Geekbrains

**Разработка веб-приложения для управления задачами и планирования**

**проектов** «**ToDo GeekBrains»**

Специализация: **Fullstack разработчик**

ФИО: Копалейшвили Артём Георгиевич

Город Москва

Год 2025

**Содержание**

[**Введение**](#_Введение)

[**Глава 1. Теоретические основы разработки веб-приложений для управления проектами**](#_Глава_1._Теоретические)

1.1. Обзор веб-приложений для управления задачами и проектами: назначение, типы, функциональные возможности.

1.2. Архитектура веб-приложений: клиент-серверное взаимодействие, frontend и backend.

1.3. Основные технологии и инструменты Fullstack-разработки:

1.3.1. Python и Flask: обзор, преимущества использования для backend.

1.3.2. HTML, CSS, JavaScript, Bootstrap, jQuery, jQuery UI: обзор, применение во frontend.

1.3.3. JSON как формат обмена данными.

1.3.4. Системы контроля версий (Git).

* 1. . Паттерны проектирования, используемые в приложении.
     1. Модель MVC

[**Глава 2. Проектирование и реализация веб-приложения ToDo GeekBrains**](#_Глава_2._Проектирование)

2.1. Постановка задачи и определение функциональных требований:

2.1.1. Аутентификация и авторизация пользователей (вход, выход, разграничение прав доступа).

2.1.2. Управление проектами: создание, удаление, просмотр списка проектов.

2.1.3. Управление задачами: создание, удаление, редактирование статуса, назначение исполнителя, просмотр задач в рамках проекта.

2.1.4. Канбан-доска.

2.1.5. Комментирование задач.

2.1.6 Управление пользователями(CRUD). Только для администратора.

2.2. Проектирование структуры данных:

2.2.1. Модель данных "Пользователь" (User): атрибуты (id, username, password).

2.2.2. Модель данных "Проект" (Project): атрибуты (id, name, tasks).

2.2.3. Модель данных "Задача" (Task): атрибуты (id, title, description, status, assigned\_to, comments).

* + 1. Взаимосвязи между моделями.

2.3. Реализация backend (серверной части) на Python с использованием Flask:

2.3.1. Создание API-маршрутов (endpoints) для обработки запросов:

/login (POST): аутентификация пользователя.

/logout (GET): выход из системы.

/projects (GET): получение списка проектов.

/projects/add (POST): создание нового проекта.

/projects/delete/<project\_id> (GET): удаление проекта.

/project/<project\_id> (GET): отображение канбан-доски проекта.

/project/<project\_id>/task/add (POST): добавление задачи в проект.

/project/<project\_id>/task/<task\_id>/update/<new\_status> (GET): обновление статуса задачи.

/project/<project\_id>/task/<task\_id>/delete (GET): удаление задачи.

/admin/users (GET, POST): управление пользователями (для администратора).

/admin/users/delete/<user\_id> (GET): удаление пользователя (для администратора).

/project/<project\_id>/task/<task\_id>/comments (GET, POST): Загрузка и добавление комментариев.

2.3.2. Использование декораторов @login\_required и @admin\_required для защиты маршрутов.

2.3.3. Взаимодействие с хранилищем данных (FileStorage и UserStorage).

* + 1. Сериализация данных в JSON.

2.4. Реализация frontend (клиентской части) с использованием HTML, CSS, JavaScript, Bootstrap, jQuery:

2.4.1. Разработка структуры HTML-страниц с использованием шаблонизатора Jinja2 (шаблоны base.html, login.html, project\_list.html, project\_board.html, user\_management.html).

2.4.2. Стилизация интерфейса с помощью Bootstrap и собственных CSS-стилей.

2.4.3. Использование JavaScript и jQuery для:

Отправки AJAX-запросов на сервер.

Динамического обновления содержимого страниц без перезагрузки.

Реализации drag-and-drop функциональности для канбан-доски (jQuery UI).

Отображения уведомлений (всплывающих сообщений).

Управления модальными окнами.

Валидации форм на стороне клиента.

2.5. Взаимодействие frontend и backend:

2.5.1. Обмен данными в формате JSON.

2.5.2. Обработка ответов сервера и обновление интерфейса.

2.6. Описание классов, функций и методов.

[**Глава 3. Тестирование, отладка и развертывание приложения**](#_Глава_3._Тестирование,)

3.1. Тестирование:

3.1.1. Функциональное тестирование: проверка корректности работы всех функций приложения.

3.1.2 Тестирование API

3.1.3. Тестирование пользовательского интерфейса (UI-тестирование):

3.1.4. Тестирование безопасности: проверка защиты от несанкционированного доступа.

3.2. Отладка приложения:

3.2.1. Использование режима отладки Flask (debug=True).

3.2.2. Обработка ошибок и исключений.

3.3. Развертывание приложения (краткое описание возможных вариантов, без детальной инструкции):

3.3.1. Локальный запуск (для разработки и тестирования).

3.3.2. Развертывание на облачной платформе (например, Heroku, PythonAnywhere, AWS).

[**Заключение**](#_Заключение)

[**Список используемой литературы**](#_Список_используемой_литературы)

**[Приложения](#_Приложения)**

# Введение

Эффективное управление задачами и проектами – ключевой фактор успеха как для отдельных специалистов, так и для команд, работающих над общими целями. Неструктурированный подход к планированию, отсутствие четкого распределения обязанностей и контроля за выполнением задач приводят к срыву сроков, снижению продуктивности и качества работы. Традиционные методы, такие как ведение записей в блокнотах или использование электронных таблиц, часто оказываются недостаточно гибкими и масштабируемыми, особенно при росте числа задач и участников проекта. Поэтому существует острая потребность в специализированных инструментах, обеспечивающих комплексный подход к управлению проектами.

Эта потребность и определяет актуальность темы дипломного проекта. Разработка веб-приложения для управления задачами (task manager/to-do application) позволяет создать решение, которое централизует информацию, упрощает планирование, обеспечивает прозрачность процессов и способствует эффективной командной работе.

**Цель дипломного проекта:** Разработка веб-приложения для управления задачами и проектами, предоставляющего пользователям инструменты для организации, контроля и совместной работы над задачами в рамках проектов. Приложение должно быть интуитивно понятным, масштабируемым и обеспечивать базовый, но достаточный набор функций для эффективного управления задачами.

**Ключевые функции приложения (реализованные в проекте):**

* Управление проектами: Возможность создания, редактирования (в коде есть функции добавления, но не редактирования) и удаления проектов (класс Project в models/project.py, функции add\_project и delete\_project в web\_app.py, файл projects.json для хранения данных). Каждый проект представляет собой отдельную сущность со своим набором задач.
* Управление задачами внутри проектов: Создание, удаление задач, назначение исполнителей и изменение статуса задач (класс Task в models/task.py, функции add\_task, update\_task\_status, delete\_task в web\_app.py, файл projects.json). Для визуализации задач используется канбан-доска, разделяющая задачи по статусам ("Планы", "В работе", "Тестирование", "Завершено") (шаблон project\_board.html).
* Разграничение доступа: Реализована базовая система аутентификации и авторизации пользователей (класс User в models/user.py, функции login, logout, декораторы login\_required и admin\_required в web\_app.py, файл users.json). Поддерживаются роли "администратор" и "пользователь", с разным уровнем доступа к функциональности (например, управление пользователями доступно только администратору).
* Комментирование задач: Возможность добавления комментариев к задачам для обсуждения деталей и хода выполнения (метод add\_comment в классе Task, функция handle\_comments в web\_app.py).
* Визуализация задач: Отображение задач в виде канбан-доски с разделением по статусам для наглядного представления прогресса выполнения проекта (шаблон project\_board.html, использующий jQuery UI для drag-and-drop функциональности).
* Хранение данных: JSON файлы, так как в коде есть функции load\_projects и save\_projects в storage/file\_storage.py, а также,load\_tasks и save\_tasks в storage/file\_storage.py, а также load\_users, save\_users в storage/file\_storage\_users.py

**Задачи дипломного проекта:** Анализ предметной области: Изучение существующих решений для управления задачами, определение их сильных и слабых сторон, формирование требований к разрабатываемому приложению. (В текущем введении это отражено косвенно, можно добавить раздел с кратким обзором аналогов).

**Проектирование архитектуры приложения:** Определение структуры приложения, выбор технологий и инструментов разработки, проектирование модели данных (это отражено в структуре директорий и файлах моделей).

**Реализация серверной части (Backend):** Разработка API для взаимодействия с данными (проектами, задачами, пользователями) с использованием языка Python и фреймворка Flask (файл web\_app.py).

**Реализация клиентской части (Frontend):** Создание пользовательского интерфейса с использованием HTML, CSS, JavaScript и библиотек (Bootstrap, jQuery) для обеспечения удобного взаимодействия пользователя с приложением (шаблоны в папке templates).

**Интеграция Backend и Frontend:** Настройка взаимодействия между клиентской и серверной частями приложения.

**Тестирование и отладка:** Проверка работоспособности приложения, выявление и устранение ошибок.

**Используемые технологии и инструменты:**

* Python: Основной язык программирования для серверной части приложения.
* Flask: Микрофреймворк для создания веб-приложений на Python. Обеспечивает маршрутизацию запросов, обработку данных, взаимодействие с шаблонами (используется в web\_app.py).
* HTML, CSS, JavaScript: Языки для создания структуры, оформления и интерактивности пользовательского интерфейса.
* Bootstrap: CSS-фреймворк для быстрого создания адаптивного и стильного дизайна.
* jQuery: JavaScript-библиотека, упрощающая манипуляции с DOM, обработку событий и создание анимаций (используется в шаблонах, особенно в project\_board.html для drag-and-drop).
* JSON: Формат обмена данными между клиентской и серверной частями, а также для хранения данных приложения (файлы projects.json, tasks.json, users.json).
* Git: Система контроля версий для отслеживания изменений в коде и совместной работы.
* Visual Studio Code: Интегрированная среда разработки (IDE).

**Структура дипломной работы:** (Этот раздел можно оставить без изменений, если он соответствует вашей структуре) Введение (настоящий раздел). Теоретическая часть (обзор предметной области, анализ существующих решений). Практическая часть (описание процесса разработки приложения, архитектуры, реализации отдельных компонентов). Заключение (подведение итогов, выводы, перспективы развития). Список использованной литературы. Приложения (исходный код, документация).

**Автор:** Копалейшвили Артём Георгиевич, направление Fullstack-разработка.

# Глава 1. Теоретические основы разработки веб-приложений для управления проектами

Эта глава закладывает теоретический фундамент для понимания принципов разработки веб-приложений, ориентированных на управление проектами и задачами. Рассматриваются ключевые аспекты архитектуры, технологий и паттернов проектирования, которые лежат в основе таких приложений, включая те, что используются в разработанном приложении ToDo GeekBrains.

**1.1.** Обзор веб-приложений для управления задачами и проектами: назначение, типы, функциональные возможности.

Веб-приложения для управления задачами и проектами (Task & Project Management Web Applications) представляют собой программные решения, доступные через веб-браузер, предназначенные для организации, планирования, отслеживания и координации работы над задачами и проектами. Они призваны заменить разрозненные электронные таблицы, списки дел на бумаге и электронную почту, предоставляя централизованную платформу для совместной работы.

**Назначение:**

Основное назначение таких приложений — повышение эффективности работы команд и отдельных лиц за счет:

**Централизации информации:** Все данные о проектах, задачах, сроках, исполнителях и ресурсах хранятся в едином месте, что обеспечивает прозрачность и доступность для всех участников.

**Планирования и приоритизации:** Возможность создавать задачи, устанавливать сроки, назначать исполнителей, определять зависимости между задачами и расставлять приоритеты.

**Отслеживания прогресса:** Контроль выполнения задач, мониторинг сроков, выявление проблемных зон и отставаний.

**Коммуникации и совместной работы:** Обмен информацией, обсуждение задач, предоставление обратной связи и совместная работа над документами.

**Управления ресурсами:** Распределение ресурсов (времени, людей, бюджета) между проектами и задачами.

**Аналитики и отчетности:** Генерация отчетов о ходе выполнения проектов, производительности команд и отдельных сотрудников.

**Типы:**

Веб-приложения для управления задачами и проектами можно классифицировать по различным критериям:

**По масштабу:**

* **Индивидуальные планировщики:** Для персонального использования (пример, реализованный ToDo GeekBrains в своей CLI-части).
* **Командные системы:** Для небольших и средних команд.
* **Корпоративные системы:** Для крупных организаций с большим количеством проектов и сложной структурой управления.

**По функциональности:**

* **Простые списки задач (To-Do List**): Базовый функционал для создания и отслеживания простых задач.
* **Канбан-доски:** Визуализация задач на доске, разделенной на колонки, соответствующие этапам выполнения (реализовано в веб-приложении ToDo GeekBrains).
* **Диаграммы Ганта:** Планирование проектов с учетом временных рамок и зависимостей между задачами.
* **Системы управления проектами (PMS):** Комплексные решения, включающие широкий спектр инструментов для планирования, выполнения, контроля и анализа проектов.

**По модели развертывания:**

* SaaS (Software as a Service) Облачные решения.
* On-premise Размещение на серверах клиента.

**Функциональные возможности:**

Веб-приложения для управления задачами и проектами предлагают широкий набор функций, которые могут варьироваться в зависимости от типа приложения и его целевой аудитории. Ключевые функциональные возможности включают:

**Создание и управление задачами:** Возможность создавать задачи, указывать их описание, назначать исполнителей, устанавливать сроки, определять приоритеты, добавлять подзадачи и метки.

