаём новую
    let buffer;
    try {
        buffer = await fs.readFile(dbFile);
        db = new SQL.Database(buffer);
    } catch (error) {
        // Файл не существует, создаём новую БД
        db = new SQL.Database();
    }

    // Включаем foreign keys
    db.run('PRAGMA foreign_keys = ON');

```

```

// Инициализация таблиц
initTables();

// Сохраняем БД
saveDatabase();

// Очистка устаревших сессий и токенов
cleanupExpiredSessions();

// Периодическое сохранение (каждые 5 минут)
setInterval(() => {
    saveDatabase();
    cleanupExpiredSessions();
}, 5 * 60 * 1000);
}

// Инициализация таблиц
function initTables() {
    // Таблица пользователей
    db.run(`
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT UNIQUE NOT NULL,
            password_hash TEXT NOT NULL,
            role TEXT NOT NULL DEFAULT 'user',
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP
        )
    `);

    // Таблица сессий
    db.run(`
        CREATE TABLE IF NOT EXISTS sessions (
            id TEXT PRIMARY KEY,
            user_id INTEGER NOT NULL,
            expires_at DATETIME NOT NULL,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
        )
    `);

    // Таблица CSRF токенов
    db.run(`
        CREATE TABLE IF NOT EXISTS csrf_tokens (
            token TEXT PRIMARY KEY,
            session_id TEXT NOT NULL,
            expires_at DATETIME NOT NULL,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY (session_id) REFERENCES sessions(id) ON DELETE CASCADE
        )
    `);

    // Таблица задач
    db.run(`
        CREATE TABLE IF NOT EXISTS tasks (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            user_id INTEGER NOT NULL,
            title TEXT NOT NULL,
            description TEXT,
            completed INTEGER DEFAULT 0,
            created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
            updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,

```

```

        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
`);

// Создаём индексы
try {
    db.run('CREATE INDEX IF NOT EXISTS idx_sessions_user_id ON
sessions(user_id)');
    db.run('CREATE INDEX IF NOT EXISTS idx_sessions_expires_at ON
sessions(expires_at)');
    db.run('CREATE INDEX IF NOT EXISTS idx_csrf_session_id ON
csrf_tokens(session_id)');
    db.run('CREATE INDEX IF NOT EXISTS idx_tasks_user_id ON tasks(user_id)');
} catch (e) {
    // Индексы могут уже существовать
}

// Сохранение базы данных в файл
function saveDatabase() {
    if (!db) return;
    try {
        const data = db.export();
        const buffer = Buffer.from(data);
        fs.writeFileSync(dbFile, buffer);
    } catch (error) {
        console.error('Ошибка сохранения БД:', error);
    }
}

// Очистка устаревших сессий и токенов
function cleanupExpiredSessions() {
    if (!db) return;
    const now = new Date().toISOString();
    try {
        const stmt1 = db.prepare('DELETE FROM csrf_tokens WHERE expires_at < ?');
        stmt1.bind([now]);
        stmt1.step();
        stmt1.free();

        const stmt2 = db.prepare('DELETE FROM sessions WHERE expires_at < ?');
        stmt2.bind([now]);
        stmt2.step();
        stmt2.free();

        saveDatabase();
    } catch (e) {
        // Игнорируем ошибки при очистке
        console.error('Ошибка очистки сессий:', e);
    }
}

// Обёртка для совместимости с better-sqlite3 API
const dbWrapper = {
    prepare: (sql) => {
        if (!db) {
            throw new Error('База данных не инициализирована');
        }

        const stmt = db.prepare(sql);

```

```

return {
  run: (...params) => {
    try {
      if (params.length > 0) {
        // Фильтруем undefined значения
        const filteredParams = params.filter(p => p !== undefined);
        if (filteredParams.length > 0) {
          stmt.bind(filteredParams);
        }
      }
      stmt.step();