**Управление проектами:** Создание проектов, объединение задач в проекты, определение этапов проекта, установка общих сроков и целей.

**Визуализация задач:** Отображение задач в виде списка, канбан-доски, диаграммы Ганта или календаря.

**Управление сроками:** Установка сроков выполнения задач и проектов, напоминания о приближающихся дедлайнах.

**Назначение исполнителей:** Распределение задач между участниками команды, определение ролей и уровней доступа.

**Комментирование и обсуждение:** Возможность добавлять комментарии к задачам, обмениваться сообщениями и файлами. (Реализовано в ToDo GeekBrains)

**Управление статусами задач**: Отслеживание прогресса выполнения задач с помощью статусов ("Планы", "В работе", "Тестирование", "Завершено" и т.д.) (Реализовано в ToDo GeekBrains)

**Управление пользователями:** Добавление и удаление пользователей из проекта, разграничение прав. (Реализовано в ToDo GeekBrains)

**Уведомления:** Получение уведомлений об изменениях в задачах, комментариях и приближающихся сроках.

**Отчетность:** Генерация отчетов о ходе выполнения проектов, затраченном времени и производительности команды.

**Интеграция:** Интеграция с другими приложениями (календари, почтовые клиенты, облачные хранилища).

**Критические функции (реализованные в ToDo GeekBrains):**

* Создание и управление проектами.
* Создание и управление задачами (включая описание, исполнителя и статус).
* Визуализация задач в формате канбан-доски.
* Управление статусами задач.
* Комментирование задач.
* Управление пользователями (добавление, удаление, аутентификация).

**1.2. Архитектура веб-приложений: клиент-серверное взаимодействие, frontend и backend.**

Веб-приложения для управления задачами и проектами, как и большинство веб-приложений, строятся на основе клиент-серверной архитектуры. Эта архитектура предполагает разделение приложения на две основные части: клиентскую (frontend) и серверную (backend).

**Клиент (Frontend):** Это часть приложения, которая работает в браузере пользователя. Она отвечает за отображение интерфейса, взаимодействие с пользователем (обработку кликов, ввод данных, навигацию) и отправку запросов на сервер. Клиентская часть обычно реализуется с использованием HTML, CSS и JavaScript.

**Сервер (Backend):**

Это часть приложения, которая работает на удаленном сервере. Она отвечает за обработку запросов от клиента, взаимодействие с базой данных, выполнение бизнес-логики (например, проверка прав доступа, расчеты, обработка данных) и отправку ответов клиенту. Серверная часть может быть реализована с использованием различных языков программирования (Python, Java, Ruby, PHP, Node.js и др.) и фреймворков.

**База данных:** Место, где хранятся все данные проекта.

**Клиент-серверное взаимодействие:**

Взаимодействие между клиентом и сервером происходит по протоколу HTTP (Hypertext Transfer Protocol) или HTTPS (защищенная версия HTTP). Клиент отправляет запросы на сервер, а сервер обрабатывает эти запросы и отправляет ответы. Запросы и ответы могут содержать данные в различных форматах, например, HTML, JSON или XML.

**Пользовательский запрос:** Пользователь взаимодействует с интерфейсом приложения в браузере (например, нажимает кнопку "Создать задачу").

**Отправка запроса:** Браузер (клиент) формирует HTTP-запрос и отправляет его на сервер. Этот запрос может содержать данные, введенные пользователем (например, название задачи, описание, исполнитель).

**Обработка запроса:** Сервер получает запрос, обрабатывает его (например, проверяет права доступа пользователя, валидирует данные) и выполняет необходимые действия (например, создает новую запись в базе данных).

**Отправка ответа:** Сервер формирует HTTP-ответ и отправляет его клиенту. Этот ответ может содержать данные (например, подтверждение успешного создания задачи или сообщение об ошибке) или инструкции для обновления интерфейса.

**Отображение ответа:** Браузер (клиент) получает ответ от сервера и обновляет интерфейс в соответствии с полученными данными (например, отображает сообщение об успешном создании задачи или выводит список задач).

**Схема взаимодействия:**

**[Пользователь] <-> [Браузер (Frontend: HTML, CSS, JavaScript)] <-> [HTTP/HTTPS] <-> [Сервер (Backend: Python, Flask)] <-> [База данных]**

**Frontend (клиентская часть):**

**Отвечает за:**

* Отображение пользовательского интерфейса (UI).
* Взаимодействие с пользователем (обработка событий, ввод данных).
* Отправку запросов на сервер.
* Отображение данных, полученных от сервера.

**Технологии:**

* HTML (HyperText Markup Language): Язык разметки, используемый для создания структуры веб-страницы (заголовки, абзацы, списки, формы, таблицы и т.д.).
* CSS (Cascading Style Sheets): Язык стилей, используемый для оформления веб-страницы (цвет, шрифт, расположение элементов, анимация и т.д.).
* JavaScript: Язык программирования, используемый для добавления интерактивности на веб-страницу (обработка событий, динамическое обновление контента, отправка запросов на сервер без перезагрузки страницы и т.д.).
* Фреймворки и библиотеки: React, Angular, Vue.js, jQuery (упрощают разработку и добавляют готовые компоненты).
* Bootstrap: CSS-фреймворк для создания адаптивного дизайна (используется в ToDo GeekBrains).

**Backend (серверная часть):**

**Отвечает за:**

* Обработку запросов от клиента.
* Взаимодействие с базой данных (сохранение, извлечение, обновление, удаление данных).
* Выполнение бизнес-логики (например, проверка прав доступа, валидация данных, расчеты).
* Отправку ответов клиенту.

**Технологии:**

* Языки программирования: Python (используется в ToDo GeekBrains), Java, Ruby, PHP, Node.js и др.
* Фреймворки: Flask (используется в ToDo GeekBrains), Django (Python), Spring (Java), Ruby on Rails (Ruby), Laravel (PHP), Express.js (Node.js) и др.
* Базы данных: MySQL, PostgreSQL, MongoDB, SQLite и др.
* Веб-серверы: Apache, Nginx, Gunicorn (используется для развертывания Flask-приложений).

**1.3. Основные технологии и инструменты Fullstack-разработки**

Fullstack-разработка подразумевает создание как клиентской, так и серверной части приложения. Разработчик, владеющий fullstack-технологиями, способен самостоятельно создать полноценное веб-приложение.

**1.3.1. Python и Flask: обзор, преимущества использования для backend**

**Python:** Высокоуровневый язык программирования общего назначения, известный своей читаемостью, простотой и большим количеством библиотек для различных задач. Python широко используется в веб-разработке, машинном обучении, анализе данных и автоматизации.

**Преимущества Python для backend-разработки:** Простота и читаемость: Python имеет простой и понятный синтаксис, что делает его легким для изучения и использования. Код на Python легко читать и поддерживать. Большое количество библиотек: Python имеет огромное количество готовых библиотек для различных задач, включая веб-разработку, работу с базами данных, обработку данных и многое другое. Широкое сообщество: Python имеет большое и активное сообщество разработчиков, что обеспечивает доступ к большому количеству документации, учебных материалов и поддержки. Кроссплатформенность: Python работает на различных операционных системах (Windows, macOS, Linux). Поддержка асинхронности: Позволяет обрабатывать множество запросов без блокирования основного потока.

**Flask:** Микрофреймворк для веб-разработки на Python. Flask предоставляет базовые инструменты для создания веб-приложений, но при этом остается гибким и расширяемым. Он не навязывает жесткую структуру проекта и позволяет разработчику выбирать те компоненты, которые ему необходимы.

**Преимущества Flask:**

* Минимализм: Flask предоставляет только самые необходимые функции, что делает его легким и быстрым.
* Гибкость: Flask позволяет разработчику выбирать и использовать те библиотеки и инструменты, которые ему нужны.
* Расширяемость: Flask легко расширяется с помощью различных расширений (Flask-SQLAlchemy, Flask-Login, Flask-WTF и др.).
* Простота: Flask прост в изучении и использовании, особенно для начинающих разработчиков.
* Встроенный сервер разработки: Flask имеет встроенный сервер разработки, который упрощает тестирование приложения.
* Поддержка шаблонов Jinja2: Flask использует шаблонизатор Jinja2, который позволяет создавать динамические HTML-страницы.
* RESTful API: Flask хорошо подходит для создания RESTful API.

**Применение Flask в приложении ToDo GeekBrains:** Flask используется для создания серверной части приложения, обработки запросов от клиента (добавление, удаление, изменение задач и проектов), взаимодействия с базой данных (хранение данных в JSON-файлах) и отправки ответов клиенту. Flask предоставляет маршрутизацию URL-адресов, обработку форм, управление сессиями и другие функции, необходимые для работы веб-приложения.

**1.3.2. HTML, CSS, JavaScript, Bootstrap, jQuery, jQuery UI: обзор, применение во frontend**

**HTML (HyperText Markup Language):** Язык разметки, используемый для создания структуры веб-страницы. HTML определяет, какие элементы будут отображаться на странице (заголовки, абзацы, списки, формы, изображения, ссылки и т.д.) и как они будут расположены друг относительно друга.

**CSS (Cascading Style Sheets):** Язык стилей, используемый для оформления веб-страницы. CSS определяет, как будут выглядеть элементы HTML (цвет, шрифт, размер, фон, расположение, анимация и т.д.).

**JavaScript:** Язык программирования, используемый для добавления интерактивности на веб-страницу. JavaScript позволяет обрабатывать события (клики, наведение курсора, ввод данных), динамически изменять содержимое страницы, отправлять запросы на сервер без перезагрузки страницы и многое другое.

**Bootstrap:** Популярный CSS-фреймворк, который предоставляет готовые компоненты и стили для создания адаптивного веб-дизайна. Bootstrap включает в себя сетку, кнопки, формы, навигационные панели, модальные окна и другие элементы интерфейса.

**Преимущества Bootstrap:**

* Адаптивность: Bootstrap позволяет создавать веб-страницы, которые хорошо выглядят на различных устройствах (компьютеры, планшеты, смартфоны).
* Готовые компоненты: Bootstrap предоставляет большое количество готовых компонентов, что ускоряет разработку.
* Простота использования: Bootstrap прост в изучении и использовании.
* Настраиваемость: Bootstrap можно настроить в соответствии с требованиями проекта.
* Большое сообщество: Bootstrap имеет большое и активное сообщество, что обеспечивает доступ к документации и поддержке.

**jQuery:** Библиотека JavaScript, которая упрощает работу с DOM (Document Object Model) и выполнение AJAX-запросов. jQuery предоставляет удобные функции для манипулирования элементами HTML, обработки событий, создания анимации и отправки запросов на сервер.

**Преимущества jQuery:**

* Упрощение работы с DOM: jQuery делает работу с DOM более простой и понятной.
* Кроссбраузерность: jQuery решает проблемы совместимости с различными браузерами.
* AJAX: jQuery упрощает выполнение AJAX-запросов.
* Анимация: jQuery предоставляет простые функции для создания анимации.
* Плагины: jQuery имеет большое количество плагинов, которые расширяют его функциональность.

**jQuery UI:** Надстройка над jQuery, которая предоставляет набор виджетов и эффектов для создания интерактивных пользовательских интерфейсов. jQuery UI включает в себя такие элементы, как календари, диалоговые окна, ползунки, вкладки, автозаполнение и перетаскивание элементов (drag-and-drop).

**Преимущества jQuery UI:** Готовые виджеты Поддержка тем Расширяемость

**Применение во frontend приложения ToDo GeekBrains:**

* HTML: Используется для создания структуры веб-страниц (формы входа, списка проектов, доски задач).
* CSS: Используется для оформления веб-страниц (стили Bootstrap и собственные стили).
* JavaScript: Используется для добавления интерактивности (обработка событий, отправка AJAX-запросов, динамическое обновление контента).
* Bootstrap: Используется для создания адаптивного дизайна и использования готовых компонентов (кнопки, формы, модальные окна).
* jQuery: Используется для упрощения работы с DOM и выполнения AJAX-запросов (обновление статуса задачи, добавление комментариев).
* jQuery UI: Используется для реализации перетаскивания задач между колонками на канбан-доске.

**1.3.3. JSON как формат обмена данными**

**JSON (JavaScript Object Notation):** Легковесный формат обмена данными, основанный на подмножестве синтаксиса JavaScript. JSON представляет собой текстовый файл, содержащий данные в виде пар "ключ-значение". Он широко используется в веб-разработке для передачи данных между клиентом и сервером.

**Преимущества JSON:**

* Легковесность: JSON имеет небольшой размер по сравнению с другими форматами данных (например, XML).
* Простота: JSON имеет простой и понятный синтаксис.
* Читаемость: JSON легко читается как человеком, так и машиной.
* Поддержка большинством языков программирования: JSON поддерживается большинством языков программирования, включая Python, JavaScript, Java, Ruby, PHP и др.
* Широкое использование в веб-разработке: JSON является стандартом де-факто для обмена данными в веб-приложениях.

**Пример JSON:**

**{**

**"project": {**

**"id": "123",**

**"name": "Разработка веб-приложения",**

**"tasks": [**

**{**

**"id": "456",**

**"title": "Создать макет дизайна",**

**"status": "В работе"}]}}**

**Применение в приложении ToDo GeekBrains:**

JSON используется как основной формат для хранения данных приложения (проектов, задач, пользователей) в файлах projects.json, tasks.json (в CLI-версии) и users.json. Также JSON используется для обмена данными между клиентом и сервером в веб-версии приложения. Например, при добавлении нового комментария клиент отправляет на сервер JSON-объект с данными комментария, а сервер возвращает JSON-объект с подтверждением успешного добавления.

**1.3.4. Системы контроля версий (Git)**

Система контроля версий (Version Control System, VCS): Программное обеспечение, которое позволяет отслеживать изменения в файлах, возвращаться к предыдущим версиям, работать над проектом совместно с другими разработчиками и управлять различными версиями проекта.

**Git:** Распределенная система контроля версий, разработанная Линусом Торвальдсом. Git является самой популярной системой контроля версий и используется большинством разработчиков.

**Преимущества Git:**

* Распределенность: Каждый разработчик имеет полную копию репозитория, что позволяет работать автономно и повышает надежность системы.
* Скорость: Git работает очень быстро, особенно при выполнении локальных операций.
* Ветвление (branching): Git позволяет легко создавать ветки для разработки новых функций или исправления ошибок, не затрагивая основную кодовую базу.
* Слияние (merging): Git предоставляет удобные инструменты для слияния изменений из разных веток.
* История изменений: Git хранит полную историю изменений, что позволяет отслеживать, кто, когда и какие изменения внес в проект.
* Бесплатность и открытый исходный код: Git является бесплатным и имеет открытый исходный код.
* Большое сообщество: Git имеет большое и активное сообщество, что обеспечивает доступ к документации и поддержке.

**Основные команды Git:**

* git init: Инициализация нового репозитория.
* git clone: Клонирование удаленного репозитория.
* git add: Добавление файлов в индекс (staging area).
* git commit: Сохранение изменений в репозитории.
* git push: Отправка изменений в удаленный репозиторий.
* git pull: Получение изменений из удаленного репозитория.
* git branch: Работа с ветками.
* git checkout: Переключение между ветками.
* git merge: Слияние веток.
* git status: Просмотр текущего состояния репозитория.
* git log: Просмотр истории изменений.

**Применение в приложении ToDo GeekBrains:**

Система контроля версий Git, хотя и не является частью работающего приложения, критически важна в процессе его разработки. Весь код приложения должен быть размещён в репозитории Git на платформе, такой как GitHub, GitLab или Bitbucket.

**Это позволяет:**

* Отслеживать изменения в коде.
* Возвращаться к предыдущим версиям.
* Совместно работать над проектом (хотя в данном случае проект индивидуальный, понимание принципов совместной работы важно).
* Демонстрировать код и историю разработки (что важно при сдаче дипломного проекта).
* Создавать резервные копии кода.

**1.4 Паттерны проектирования**

Паттерны (шаблоны) проектирования — это повторяемые архитектурные конструкции, представляющие собой решения часто встречающихся проблем в рамках определённого контекста при проектировании программного обеспечения.

**1.4.1 Модель MVC**

**MVC (Model-View-Controller**): Архитектурный паттерн, который разделяет приложение на три основных компонента: модель, представление и контроллер. MVC широко используется в веб-разработке для создания масштабируемых и поддерживаемых приложений.

**Модель (Model):** Отвечает за данные приложения и бизнес-логику. Модель предоставляет данные и методы для работы с ними. Модель не зависит от представления и контроллера. В контексте приложения ToDo GeekBrains, классы Project, Task и User являются частью модели. Они описывают структуру данных и содержат методы для работы с ними (например, add\_task, remove\_task, update\_task\_status в классе Project).

**Представление (View):** Отвечает за отображение данных пользователю. Представление получает данные от модели и отображает их в удобном для пользователя виде. Представление не содержит бизнес-логики. В приложении ToDo GeekBrains, HTML-шаблоны (base.html, index.html, login.html, project\_board.html, project\_list.html, user\_management.html) являются представлениями. Они используют шаблонизатор Jinja2 (встроенный во Flask) для динамического отображения данных, полученных от контроллера.

**Контроллер (Controller):** Отвечает за обработку действий пользователя и обновление модели и представления. Контроллер получает запросы от пользователя, обрабатывает их, взаимодействует с моделью и выбирает подходящее представление для отображения. В Flask, функции-обработчики маршрутов (routes) в файле web\_app.py выступают в роли контроллеров. Например, функция project\_board(project\_id) обрабатывает запрос на отображение доски задач проекта, получает данные о проекте и задачах из модели, а затем передает эти данные в шаблон project\_board.html для отображения.

**Взаимодействие компонентов MVC:**

* Пользовательское действие: Пользователь взаимодействует с представлением (например, нажимает кнопку).
* Запрос к контроллеру: Представление отправляет запрос контроллеру.
* Обработка запроса: Контроллер обрабатывает запрос, взаимодействует с моделью (получает или изменяет данные).
* Обновление модели: Модель обновляет данные (если необходимо).
* Выбор представления: Контроллер выбирает подходящее представление для отображения.
* Отображение данных: Представление получает данные от модели и отображает их пользователю.

**Преимущества MVC:**

* Разделение ответственности: MVC разделяет приложение на три компонента, каждый из которых имеет свою область ответственности. Это упрощает разработку, тестирование и поддержку приложения.
* Повторное использование кода: Модели и представления могут быть повторно использованы в различных частях приложения.
* Масштабируемость: MVC позволяет легко масштабировать приложение, добавляя новые модели, представления и контроллеры.
* Тестируемость: Каждый компонент MVC можно тестировать отдельно.
* Поддержка: MVC-приложения легче поддерживать и модифицировать.

**Применение в приложении ToDo GeekBrains:**

* Приложение ToDo GeekBrains, особенно его веб-часть, следует паттерну MVC.
* Модели: Project, Task, User (в models/).
* Представления: HTML-шаблоны (в templates/).
* Контроллеры: Функции-обработчики маршрутов во web\_app.py.

Например, при запросе страницы со списком проектов (/projects), контроллер (project\_list функция) обращается к модели (project\_storage.load\_projects()), получает список проектов, и передает его представлению (project\_list.html), которое генерирует HTML-страницу. При добавлении новой задачи через веб-интерфейс, данные из формы отправляются контроллеру (add\_task функция), который создает экземпляр Task, добавляет его в Project, сохраняет изменения в хранилище и перенаправляет пользователя обратно на страницу проекта.

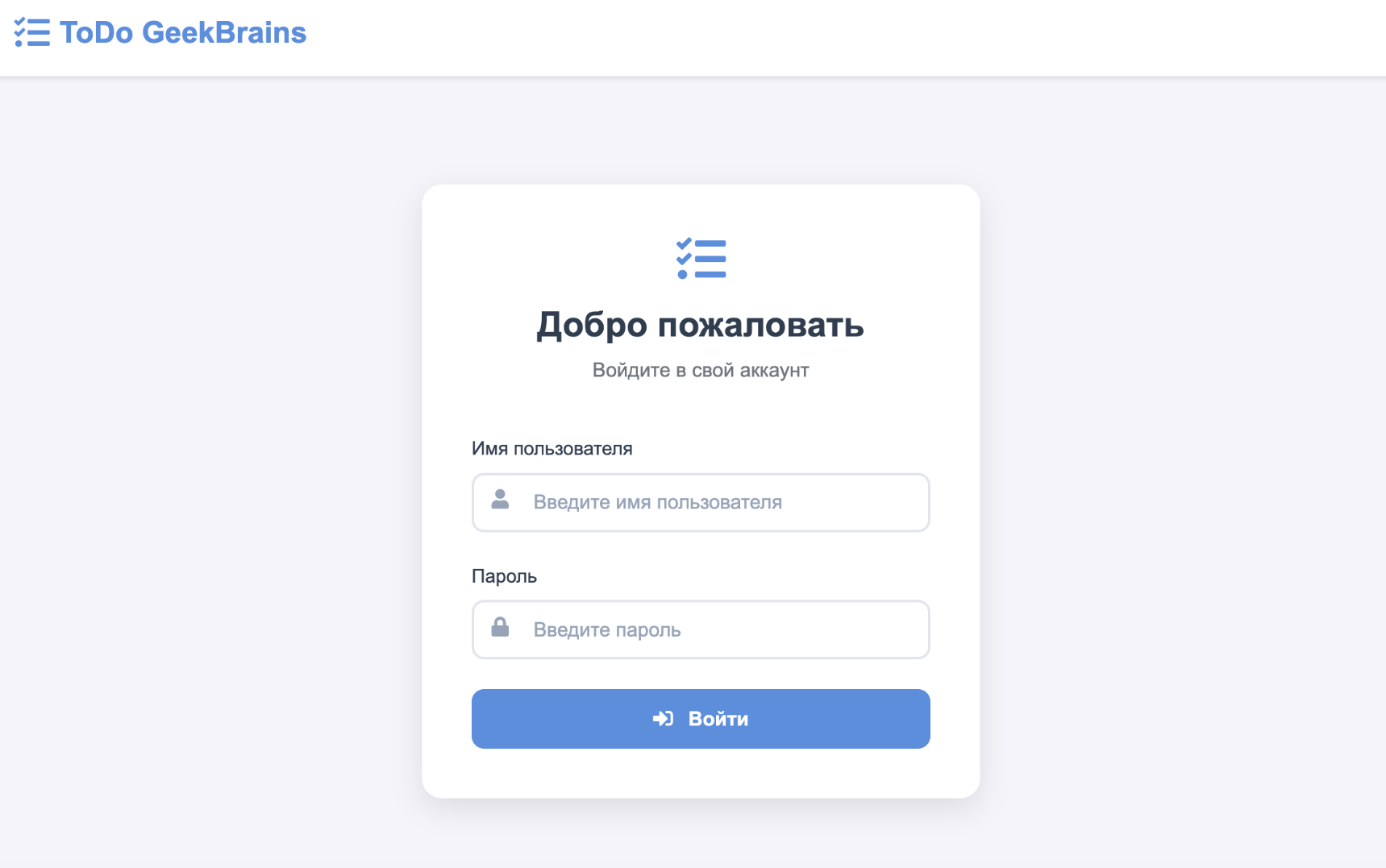
# Глава 2. Проектирование и реализация веб-приложения ToDo GeekBrains

Эта глава посвящена детальному описанию процесса разработки веб-приложения ToDo GeekBrains, начиная от постановки задачи и заканчивая описанием взаимодействия клиентской и серверной частей. Здесь подробно рассматриваются функциональные требования, структура данных, реализация backend и frontend, а также механизмы их взаимодействия.

**2.1. Постановка задачи и определение функциональных требований**

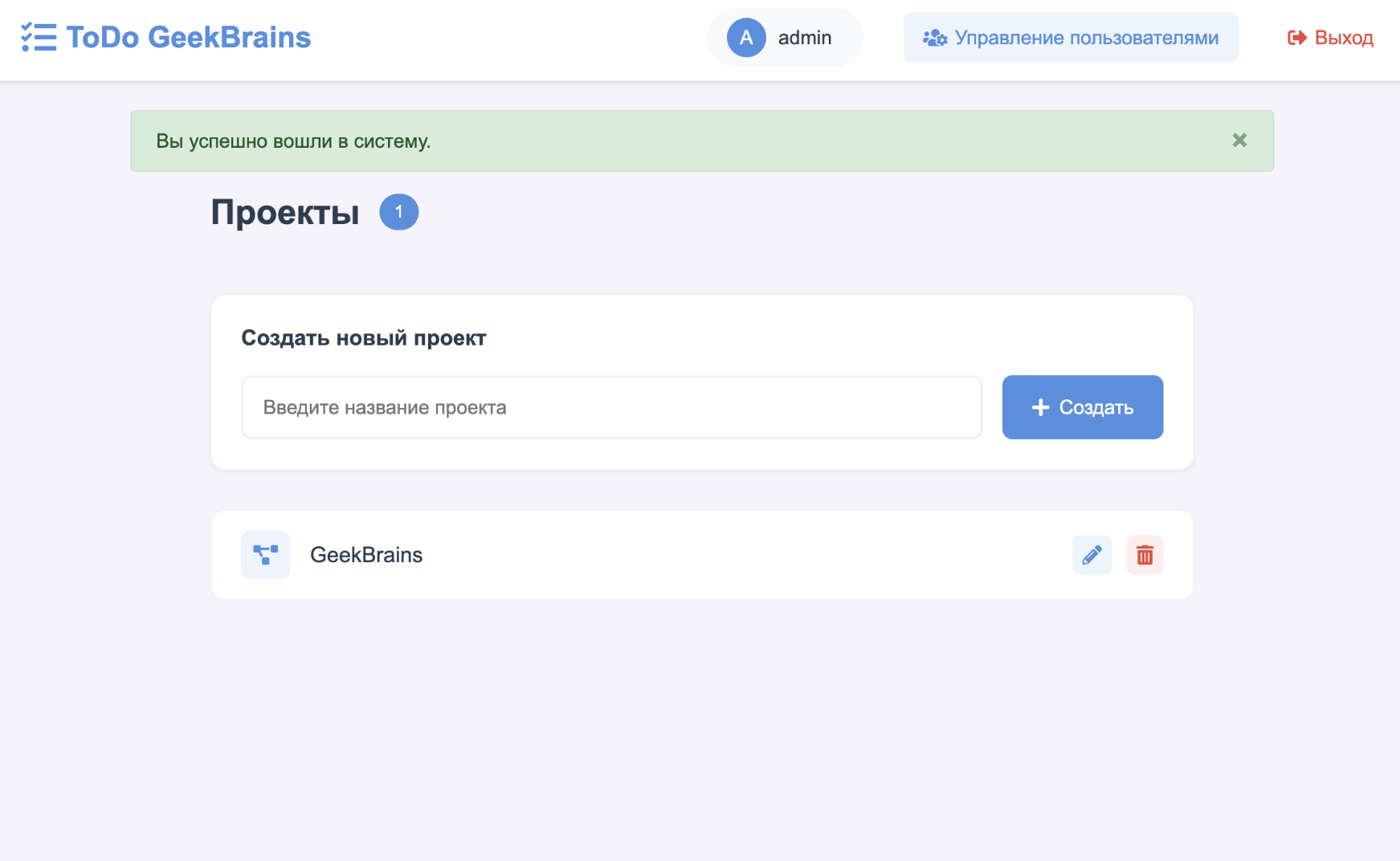
В рамках дипломного проекта было разработано веб-приложение ToDo GeekBrains, предназначенное для управления проектами и задачами. Приложение предоставляет пользователям возможности по организации рабочего процесса, отслеживанию прогресса выполнения задач и совместной работе. Перед началом разработки были сформулированы следующие функциональные требования:

**2.1.1. Аутентификация и авторизация пользователей (вход, выход, разграничение прав доступа)**



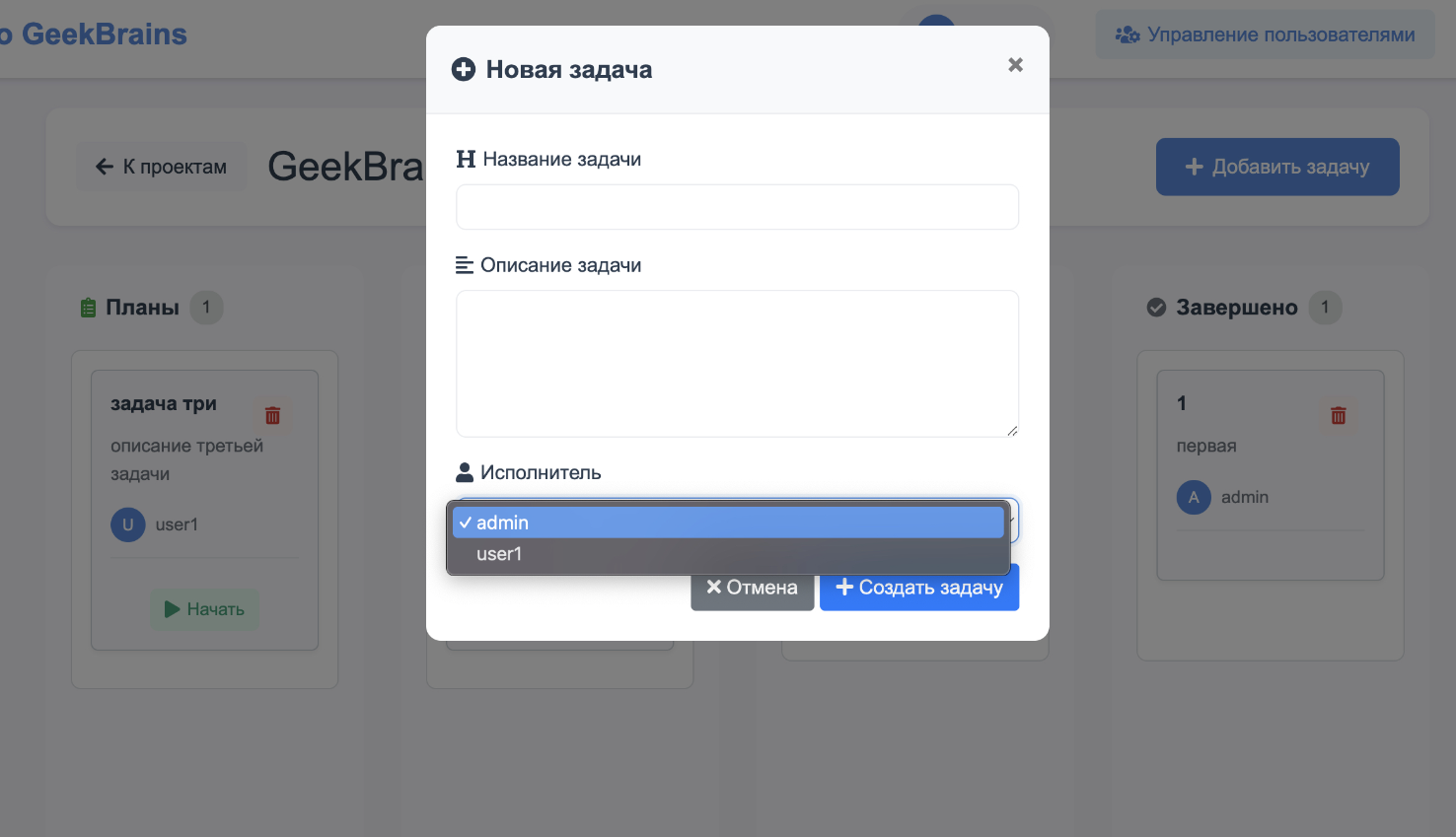
* **Вход:** Пользователи должны иметь возможность входить в систему, используя свой логин и пароль. При успешной аутентификации пользователь получает доступ к функционалу приложения. Реализовано на странице /login (файл to\_do\_app/templates/login.html).
* **Выход:** Пользователи должны иметь возможность безопасно выйти из системы, завершив сеанс работы. Реализовано через маршрут /logout (файл to\_do\_app/web\_app.py).
* **Разграничение прав доступа:** В приложении предусмотрены две роли: "администратор" и "пользователь". Администратор имеет доступ ко всему функционалу, включая управление пользователями. Обычные пользователи имеют доступ к управлению проектами и задачами. Реализовано с помощью декораторов @login\_required и @admin\_required (файл to\_do\_app/web\_app.py).

**2.1.2. Управление проектами: создание, удаление, просмотр списка проектов**



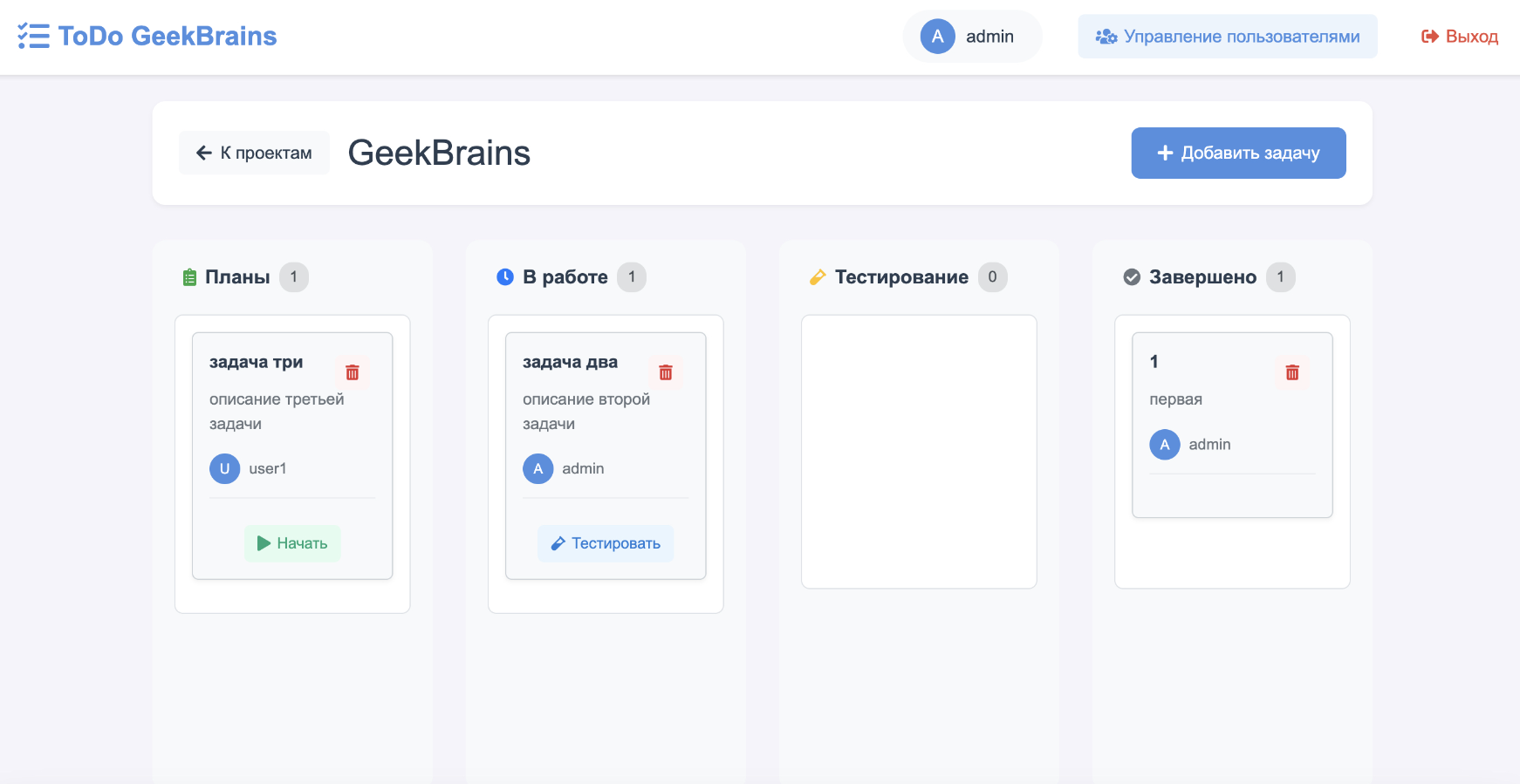
* **Создание:** Пользователи могут создавать новые проекты, указывая их название. Реализовано на странице /projects через форму, отправляющую POST-запрос на /projects/add (файл to\_do\_app/templates/project\_list.html).
* **Удаление:** Пользователи могут удалять существующие проекты. Реализовано через GET-запрос на /projects/delete/<project\_id> (файл to\_do\_app/web\_app.py).
* **Просмотр списка:** Пользователи могут просматривать список всех доступных им проектов. Реализовано на странице /projects (файл to\_do\_app/templates/project\_list.html).

**2.1.3. Управление задачами: создание, удаление, редактирование статуса, назначение исполнителя, просмотр задач в рамках проекта**



* **Создание:** Внутри каждого проекта пользователи могут создавать новые задачи, указывая название, описание, и назначая исполнителя. Реализовано внутри проекта (/project/<project\_id>) через форму, отправляющую POST-запрос на /project/<project\_id>/task/add (файл to\_do\_app/templates/project\_board.html).
* **Удаление:** Пользователи могут удалять задачи внутри проекта. Реализовано через GET-запрос на /project/<project\_id>/task/<task\_id>/delete (файл to\_do\_app/web\_app.py).
* **Редактирование статуса:** Пользователи могут изменять статус задачи ("Планы", "В работе", "Тестирование", "Завершено"). Реализовано через GET-запрос на /project/<project\_id>/task/<task\_id>/update/<new\_status> (файл to\_do\_app/web\_app.py).
* **Назначение исполнителя:** При создании задачи можно выбрать исполнителя из списка зарегистрированных пользователей.
* **Просмотр задач:** Задачи отображаются в рамках соответствующего проекта в виде канбан-доски.

**2.1.4. Канбан-доска**



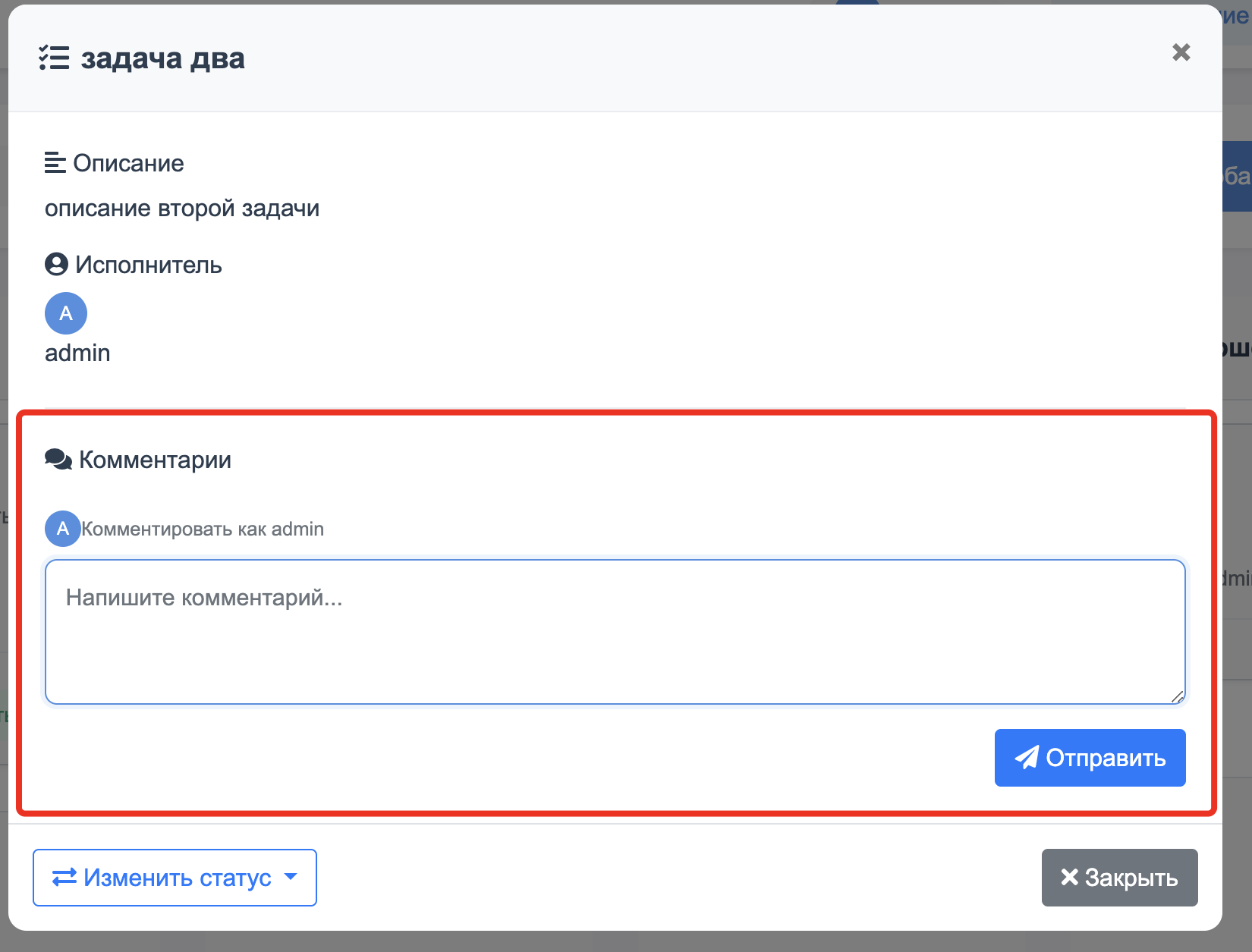
Задачи визуализируются в виде карточек, распределенных по столбцам, соответствующим их статусам.