      // Получаем lastInsertRowid сразу после выполнения
      let lastInsertRowid = 0;
      try {
        const lastIdStmt = db.prepare('SELECT last_insert_rowid() as id');
        if (lastIdStmt.step()) {
          const row = lastIdStmt.getAsObject({});
          lastInsertRowid = parseInt(row.id) || 0;
        }
        lastIdStmt.free();
      } catch (e) {
        // Игнорируем ошибки получения lastInsertRowid
        console.error('Ошибка получения lastInsertRowid:', e);
      }

      stmt.reset();
      saveDatabase();

      return {
        lastInsertRowid: lastInsertRowid,
        changes: 1 // sql.js не возвращает точное количество изменений
      };
    } catch (error) {
      stmt.reset();
      console.error('Ошибка выполнения запроса:', error, 'SQL:', sql,
'Params:', params);
      throw error;
    }
  },
  get: (...params) => {
    try {
      if (params.length > 0) {
        // Фильтруем undefined значения
        const filteredParams = params.filter(p => p !== undefined);
        if (filteredParams.length > 0) {
          stmt.bind(filteredParams);
        }
      }
      const hasRow = stmt.step();
      if (!hasRow) {
        stmt.reset();
        return null;
      }

      // Получаем имена колонок
      const columnNames = stmt.getColumnNames();
      // Получаем значения
      const values = stmt.get();

      // Создаём объект из колонок и значений

```

```

        const result = {};
        for (let i = 0; i < columnNames.length; i++) {
            result[columnNames[i]] = values[i];
        }

        stmt.reset();
        return result;
    } catch (error) {
        stmt.reset();
        console.error('Ошибка получения данных:', error, 'SQL:', sql,
'Params:', params);
        throw error;
    }
},
all: (...params) => {
    try {
        if (params.length > 0) {
            // Фильтруем undefined значения
            const filteredParams = params.filter(p => p !== undefined);
            if (filteredParams.length > 0) {
                stmt.bind(filteredParams);
            }
        }

        // Получаем имена колонок один раз
        const columnNames = stmt.getColumnNames();
        const results = [];

        while (stmt.step()) {
            // Получаем значения
            const values = stmt.get();
            // Создаём объект из колонок и значений
            const row = {};
            for (let i = 0; i < columnNames.length; i++) {
                row[columnNames[i]] = values[i];
            }
            results.push(row);
        }

        stmt.reset();
        return results;
    } catch (error) {
        stmt.reset();
        console.error('Ошибка получения всех данных:', error, 'SQL:', sql,
'Params:', params);
        throw error;
    }
}
};

exec: (sql) => {
    if (!db) {
        throw new Error('База данных не инициализирована');
    }
    db.run(sql);
    saveDatabase();
}
};

// Экспортируем обёртку и функцию инициализации
module.exports = {

```

```

    get db() {
      return dbWrapper;
    },
    initDatabase,
    saveDatabase
  };

// routes/tasks.js
const express = require('express');
const { db } = require('../database');
const { requireAuth, requireRole } = require('../security/session');
const { csrfProtection } = require('../security/csrf');
const { validateTask, validateId } = require('../security/validation');

const router = express.Router();

// Все маршруты требуют аутентификации
router.use(requireAuth);
router.use(csrfProtection);

/**
 * Получение всех задач пользователя
 */
router.get('/', (req, res) => {
  try {
    const tasks = db.prepare(`
      SELECT id, title, description, completed, created_at, updated_at
      FROM tasks
      WHERE user_id = ?
      ORDER BY created_at DESC
    `).all(req.user.id);

    res.json({ tasks });
  } catch (error) {
    console.error('Ошибка получения задач:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

/**
 * Получение всех задач (только для администраторов)
 * Должен быть определён ПЕРЕД /:id, чтобы Express правильно обрабатывал маршрут
 */
router.get('/admin/all', requireRole('admin'), (req, res) => {
  try {
    const usernameFilter = req.query.username || '';
    const taskTitleFilter = req.query.task || '';

    let query = `
      SELECT    t.id,    t.title,    t.description,    t.completed,    t.created_at,
      t.updated_at,
                u.username, u.id as user_id
      FROM tasks t
      JOIN users u ON t.user_id = u.id
      WHERE 1=1
    `;
    const params = [];

    // Фильтр по имени пользователя
    if (usernameFilter.trim()) {
      query += ' AND u.username LIKE ?';
    }
  }
});

```

```

    params.push(`%${usernameFilter.trim()}%`);
  }

  // Фильтр по названию задачи
  if (taskTitleFilter.trim()) {
    query += ' AND t.title LIKE ?';
    params.push(`%${taskTitleFilter.trim()}%`);
  }

  query += ' ORDER BY t.created_at DESC';

  const tasks = db.prepare(query).all(...params);

  res.json({ tasks });
} catch (error) {
  console.error('Ошибка получения всех задач:', error);
  res.status(500).json({ error: 'Внутренняя ошибка сервера' });
}
});

/**
 * Получение задачи по ID
 */
router.get('/:id', (req, res) => {
  try {
    const idValidation = validateId(req.params.id);
    if (!idValidation.valid) {
      return res.status(400).json({ error: idValidation.error });
    }

    // Получаем задачу с информацией о владельце
    const task = db.prepare(`
      SELECT id, title, description, completed, created_at, updated_at, user_id
      FROM tasks
      WHERE id = ?
    `).get(idValidation.value);

    if (!task) {
      return res.status(404).json({ error: 'Задача не найдена' });
    }

    // Проверка прав: администратор может просматривать любые задачи, обычный
    // пользователь - только свои
    if (req.user.role !== 'admin' && task.user_id !== req.user.id) {
      return res.status(403).json({ error: 'Недостаточно прав для просмотра этой
задачи' });
    }

    // Удаляем user_id из ответа (не нужен клиенту)
    delete task.user_id;

    res.json({ task });
  } catch (error) {
    console.error('Ошибка получения задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

/**
 * Создание новой задачи
 */

```

```

router.post('/', (req, res) => {
  try {
    const taskValidation = validateTask(req.body.title, req.body.description);
    if (!taskValidation.valid) {
      return res.status(400).json({ error: taskValidation.error });
    }

    const { title, description } = taskValidation.value;

    // Получаем статус выполнения (если передан)
    const completed = req.body.completed ? 1 : 0;

    const result = db.prepare(`
      INSERT INTO tasks (user_id, title, description, completed)
      VALUES (?, ?, ?, ?)
    `).run(req.user.id, title, description, completed);

    const task = db.prepare(`
      SELECT id, title, description, completed, created_at, updated_at
      FROM tasks
      WHERE id = ?
    `).get(result.lastInsertRowid);

    res.status(201).json({ task });
  } catch (error) {
    console.error('Ошибка создания задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

/**
 * Обновление задачи
 */
router.put('/:id', (req, res) => {
  try {
    const idValidation = validateId(req.params.id);
    if (!idValidation.valid) {
      return res.status(400).json({ error: idValidation.error });
    }

    // Проверка существования задачи
    const existing = db.prepare('SELECT id, user_id FROM tasks WHERE id = ?').get(idValidation.value);

    if (!existing) {
      return res.status(404).json({ error: 'Задача не найдена' });
    }

    // Проверка прав: администратор может редактировать любые задачи, обычный
    // пользователь - только свои
    if (req.user.role !== 'admin' && existing.user_id !== req.user.id) {
      return res.status(403).json({ error: 'Недостаточно прав для редактирования этой задачи' });
    }

    // Валидация обновляемых данных
    const updates = {};
    if (req.body.title !== undefined) {
      const taskValidation = validateTask(req.body.title, req.body.description);
      if (!taskValidation.valid) {
        return res.status(400).json({ error: taskValidation.error });
      }
    }
  }
});

```

```

    }
    updates.title = taskValidation.value.title;
    updates.description = taskValidation.value.description;
  }

  if (req.body.completed !== undefined) {
    updates.completed = req.body.completed ? 1 : 0;
  }

  // Проверка, что есть хотя бы одно поле для обновления
  if (Object.keys(updates).length === 0) {
    return res.status(400).json({ error: 'Не указаны поля для обновления' });
  }

  // Обновление задачи (для администратора не проверяем user_id в WHERE)
  const setClause = Object.keys(updates).map(key => `${key} = ?`).join(', ');
  const values = [...Object.values(updates), idValidation.value];

  db.prepare(`
    UPDATE tasks
    SET ${setClause}, updated_at = CURRENT_TIMESTAMP
    WHERE id = ?
  `).run(...values);

  const task = db.prepare(`
    SELECT id, title, description, completed, created_at, updated_at
    FROM tasks
    WHERE id = ?
  `).get(idValidation.value);

  res.json({ task });
} catch (error) {
  console.error('Ошибка обновления задачи:', error);
  res.status(500).json({ error: 'Внутренняя ошибка сервера' });
}
});

/**
 * Удаление задачи
 */
router.delete('/:id', (req, res) => {
  try {
    const idValidation = validateId(req.params.id);
    if (!idValidation.valid) {
      return res.status(400).json({ error: idValidation.error });
    }

    // Проверка существования задачи
    const task = db.prepare('SELECT id, user_id FROM tasks WHERE id = ?').get(idValidation.value);

    if (!task) {
      return res.status(404).json({ error: 'Задача не найдена' });
    }