Пользователи могут перетаскивать задачи между столбцами для изменения их статуса (реализовано с использованием jQuery UI). Реализовано на странице /project/<project\_id> (файл to\_do\_app/templates/project\_board.html).

**2.1.5. Комментирование задач**

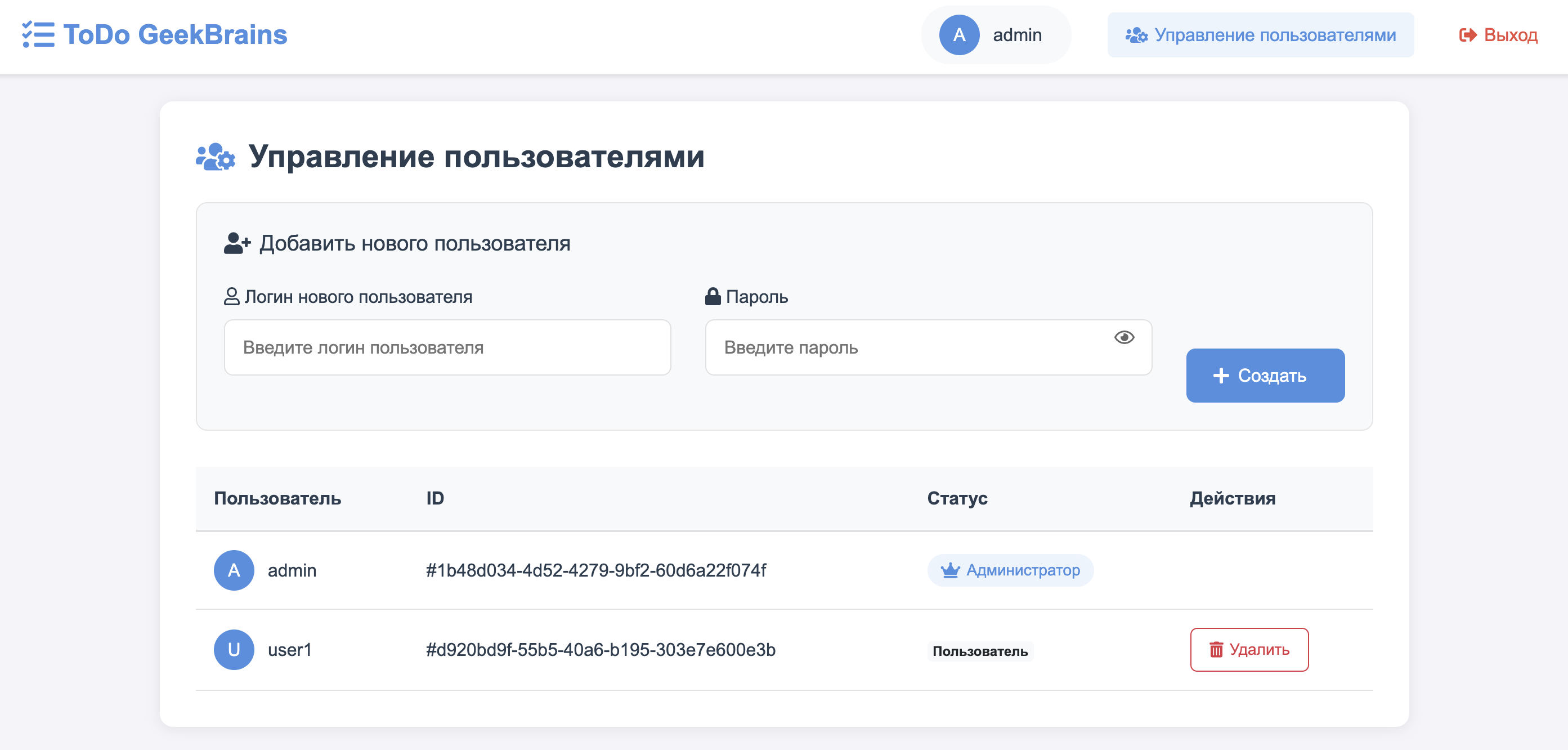
Пользователи могут добавлять комментарии к задачам, обсуждая детали и прогресс выполнения.

Комментарии отображаются в хронологическом порядке с указанием автора и текста комментария. Реализовано через маршрут /project/<project\_id>/task/<task\_id>/comments (файлы to\_do\_app/web\_app.py и to\_do\_app/templates/project\_board.html).



**2.1.6 Управление пользователями (CRUD). Только для администратора**

* Создание: Администратор может создавать новых пользователей, указывая их логин и пароль.
* Чтение (просмотр): Администратор может просматривать список всех зарегистрированных пользователей.
* Удаление: Администратор может удалять пользователей (кроме собственной учетной записи).



CRUD операции реализованы на странице /admin/users (файл to\_do\_app/templates/user\_management.html), а удаление пользователя через /admin/users/delete/<user\_id>.

**2.2. Проектирование структуры данных**

Для хранения данных приложения были разработаны три основные модели: "Пользователь", "Проект" и "Задача". Эти модели определяют структуру данных и взаимосвязи между ними.

**2.2.1. Модель данных "Пользователь" (User)**

Модель User (файл to\_do\_app/models/user.py) представляет информацию о зарегистрированных пользователях.

* Атрибуты:
* id (str): Уникальный идентификатор пользователя (генерируется с помощью uuid.uuid4()).
* username (str): Логин пользователя.
* password (str): Пароль пользователя (в данном примере хранится в открытом виде, что недопустимо в реальных приложениях). В реальном приложении следует использовать хеширование паролей.

**Методы:**

* to\_dict(): Преобразует объект пользователя в словарь для сериализации.
* from\_dict(dict\_obj): (classmethod) Создает объект пользователя из словаря.

**2.2.2. Модель данных "Проект" (Project)**

Модель Project (файл to\_do\_app/models/project.py) представляет информацию о проектах.

**Атрибуты:**

* id (str): Уникальный идентификатор проекта (генерируется с помощью uuid.uuid4()).
* name (str): Название проекта.
* tasks (list): Список задач, относящихся к проекту (список объектов Task).

**Методы:**

* add\_task(task): Добавляет задачу в проект.
* remove\_task(task\_id): Удаляет задачу по ID.
* update\_task\_status(task\_id, new\_status): Обновляет статус задачи.
* to\_dict(): Преобразует объект проекта в словарь.
* from\_dict(dict\_obj): (classmethod) Создает объект проекта из словаря.

**2.2.3. Модель данных "Задача" (Task)**

Модель Task (файл to\_do\_app/models/task.py) представляет информацию о задачах в рамках проекта.

**Атрибуты:**

* id (str): Уникальный идентификатор задачи (генерируется с помощью uuid.uuid4()).
* title (str): Название задачи.
* description (str): Описание задачи.
* status (str): Статус задачи ("Планы", "В работе", "Тестирование", "Завершено").
* assigned\_to (str): Логин пользователя, назначенного исполнителем задачи.
* comments (list): Список комментариев к задаче (список словарей с ключами "author" и "text").

**Методы:**

* add\_comment(author, text): Добавляет комментарий к задаче.
* to\_dict(): Преобразует объект задачи в словарь.
* from\_dict(dict\_obj): (classmethod) Создаёт объект задачи из словаря.
* \_\_str\_\_(): Возвращает строковое представление задачи.

**2.2.4. Взаимосвязи между моделями**

Пользователь - Проект: Один ко многим (один пользователь может быть назначен на несколько проектов, но в рамках данного приложения эта связь не выражена явно в моделях; скорее, она подразумевается через назначение задач пользователям внутри проектов).

Проект - Задача: Один ко многим (один проект может содержать множество задач). Эта связь реализована через атрибут tasks в модели Project, который содержит список объектов Task.

Задача - Пользователь: Многие к одному (несколько задач могут быть назначены одному пользователю). Эта связь реализована через атрибут assigned\_to в модели Task, который содержит логин пользователя.

Задача - Комментарий: Один ко многим (к одной задаче может быть несколько комментариев). Эта связь реализована через атрибут comments в модели Task.

**2.3. Реализация backend (серверной части) на Python с использованием Flask**

Серверная часть приложения разработана на языке Python с использованием микрофреймворка Flask. Flask обеспечивает управление маршрутизацией, обработку HTTP-запросов и взаимодействие с хранилищем данных.

**2.3.1. Создание API-маршрутов (endpoints) для обработки запросов**

В файле to\_do\_app/web\_app.py определены следующие API-маршруты:

* /login (POST): Аутентификация пользователя. Принимает username и password в теле запроса. Ищет пользователя в хранилище (users). При успехе устанавливает флаг logged\_in и имя пользователя в сессии.
* /logout (GET): Выход из системы. Удаляет флаг logged\_in и имя пользователя из сессии.
* /projects (GET): Получение списка проектов. Возвращает список всех проектов из хранилища.
* /projects/add (POST): Создание нового проекта. Принимает name в теле запроса. Создает новый объект Project и сохраняет его в хранилище.
* /projects/delete/<project\_id> (GET): Удаление проекта. Удаляет проект с указанным project\_id из хранилища.
* /project/<project\_id> (GET): Отображение канбан-доски проекта. Возвращает информацию о проекте с указанным project\_id, включая список задач, распределенных по статусам.
* /project/<project\_id>/task/add (POST): Добавление задачи в проект. Принимает title, description и assigned\_to в теле запроса. Создает новый объект Task и добавляет его в список задач проекта.
* /project/<project\_id>/task/<task\_id>/update/<new\_status> (GET): Обновление статуса задачи. Обновляет статус задачи с указанными task\_id и new\_status в проекте.
* /project/<project\_id>/task/<task\_id>/delete (GET): Удаление задачи. Удаляет задачу с указанным task\_id из проекта.
* /admin/users (GET, POST): Управление пользователями (для администратора). В GET-запросе возвращает список пользователей. В POST-запросе создает нового пользователя, принимая new\_username и new\_password.
* /admin/users/delete/<user\_id> (GET): Удаление пользователя (для администратора). Удаляет пользователя с указанным user\_id.
* /project/<project\_id>/task/<task\_id>/comments (GET, POST): Загрузка (GET) и добавление (POST) комментариев. Для GET возвращает список комментариев к задаче. Для POST принимает comment в теле запроса, добавляет комментарий к задаче, сохраняя автора из текущей сессии.

**2.3.2. Использование декораторов @login\_required и @admin\_required для защиты маршрутов**

* @login\_required: Этот декоратор проверяет, авторизован ли пользователь (есть ли флаг logged\_in в сессии). Если пользователь не авторизован, он перенаправляется на страницу входа (/login). Этот декоратор применяется ко всем маршрутам, требующим аутентификации.
* @admin\_required: Этот декоратор дополнительно проверяет, является ли авторизованный пользователь администратором (значение username в сессии должно быть равно 'admin'). Если пользователь не является администратором, он перенаправляется на страницу списка проектов (/projects). Этот декоратор применяется к маршрутам управления пользователями.

**2.3.3. Взаимодействие с хранилищем данных (FileStorage и UserStorage)**

* FileStorage (файл to\_do\_app/storage/file\_storage.py): Этот класс отвечает за хранение данных о проектах и задачах в JSON-файле (projects.json). Он предоставляет методы для загрузки (load\_projects, load\_tasks) и сохранения (save\_projects, save\_tasks) данных.
* UserStorage (файл to\_do\_app/storage/file\_storage\_users.py): Этот класс отвечает за хранение данных о пользователях в JSON-файле (users.json). Он предоставляет методы для загрузки (load\_users) и сохранения (save\_users)

**2.3.4. Сериализация данных в JSON**

Для обмена данными между backend и frontend используется формат JSON. Модели данных (User, Project, Task) имеют методы to\_dict(), которые преобразуют объекты в словари, которые затем могут быть сериализованы в JSON с помощью стандартной библиотеки Python json. Аналогично, методы from\_dict() используются для создания объектов из словарей, полученных из JSON. Flask автоматически обрабатывает сериализацию и десериализацию JSON при отправке и получении данных через API.