    // Проверка прав: администратор может удалять любые задачи, обычный
    // пользователь - только свои
    if (req.user.role !== 'admin' && task.user_id !== req.user.id) {
      return res.status(403).json({ error: 'Недостаточно прав для удаления этой задачи' });
    }
  }
});

```

```

    // Удаление задачи
    const result = db.prepare('DELETE FROM tasks WHERE id = ?').run(idValidation.value);

    if (result.changes === 0) {
        return res.status(404).json({ error: 'Задача не найдена' });
    }

    res.json({ message: 'Задача удалена' });
} catch (error) {
    console.error('Ошибка удаления задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
}
});

module.exports = router;

// routes/tasks.js
const express = require('express');
const { db } = require('../database');
const { requireAuth, requireRole } = require('../security/session');
const { csrfProtection } = require('../security/csrf');
const { validateTask, validateId } = require('../security/validation');

const router = express.Router();

// Все маршруты требуют аутентификации
router.use(requireAuth);
router.use(csrfProtection);

/**
 * Получение всех задач пользователя
 */
router.get('/', (req, res) => {
    try {
        const tasks = db.prepare(`
            SELECT id, title, description, completed, created_at, updated_at
            FROM tasks
            WHERE user_id = ?
            ORDER BY created_at DESC
        `).all(req.user.id);

        res.json({ tasks });
    } catch (error) {
        console.error('Ошибка получения задач:', error);
        res.status(500).json({ error: 'Внутренняя ошибка сервера' });
    }
});

/**
 * Получение всех задач (только для администраторов)
 * Должен быть определён ПЕРЕД /:id, чтобы Express правильно обрабатывал маршрут
 */
router.get('/admin/all', requireRole('admin'), (req, res) => {
    try {
        const usernameFilter = req.query.username || '';
        const taskTitleFilter = req.query.task || '';

        let query = `

```

```

        SELECT    t.id,    t.title,    t.description,    t.completed,    t.created_at,
t.updated_at,
        u.username, u.id as user_id
    FROM tasks t
    JOIN users u ON t.user_id = u.id
    WHERE 1=1
`;
const params = [];

// Фильтр по имени пользователя
if (usernameFilter.trim()) {
    query += ' AND u.username LIKE ?';
    params.push(`%${usernameFilter.trim()}%`);
}

// Фильтр по названию задачи
if (taskTitleFilter.trim()) {
    query += ' AND t.title LIKE ?';
    params.push(`%${taskTitleFilter.trim()}%`);
}

query += ' ORDER BY t.created_at DESC';

const tasks = db.prepare(query).all(...params);

res.json({ tasks });
} catch (error) {
    console.error('Ошибка получения всех задач:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
}
});

/**
 * Получение задачи по ID
 */
router.get('/:id', (req, res) => {
    try {
        const idValidation = validateId(req.params.id);
        if (!idValidation.valid) {
            return res.status(400).json({ error: idValidation.error });
        }

        // Получаем задачу с информацией о владельце
        const task = db.prepare(`
            SELECT id, title, description, completed, created_at, updated_at, user_id
            FROM tasks
            WHERE id = ?
        `).get(idValidation.value);

        if (!task) {
            return res.status(404).json({ error: 'Задача не найдена' });
        }

        // Проверка прав: администратор может просматривать любые задачи, обычный
        // пользователь - только свои
        if (req.user.role !== 'admin' && task.user_id !== req.user.id) {
            return res.status(403).json({ error: 'Недостаточно прав для просмотра этой
задачи' });
        }

        // Удаляем user_id из ответа (не нужен клиенту)

```

```

    delete task.user_id;

    res.json({ task });
  } catch (error) {
    console.error('Ошибка получения задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

/**
 * Создание новой задачи
 */
router.post('/', (req, res) => {
  try {
    const taskValidation = validateTask(req.body.title, req.body.description);
    if (!taskValidation.valid) {
      return res.status(400).json({ error: taskValidation.error });
    }

    const { title, description } = taskValidation.value;

    // Получаем статус выполнения (если передан)
    const completed = req.body.completed ? 1 : 0;

    const result = db.prepare(`
      INSERT INTO tasks (user_id, title, description, completed)
      VALUES (?, ?, ?, ?)
    `).run(req.user.id, title, description, completed);

    const task = db.prepare(`
      SELECT id, title, description, completed, created_at, updated_at
      FROM tasks
      WHERE id = ?
    `).get(result.lastInsertRowid);

    res.status(201).json({ task });
  } catch (error) {
    console.error('Ошибка создания задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
  }
});