**2.4. Реализация frontend (клиентской части) с использованием HTML, CSS, JavaScript, Bootstrap, jQuery**

Клиентская часть приложения разработана с использованием стандартных веб-технологий: HTML, CSS и JavaScript. Для упрощения разработки и улучшения внешнего вида используются библиотека Bootstrap (фреймворк для стилизации) и библиотека jQuery (для манипулирования DOM и реализации AJAX-запросов).

**2.4.1. Разработка структуры HTML-страниц с использованием шаблонизатора Jinja2**

* **base.html:** Базовый шаблон, определяющий общую структуру всех страниц приложения (шапка, подвал, подключение стилей и скриптов). Все остальные шаблоны наследуются от него.
* **login.html:** Шаблон страницы входа в систему. Содержит форму с полями для ввода логина и пароля.
* **project\_list.html:** Шаблон страницы списка проектов. Отображает список проектов и форму для создания нового проекта.
* **project\_board.html:** Шаблон страницы канбан-доски проекта. Отображает задачи, распределенные по столбцам статусов, и форму для добавления новой задачи.
* user\_management.html: Шаблон страницы управления пользователями (доступен только администратору). Отображает список пользователей и форму для добавления нового пользователя.

**2.4.2. Стилизация интерфейса с помощью Bootstrap и собственных CSS-стилей**

Bootstrap предоставляет набор готовых CSS-классов для стилизации элементов интерфейса (кнопки, формы, таблицы и т.д.), что значительно ускоряет процесс разработки и обеспечивает единообразный внешний вид приложения. Кроме того, в файле base.html определены собственные CSS-стили, которые переопределяют и дополняют стили Bootstrap, придавая приложению уникальный дизайн.

**2.4.3. Использование JavaScript и jQuery для:**

* **Отправки AJAX-запросов на сервер:** jQuery используется для отправки асинхронных запросов к API Flask без перезагрузки страницы. Например, при добавлении новой задачи, изменении статуса задачи или добавлении комментария отправляется AJAX-запрос, который обновляет данные на сервере и в интерфейсе.
* **Динамического обновления содержимого страниц без перезагрузки:** После успешного выполнения AJAX-запросов jQuery используется для обновления соответствующих элементов на странице (например, добавление новой задачи в список, обновление счетчика задач в столбце статуса, добавление нового комментария).
* **Реализации drag-and-drop функциональности для канбан-доски (jQuery UI):** Библиотека jQuery UI используется для реализации возможности перетаскивания задач между столбцами канбан-доски. При перетаскивании задачи отправляется AJAX-запрос на сервер для обновления статуса задачи.
* **Отображения уведомлений (всплывающих сообщений):** Для отображения уведомлений об успешном выполнении операций (добавление задачи, изменение статуса, добавление комментария) или об ошибках используются всплывающие сообщения (alert'ы Bootstrap).
* **Управления модальными окнами:** Модальные окна Bootstrap используются для отображения форм добавления новой задачи и просмотра полной информации о задаче. jQuery используется для открытия и закрытия модальных окон.
* **Валидации форм на стороне клиента:** Перед отправкой данных на сервер выполняется базовая валидация форм (например, проверка заполнения обязательных полей) с помощью JavaScript.

**2.5. Взаимодействие frontend и backend**

Взаимодействие между frontend и backend осуществляется посредством обмена данными в формате JSON через API, реализованный на Flask.

**2.5.1. Обмен данными в формате JSON**

Frontend отправляет запросы к API, передавая данные в формате JSON в теле запроса (для POST-запросов) или в параметрах URL (для GET-запросов). Backend обрабатывает запросы, взаимодействует с хранилищем данных и возвращает ответы также в формате JSON.

**2.5.2. Обработка ответов сервера и обновление интерфейса**

После получения ответа от сервера frontend обрабатывает его и обновляет интерфейс. Например, если сервер вернул JSON с данными о новой задаче, frontend добавляет эту задачу в соответствующий столбец канбан-доски. Если сервер вернул сообщение об ошибке, frontend отображает это сообщение пользователю.

**2.6. Описание классов, функций и методов**

В коде используются следующие основные классы и функции:

* **Классы моделей данных:** User, Project, Task (описаны в разделе 2.2).
* **Классы хранилища данных:** FileStorage, UserStorage (описаны в разделе 2.3.3).
* **Функции Flask (маршруты):** Описаны в разделе 2.3.1.
* **Вспомогательные функции:** save\_projects, get\_project в файле web\_app.py. Служат для сохранения списка проектов и получения объекта проекта по ID соответственно.
* **Декораторы:** @login\_required, @admin\_required (описаны в разделе 2.3.2).
* **Функции JavaScript:** В файлах шаблонов (особенно project\_board.html) определены функции JavaScript, использующие jQuery, для обработки событий (клики, перетаскивания), отправки AJAX-запросов и обновления интерфейса. Например: updateTaskStatus, showNotification, loadComments, addCommentToList.

# Глава 3. Тестирование, отладка и развертывание приложения "ToDo GeekBrains"

Эта глава посвящена ключевым этапам разработки приложения "ToDo GeekBrains": тестированию, отладке и развертыванию. В ней подробно рассматриваются методы, использованные для обеспечения качества, стабильности и доступности приложения.

**3.1. Тестирование**

Тестирование является неотъемлемой частью процесса разработки, позволяющей выявить и устранить ошибки, а также убедиться в соответствии приложения заявленным требованиям. В рамках разработки "ToDo GeekBrains" было проведено комплексное тестирование, включающее функциональное тестирование, тестирование API, UI-тестирование и тестирование безопасности. Все тесты проводились локально, на машине разработчика.

**3.1.1. Функциональное тестирование: проверка корректности работы всех функций приложения**

Функциональное тестирование было направлено на проверку соответствия реализованных функций приложения его спецификации. Тестировались следующие основные функции:

**Аутентификация и авторизация:**

* Успешный вход: Проверка возможности входа в систему с корректными учетными данными (логин и пароль). Тестирование проводилось как для пользователя admin, так и для обычного пользователя (user1).
* Неудачный вход: Проверка реакции системы на ввод неверных учетных данных (неверный логин, неверный пароль, пустые поля). Ожидаемое поведение – отображение сообщения об ошибке "Неверный логин или пароль.".
* Выход из системы: Проверка корректности выхода пользователя из системы и перенаправления на страницу входа.
* Ограничение доступа: Проверка ограничения доступа к страницам, требующим авторизации (список проектов, доска проекта, управление пользователями) для неавторизованных пользователей. Ожидаемое поведение – перенаправление на страницу входа с сообщением "Пожалуйста, войдите в систему.".
* Разграничение прав доступа (admin/user): Проверка доступа к функциональности управления пользователями (добавление, удаление). Только пользователь admin должен иметь доступ к этой странице. Ожидаемое поведение при попытке доступа обычного пользователя – перенаправление на страницу списка проектов с сообщением "Доступ только для администратора.".

**Управление проектами:**

* Создание проекта: Проверка возможности создания нового проекта с уникальным названием. Проверка, что пустое название проекта не допускается (отображается сообщение "Название проекта не может быть пустым!").
* Удаление проекта: Проверка возможности удаления существующего проекта. Тестировалось, что при подтверждении удаления проект действительно удаляется из списка.
* Отображение списка проектов: Проверка корректности отображения списка проектов для авторизованного пользователя.
* Переход на доску проекта

**Управление задачами (внутри проекта):**

* Создание задачи: Проверка возможности создания новой задачи с указанием названия, описания и исполнителя. Проверка, что пустые название и описание не допускаются (отображается сообщение "Название и описание задачи не могут быть пустыми!").
* Удаление задачи: Проверка возможности удаления задачи из проекта.
* Изменение статуса задачи: Проверка возможности изменения статуса задачи между "Планы", "В работе", "Тестирование" и "Завершено" с использованием механизма drag-and-drop и кнопок. Проверка, что недопустимые статусы не принимаются (отображается сообщение "Некорректный статус.").
* Отображение задач на канбан-доске: Проверка корректного отображения задач в соответствующих колонках канбан-доски в зависимости от их статуса.
* Добавление комментариев: Проверка возможности добавления комментариев к задаче авторизованными пользователями. Проверялось, что пустые комментарии не добавляются.
* Отображение комментариев: Проверка корректного отображения комментариев к задаче, включая имя автора и текст комментария.
* Назначение исполнителя

**Управление пользователями (доступно только для администратора):**

* Добавление пользователя: Проверка возможности добавления нового пользователя с указанием логина и пароля. Проверка, что логин и пароль не могут быть пустыми, а также что пользователь с таким логином не существует (отображаются соответствующие сообщения).
* Удаление пользователя: Проверка возможности удаления существующего пользователя. Проверка, что пользователя admin удалить нельзя (отображается сообщение "Нельзя удалить пользователя admin.").

Тестирование проводилось путем выполнения сценариев использования каждой функции и проверки соответствия фактического результата ожидаемому. При обнаружении несоответствий (багов) производилась отладка кода и повторное тестирование.

**3.1.2. Тестирование API**

Тестирование API приложения "ToDo GeekBrains" было проведено, хотя и не выделено в явном виде в коде с использованием специализированных фреймворков (например, pytest или unittest). Тестирование API неявно осуществлялось в рамках функционального и UI-тестирования, поскольку взаимодействие пользовательского интерфейса с серверной частью (Flask) происходит через API-эндпоинты.

Каждый маршрут (route) во Flask-приложении (web\_app.py) представляет собой API-эндпоинт. Вот основные эндпоинты и соответствующие им проверки:

* /login (POST): Проверяет учетные данные пользователя и устанавливает сессию при успешной аутентификации.
* /logout: Удаляет данные сессии, осуществляя выход пользователя.
* /projects (GET): Возвращает список проектов.
* /projects/add (POST): Создает новый проект.
* /projects/delete/<project\_id> (GET): Удаляет проект с указанным ID.
* /project/<project\_id> (GET): Возвращает представление канбан-доски для проекта.
* /project/<project\_id>/task/add (POST): Добавляет новую задачу в проект.
* /project/<project\_id>/task/<task\_id>/update/<new\_status> (GET): Обновляет статус задачи.
* /project/<project\_id>/task/<task\_id>/delete (GET): Удаляет задачу.
* /project/<project\_id>/task/<task\_id>/comments (POST/GET): Добавляет комментарий/загружает список.
* /admin/users (GET, POST): Управление пользователями (добавление).
* /admin/users/delete/<user\_id> (GET): Удаление пользователя.

В процессе функционального и UI-тестирования осуществлялись запросы к этим эндпоинтам (через взаимодействие с веб-интерфейсом или напрямую, например, с использованием curl или Postman, хотя явных инструкций по использованию Postman в коде нет), и проверялись ответы сервера (коды состояния HTTP, возвращаемые данные). Например, при создании задачи проверялось, что сервер возвращает код 200 OK и что задача действительно добавляется в список задач проекта (это видно при последующем отображении доски проекта). При попытке удаления несуществующего проекта проверялось, что сервер возвращает корректное сообщение об ошибке.

В идеальном сценарии, для более строгого и автоматизированного тестирования API, следовало бы использовать отдельные тестовые сценарии, которые бы напрямую вызывали API-эндпоинты с различными входными данными (включая некорректные) и проверяли бы соответствие ответов сервера ожидаемым результатам. Это позволило бы более точно локализовать ошибки, связанные с работой API.

**3.1.3. Тестирование пользовательского интерфейса (UI-тестирование)**

UI-тестирование было сосредоточено на проверке удобства использования, корректности отображения элементов интерфейса и отзывчивости приложения на действия пользователя. Тестировались следующие аспекты:

* Навигация: Проверка корректности переходов между страницами (вход, список проектов, доска проекта, управление пользователями).
* Отображение данных: Проверка правильности отображения списка проектов, задач на канбан-доске (включая названия, описания, исполнителей, статусы), комментариев, списка пользователей.
* Интерактивность: Проверка работы кнопок (создание проекта, создание задачи, удаление проекта/задачи, изменение статуса задачи, добавление комментария, вход/выход, добавление/удаление пользователя), полей ввода, выпадающих списков (выбор исполнителя при создании задачи), механизма drag-and-drop для изменения статуса задач на канбан-доске.
* Обратная связь: Проверка отображения сообщений об успешных операциях (например, "Проект успешно создан", "Задача успешно добавлена", "Статус задачи обновлён") и ошибках (например, "Неверный логин или пароль", "Название проекта не может быть пустым"). Проверка, что сообщения отображаются в течение достаточного времени и затем исчезают.
* Адаптивность: Проверка корректности отображения на разных разрешениях экрана

Тестирование проводилось вручную, путем взаимодействия с приложением через веб-браузер. Использовались различные браузеры (Google Chrome, Mozilla Firefox) для проверки кроссбраузерности. Особое внимание уделялось проверке работы механизма drag-and-drop, поскольку он реализован с использованием JavaScript-библиотеки jQuery UI.

В процессе тестирования выявлялись и устранялись мелкие недочеты в верстке и стилях, а также проблемы с интерактивностью элементов.