/**
 * Обновление задачи
 */
router.put('/:id', (req, res) => {
  try {
    const idValidation = validateId(req.params.id);
    if (!idValidation.valid) {
      return res.status(400).json({ error: idValidation.error });
    }

    // Проверка существования задачи
    const existing = db.prepare('SELECT id, user_id FROM tasks WHERE id = ?').get(idValidation.value);

    if (!existing) {
      return res.status(404).json({ error: 'Задача не найдена' });
    }
  }
});

```

```

    // Проверка прав: администратор может редактировать любые задачи, обычный
    пользователь - только свои
    if (req.user.role !== 'admin' && existing.user_id !== req.user.id) {
        return res.status(403).json({ error: 'Недостаточно прав для редактирования
этой задачи' });
    }

    // Валидация обновляемых данных
    const updates = {};
    if (req.body.title !== undefined) {
        const taskValidation = validateTask(req.body.title, req.body.description);
        if (!taskValidation.valid) {
            return res.status(400).json({ error: taskValidation.error });
        }
        updates.title = taskValidation.value.title;
        updates.description = taskValidation.value.description;
    }

    if (req.body.completed !== undefined) {
        updates.completed = req.body.completed ? 1 : 0;
    }

    // Проверка, что есть хотя бы одно поле для обновления
    if (Object.keys(updates).length === 0) {
        return res.status(400).json({ error: 'Не указаны поля для обновления' });
    }

    // Обновление задачи (для администратора не проверяем user_id в WHERE)
    const setClause = Object.keys(updates).map(key => `${key} = ?`).join(', ');
    const values = [...Object.values(updates), idValidation.value];

    db.prepare(`
        UPDATE tasks
        SET ${setClause}, updated_at = CURRENT_TIMESTAMP
        WHERE id = ?
    `).run(...values);

    const task = db.prepare(`
        SELECT id, title, description, completed, created_at, updated_at
        FROM tasks
        WHERE id = ?
    `).get(idValidation.value);

    res.json({ task });
} catch (error) {
    console.error('Ошибка обновления задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
}
});

/**
 * Удаление задачи
 */
router.delete('/:id', (req, res) => {
    try {
        const idValidation = validateId(req.params.id);
        if (!idValidation.valid) {
            return res.status(400).json({ error: idValidation.error });
        }

        // Проверка существования задачи

```

```

    const task = db.prepare('SELECT id, user_id FROM tasks WHERE id = ?').get(idValidation.value);

    if (!task) {
        return res.status(404).json({ error: 'Задача не найдена' });
    }

    // Проверка прав: администратор может удалять любые задачи, обычный
    // пользователь - только свои
    if (req.user.role !== 'admin' && task.user_id !== req.user.id) {
        return res.status(403).json({ error: 'Недостаточно прав для удаления этой задачи' });
    }

    // Удаление задачи
    const result = db.prepare('DELETE FROM tasks WHERE id = ?').run(idValidation.value);

    if (result.changes === 0) {
        return res.status(404).json({ error: 'Задача не найдена' });
    }

    res.json({ message: 'Задача удалена' });
} catch (error) {
    console.error('Ошибка удаления задачи:', error);
    res.status(500).json({ error: 'Внутренняя ошибка сервера' });
}
});

module.exports = router;

// security/csrf.js
const crypto = require('crypto');
const { db } = require('../database');

// Время жизни CSRF токена (1 час)
const CSRF_TOKEN_DURATION = 60 * 60 * 1000;

/**
 * Генерация CSRF токена
 */
function generateToken() {
    return crypto.randomBytes(32).toString('hex');
}

/**
 * Создание и сохранение CSRF токена для сессии
 */
function createToken(sessionId) {
    const token = generateToken();
    const expiresAt = new Date(Date.now() + CSRF_TOKEN_DURATION).toISOString();

    db.prepare(`
        INSERT INTO csrf_tokens (token, session_id, expires_at)
        VALUES (?, ?, ?)
    `).run(token, sessionId, expiresAt);

    return token;
}

/**

```

```

* Проверка CSRF токена
*/
function verifyToken(token, sessionId) {
  if (!token || !sessionId) return false;

  const stored = db.prepare(`
    SELECT * FROM csrf_tokens
    WHERE token = ? AND session_id = ? AND expires_at > datetime('now')
  `).get(token, sessionId);

  return !!stored;
}

/**
* Удаление использованного токена (опционально, для одноразовых токенов)
*/
function deleteToken(token) {
  db.prepare('DELETE FROM csrf_tokens WHERE token = ?').run(token);
}