**3.1.4. Тестирование безопасности**

Тестирование безопасности в рамках данного проекта было ограничено базовыми проверками, направленными на предотвращение несанкционированного доступа к данным и функциональности:

* Проверка аутентификации: Убедились, что неавторизованные пользователи не могут получить доступ к страницам, требующим входа в систему (список проектов, доска проекта, управление пользователями). При попытке доступа к таким страницам пользователь должен перенаправляться на страницу входа.
* Проверка авторизации: Убедились, что только пользователь с правами администратора (admin) имеет доступ к странице управления пользователями. Попытки доступа к этой странице от имени обычного пользователя должны приводить к перенаправлению на страницу списка проектов с сообщением об ошибке.
* Хранение паролей: В данном приложении пароли пользователей хранятся в открытом виде в файле users.json. Это крайне небезопасно и является серьезной уязвимостью. В реальных приложениях пароли никогда не должны храниться в открытом виде. Их необходимо хешировать с использованием надежных алгоритмов (например, bcrypt, scrypt, Argon2). В рамках дипломной работы этот аспект был отмечен как требующий улучшения, но не был реализован из-за ограничений по времени и объему проекта.
* Защита от XSS (Cross-Site Scripting): В приложении не используются явные механизмы защиты от XSS-атак (например, экранирование пользовательского ввода). Однако, Flask по умолчанию использует шаблонизатор Jinja2, который автоматически экранирует переменные, подставляемые в шаблоны, что обеспечивает базовую защиту от XSS. Тем не менее, рекомендуется явно указывать необходимость экранирования при выводе пользовательского ввода в шаблонах, используя фильтр |e (например, {{ user\_input|e }}). В рамках проекта это не было сделано, что также является пунктом для улучшения.
* Защита от CSRF (Cross-Site Request Forgery): В приложении не реализована защита от CSRF-атак. Это означает, что злоумышленник потенциально может подделать запрос от имени авторизованного пользователя (например, на удаление проекта или изменение статуса задачи). Для защиты от CSRF рекомендуется использовать специальные токены, которые генерируются сервером и проверяются при каждом запросе, изменяющем данные.

В целом, тестирование безопасности выявило ряд существенных уязвимостей, которые необходимо устранить в реальном приложении.

**3.2. Отладка приложения**

Отладка приложения "ToDo GeekBrains" осуществлялась в процессе разработки для выявления и устранения ошибок. Использовались стандартные средства отладки, предоставляемые Flask и средой разработки.

**3.2.1. Использование режима отладки Flask (debug=True)**

При запуске приложения в режиме отладки (app.run(debug=True)) Flask предоставляет подробную информацию об ошибках, возникающих в процессе работы приложения.

**Это включает в себя:**

* Интерактивный отладчик в браузере: При возникновении ошибки в браузере отображается страница с подробным описанием ошибки, стеком вызовов и возможностью просмотра значений переменных в различных точках кода.
* Перезагрузка сервера при изменении кода: Flask автоматически перезапускает сервер при каждом изменении файлов проекта, что позволяет быстро увидеть результаты внесенных изменений без необходимости вручную перезапускать приложение.
* Подробные сообщения об ошибках в консоли: В консоли, где запущен сервер Flask, выводятся подробные сообщения об ошибках, включая информацию о типе ошибки, месте ее возникновения и стеке вызовов.
* Режим отладки был активно использован на этапе разработки и тестирования приложения. Он позволил быстро выявлять и устранять ошибки, связанные с логикой работы приложения, обработкой запросов, взаимодействием с данными и отображением информации в пользовательском интерфейсе.
* Важно: Режим отладки (debug=True) никогда не должен использоваться в production-окружении, поскольку он раскрывает конфиденциальную информацию о приложении и может быть использован злоумышленниками.

**3.2.2. Обработка ошибок и исключений**

В коде приложения (web\_app.py) реализована базовая обработка ошибок с использованием сообщений Flash. Flash-сообщения используются для информирования пользователя об успешных операциях или ошибках.

Например:

* При успешном входе в систему отображается сообщение: flash("Вы успешно вошли в систему.", "success").
* При неверном логине или пароле: flash("Неверный логин или пароль.", "danger").
* При попытке создать проект с пустым названием: flash("Название проекта не может быть пустым!", "warning").
* При попытке добавить пустой комментарий: возвращается JSON-ответ с ошибкой {"error": "Пустой комментарий"} и кодом 400.

Эти сообщения отображаются пользователю в верхней части страницы и автоматически исчезают через некоторое время.

**3.3. Развертывание приложения**

Развертывание приложения "ToDo GeekBrains" представляет собой процесс, делающий приложение доступным для пользователей. В рамках данного проекта рассматриваются два основных варианта развертывания: локальный запуск и развертывание на облачной платформе.

**3.3.1. Локальный запуск (для разработки и тестирования)**

Локальный запуск приложения осуществляется с помощью встроенного веб-сервера Flask.

**Для запуска приложения необходимо:**

1. Установить Python 3 и необходимые зависимости (Flask, jQuery UI, Bootstrap). Зависимости, используемые в проекте.
2. Перейти в корневую директорию проекта (lexus5122-diplom/to\_do\_app).
3. Выполнить команду python web\_app.py (или python3 web\_app.py в зависимости от операционной системы).

После этого приложение будет доступно по адресу http://127.0.0.1:5000/ в веб-браузере.

Этот способ развертывания используется в основном для разработки и тестирования приложения, поскольку встроенный сервер Flask не предназначен для обработки большого количества запросов и не обеспечивает необходимый уровень безопасности для production-окружения.

**3.3.2. Развертывание на облачной платформе (например, Heroku, PythonAnywhere, AWS)**

Для развертывания приложения в production-окружении рекомендуется использовать облачные платформы, такие как Heroku, PythonAnywhere или AWS (Amazon Web Services). Эти платформы предоставляют инфраструктуру и инструменты для развертывания, масштабирования и управления веб-приложениями.

Общий процесс развертывания на облачной платформе включает следующие шаги (детали могут отличаться в зависимости от выбранной платформы):

1. Создание учетной записи на выбранной платформе.

2. Подготовка приложения к развертыванию:

- Создание файла requirements.txt со списком всех зависимостей проекта. Это можно сделать с помощью команды pip freeze > requirements.txt.

- Создание файла Procfile (для Heroku) или аналогичного файла конфигурации для других платформ, в котором указывается команда для запуска приложения. Например, для Heroku файл Procfile может содержать следующую строку: web: gunicorn web\_app:app. Здесь используется Gunicorn – WSGI HTTP-сервер для production.

- Настройка переменных окружения (например, SECRET\_KEY).

- Изменение способа хранения данных, с json на базу данных

3. Развертывание приложения:

- Загрузка кода приложения на платформу (например, с помощью Git).

- Установка зависимостей.

- Запуск приложения.

**Преимущества использования облачных платформ:**

* Масштабируемость: Облачные платформы позволяют легко масштабировать приложение в зависимости от нагрузки.
* Надежность: Облачные платформы обеспечивают высокий уровень доступности и отказоустойчивости.
* Безопасность: Облачные платформы предоставляют различные инструменты для обеспечения безопасности приложения.
* Простота управления: Облачные платформы предоставляют удобные интерфейсы для управления приложением.
* Развертывание на облачной платформе является рекомендуемым способом для production-окружения приложения "ToDo GeekBrains".

# Заключение

В рамках данного дипломного проекта была успешно решена задача разработки веб-приложения "ToDo GeekBrains", предназначенного для управления задачами и проектами. Приложение предоставляет пользователям (как индивидуальным, так и командам) инструменты для организации рабочего процесса, планирования, отслеживания прогресса выполнения задач и совместной работы.

**Достигнутые цели и решенные задачи:**

**В процессе работы над проектом были выполнены все поставленные задачи:**

**Анализ предметной области:** Изучены существующие аналогичные решения (как простые списки задач, так и комплексные системы управления проектами), определены их основные функции, преимущества и недостатки. Это позволило сформулировать требования к разрабатываемому приложению, выбрав оптимальный набор функций, достаточный для эффективного управления задачами, но при этом не перегружающий приложение излишней сложностью.

**Проектирование архитектуры:** Разработана архитектура приложения, основанная на клиент-серверной модели и паттерне MVC (Model-View-Controller). Определены основные компоненты приложения (модели данных, представления, контроллеры), спроектирована структура данных (модели User, Project, Task и их взаимосвязи), выбраны технологии и инструменты для реализации (Python, Flask, HTML, CSS, JavaScript, Bootstrap, jQuery, jQuery UI, JSON, Git).

**Реализация Backend:** Разработана серверная часть приложения на языке Python с использованием микрофреймворка Flask. Созданы API-маршруты (endpoints) для обработки запросов от клиентской части, реализована логика взаимодействия с данными (создание, чтение, обновление и удаление проектов, задач и пользователей), обеспечена аутентификация и авторизация пользователей с разграничением прав доступа (роли "администратор" и "пользователь"), реализовано взаимодействие с хранилищем данных (JSON-файлы). Использованы декораторы @login\_required и @admin\_required для защиты маршрутов, требующих аутентификации и авторизации.

**Реализация Frontend:** Создана клиентская часть приложения с использованием HTML, CSS, JavaScript и библиотек Bootstrap (для стилизации и адаптивности) и jQuery (для динамического обновления контента, AJAX-запросов и реализации drag-and-drop функциональности). Разработаны HTML-шаблоны с использованием шаблонизатора Jinja2, реализована визуализация задач в виде канбан-доски с возможностью перетаскивания задач между столбцами для изменения их статуса.

**Интеграция Backend и Frontend:** Настроено взаимодействие между клиентской и серверной частями приложения посредством обмена данными в формате JSON через API. Реализована обработка ответов сервера и динамическое обновление интерфейса без перезагрузки страницы.

**Тестирование и отладка:** Проведено функциональное тестирование, тестирование API, UI-тестирование и базовое тестирование безопасности приложения. Выявлены и устранены ошибки, связанные с логикой работы, обработкой запросов, отображением данных и пользовательским интерфейсом. Использовался режим отладки Flask (debug=True) для упрощения процесса отладки. Реализована базовая обработка ошибок с использованием flash-сообщений Flask.

**Основные результаты и выводы:**

В результате выполнения дипломного проекта было создано полнофункциональное веб-приложение "ToDo GeekBrains", обладающее следующими ключевыми возможностями:

* Управление проектами: Создание, удаление и просмотр списка проектов.
* Управление задачами: Создание, удаление, изменение статуса задач, назначение исполнителей, просмотр задач в рамках проекта в виде канбан-доски.
* Канбан-доска: Визуализация задач с разделением по статусам ("Планы", "В работе", "Тестирование", "Завершено") и возможностью drag-and-drop.
* Комментирование задач: Добавление и просмотр комментариев к задачам.
* Аутентификация и авторизация: Вход/выход пользователей, разграничение прав доступа (роли "администратор" и "пользователь").
* Управление пользователями: Добавление и удаление пользователей (доступно только администратору).

Приложение разработано с использованием современных веб-технологий и паттернов проектирования, что обеспечивает его масштабируемость, поддерживаемость и удобство использования.

**Недостатки и направления для дальнейшего развития:**

**Несмотря на достигнутые результаты, приложение имеет ряд недостатков и потенциальных точек роста:**

* **Безопасность:** Хранение паролей в открытом виде и отсутствие защиты от CSRF-атак являются серьезными уязвимостями. Необходимо реализовать хеширование паролей и защиту от CSRF.
* **Тестирование:** Проведённого тестирования недостаточно. Необходимо использовать специализированные фреймворки для автоматизации тестирования API и UI.
* **Хранение данных:** Использование JSON-файлов для хранения данных подходит только для небольших приложений с ограниченным количеством пользователей и данных. Для более крупных проектов следует использовать полноценную базу данных (например, PostgreSQL, MySQL).
* **Редактирование проектов и задач:** Функции добавления есть, а редактирования проектов и задач, реализованные в коде, в явном виде не представлены в интерфейсе.
* **Уведомления:** Отсутствует система уведомлений о событиях (например, о назначении новой задачи, изменении статуса, добавлении комментария).
* **Развертывание:** В рамках дипломного проекта был описан процесс локального развертывания и общие принципы развертывания на облачной платформе, однако не было предоставлено подробной инструкции по развертыванию на конкретной платформе.
* **Ограничение drag-and-drop:** Задачи можно перетаскивать в любой статус, что нарушает логику процесса (например, из "Планы" сразу в "Завершено"). Желательно ограничить возможные переходы между статусами.

**В дальнейшем приложение может быть улучшено путем реализации:**

* Хеширования паролей и защиты от CSRF.
* Автоматизированного тестирования.
* Использования реляционной базы данных.
* Функций редактирования проектов и задач.
* Системы уведомлений.
* Подробной инструкции по развертыванию на облачной платформе (например, Heroku).
* Ограничений на переходы между статусами задач на канбан-доске.
* Более детальной проработки UI/UX дизайна.
* Добавления функциональности фильтрации и поиска задач.
* Реализации загрузки файлов к задачам.
* Интеграции с внешними сервисами (например, календарями, почтовыми клиентами).

В целом, разработанное веб-приложение "ToDo GeekBrains" представляет собой работающее решение для управления задачами и проектами, которое может быть использовано как для персонального планирования, так и для совместной работы в небольших командах. Приложение имеет потенциал для дальнейшего развития и улучшения. Проделанная работа демонстрирует владение навыками fullstack-разработки и понимание принципов создания современных веб-приложений.