/**
* Middleware для проверки CSRF токена
* Для GET запросов токен не требуется, для POST/PUT/DELETE - обязателен
*/
function csrfProtection(req, res, next) {
  // GET запросы не требуют CSRF защиты
  if (req.method === 'GET' || req.method === 'HEAD' || req.method === 'OPTIONS')
  {
    return next();
  }

  // Для аутентифицированных пользователей проверяем токен
  if (req.user) {
    const token = req.headers['x-csrf-token'] || req.body?.csrf_token;

    if (!token) {
      return res.status(403).json({ error: 'CSRF токен отсутствует' });
    }

    if (!verifyToken(token, req.user.sessionId)) {
      return res.status(403).json({ error: 'Недействительный CSRF токен' });
    }

    // Удаляем использованный токен (одноразовый)
    deleteToken(token);
  }

  next();
}

module.exports = {
  createToken,
  verifyToken,
  deleteToken,
  csrfProtection
};

// security/validation.js
/**
* Валидация и санитизация входных данных
*/

```

```

/**
 * Проверка username
 */
function validateUsername(username) {
  if (!username || typeof username !== 'string') {
    return { valid: false, error: 'Имя пользователя обязательно' };
  }

  const trimmed = username.trim();

  if (trimmed.length < 3 || trimmed.length > 20) {
    return { valid: false, error: 'Имя пользователя должно быть от 3 до 20 символов' };
  }

  if (!/^[a-zA-Z0-9_]+$/.test(trimmed)) {
    return { valid: false, error: 'Имя пользователя может содержать только буквы, цифры и подчёркивание' };
  }

  return { valid: true, value: trimmed };
}

/**
 * Проверка password
 */
function validatePassword(password) {
  if (!password || typeof password !== 'string') {
    return { valid: false, error: 'Пароль обязателен' };
  }

  if (password.length < 6) {
    return { valid: false, error: 'Пароль должен содержать минимум 6 символов' };
  }

  if (password.length > 100) {
    return { valid: false, error: 'Пароль слишком длинный' };
  }

  return { valid: true, value: password };
}

/**
 * Санитизация строки (защита от XSS)
 */
function sanitizeString(str) {
  if (typeof str !== 'string') return '';

  return str
    .replace(/&/g, '&amp;')
    .replace(/</g, '&lt;')
    .replace(/>/g, '&gt;')
    .replace(/"/g, '&quot;')
    .replace(/'/g, '&#x27;')
    .replace(/\\/g, '&#x2F;')
    .trim();
}

/**
 * Санитизация для HTML (менее агрессивная, для отображения)

```

```

*/
function sanitizeHTML(str) {
  if (typeof str !== 'string') return '';

  // Разрешаем только безопасные символы
  return str
    .replace(/[<>]/g, '') // Удаляем угловые скобки
    .trim();
}

/**
 * Валидация задачи
 */
function validateTask(title, description) {
  if (!title || typeof title !== 'string') {
    return { valid: false, error: 'Название задачи обязательно' };
  }

  const sanitizedTitle = sanitizeHTML(title.trim());

  if (sanitizedTitle.length === 0) {
    return { valid: false, error: 'Название задачи не может быть пустым' };
  }

  if (sanitizedTitle.length > 200) {
    return { valid: false, error: 'Название задачи слишком длинное' };
  }

  const sanitizedDesc = description ? sanitizeHTML(description.trim()) : '';

  if (sanitizedDesc.length > 1000) {
    return { valid: false, error: 'Описание задачи слишком длинное' };
  }

  return {
    valid: true,
    value: {
      title: sanitizedTitle,
      description: sanitizedDesc
    }
  };
}

/**
 * Защита от SQL injection через параметризованные запросы
 * (better-sqlite3 уже использует параметризованные запросы, но для явности)
 */
function validateId(id) {
  const numId = parseInt(id, 10);
  if (isNaN(numId) || numId <= 0) {
    return { valid: false, error: 'Недействительный ID' };
  }
  return { valid: true, value: numId };
}

module.exports = {
  validateUsername,
  validatePassword,
  sanitizeString,
  sanitizeHTML,
  validateTask,

```

```
    validateId  
};
```