# Список используемой литературы

* The Flask Mega-Tutorial / M. Grinberg // Miguel Grinberg's blog. – 2017. – URL: <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
* Документация Flask // Flask Documentation. – URL: <https://flask.palletsprojects.com/>
* Документация Jinja2 // Jinja Documentation. – URL: <https://jinja.palletsprojects.com/en/3.1.x/>
* Документация Bootstrap // Bootstrap Documentation. – URL: <https://getbootstrap.com/docs/4.5/getting-started/introduction/>
* Документация jQuery // jQuery API Documentation. – URL: <https://api.jquery.com/>
* Документация jQuery UI // jQuery UI. – URL: <https://jqueryui.com/>
* MDN Web Docs // Mozilla Developer Network. – URL: <https://developer.mozilla.org/ru/>
* Лутц М. Изучаем Python / М. Лутц. — 5-е изд. — СПб.: Символ-Плюс, 2019. Flask Web Development with Python Tutorial / Real Python. – URL: <https://realpython.com/flask-by-example-part-1-project-setup/>
* Krug S. Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability / S. Krug. — 3rd ed. — New Riders, 2014.
* Документация по безопасности Flask // Flask Security Documentation. – URL: <https://flask-security.readthedocs.io/>
* Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. — Addison-Wesley Professional, 1994.
* Martin R. Clean Code: A Handbook of Agile Software Craftsmanship / R. Martin. — Prentice Hall, 2008.
* Документация Python по работе с файловой системой // Python Documentation. – URL: <https://docs.python.org/3/library/filesys.html>
* Документация по управлению сессиями Flask // Flask Session Documentation. – URL: <https://flask-session.readthedocs.io/>

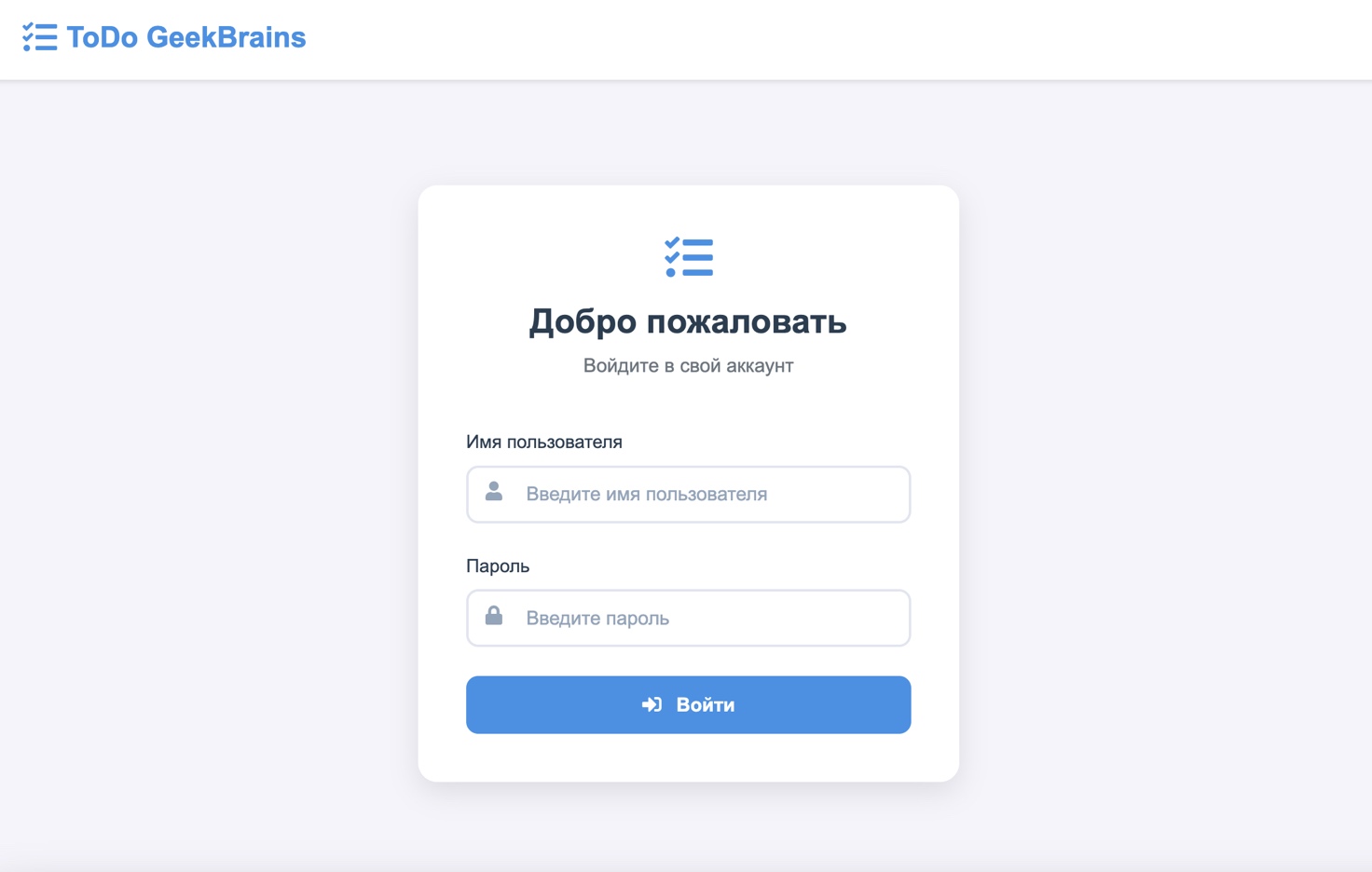
# Приложения

1. **Структура проекта**

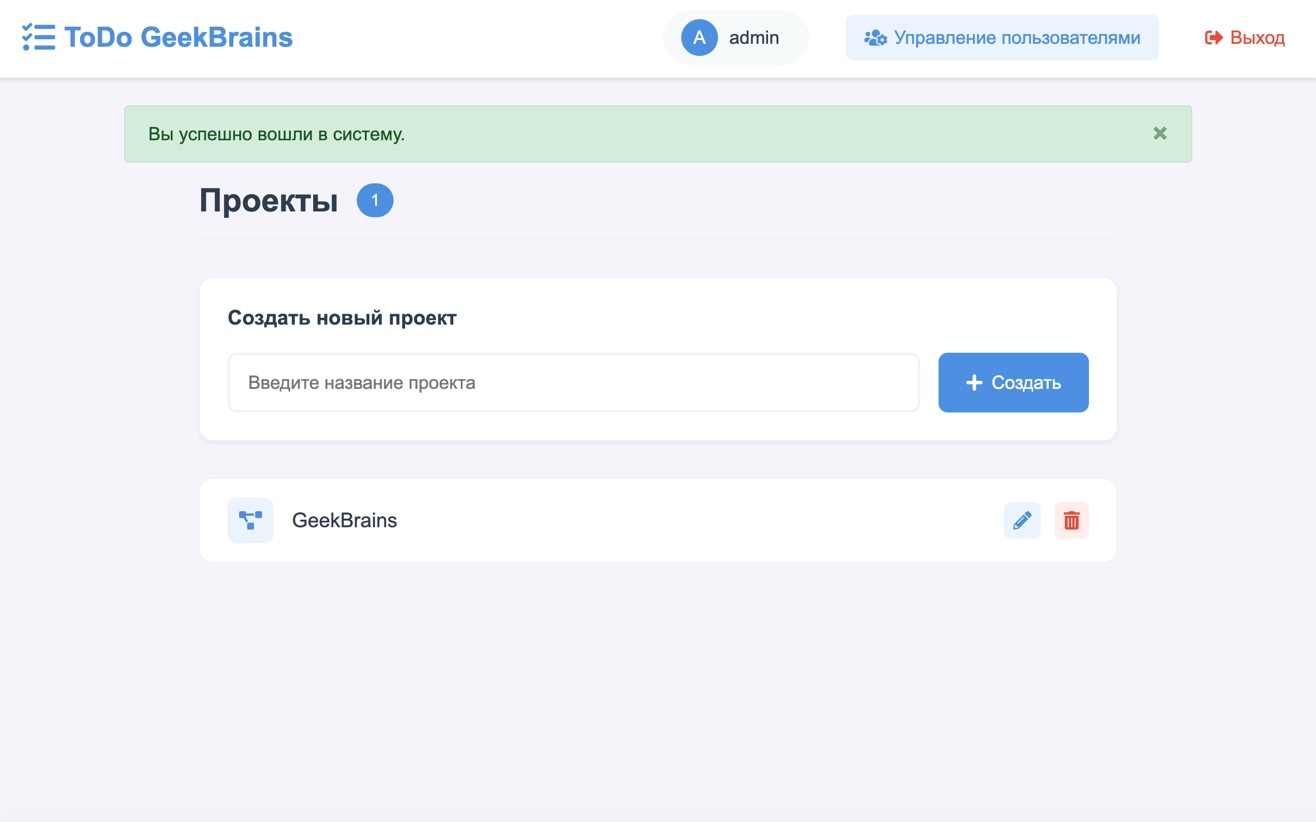
Directory structure:

* └── lexus5122-diplom/
* ├── screen/
* └── to\_do\_app/
* ├── main.py
* ├── projects.json
* ├── tasks.json
* ├── users.json
* ├── web\_app.py
* ├── .DS\_Store
* ├── cli/
* │ ├── \_\_init\_\_.py
* │ └── cli.py
* ├── models/
* │ ├── \_\_init\_\_.py
* │ ├── project.py
* │ ├── task.py
* │ ├── todo\_list.py
* │ └── user.py
* ├── storage/
* │ ├── \_\_init\_\_.py
* │ ├── file\_storage.py
* │ └── file\_storage\_users.py
* └── templates/
  + ├── base.html
  + ├── index.html
  + ├── login.html
  + ├── project\_board.html
  + ├── project\_list.html
  + └── user\_management.html

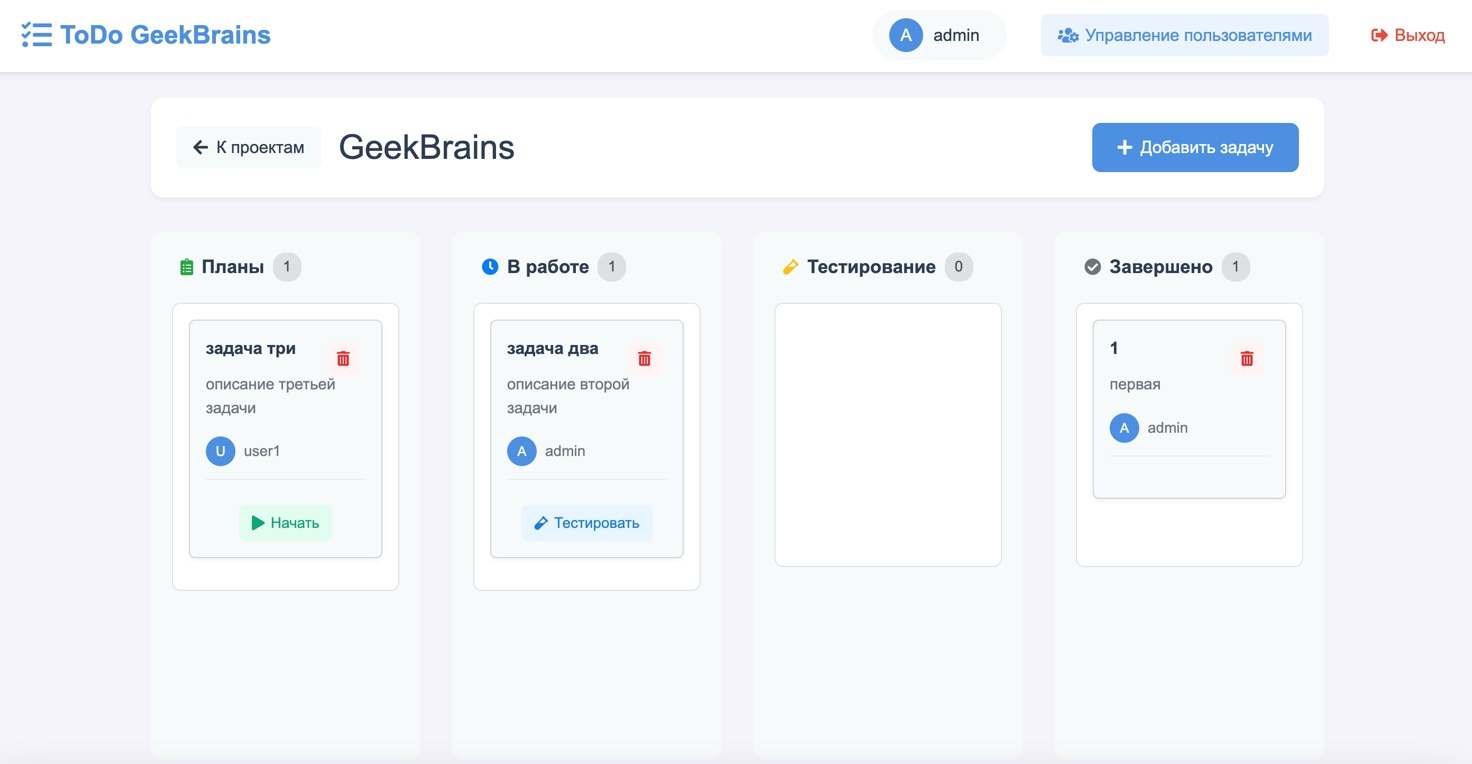
1. **Скриншоты рабочего приложения:**
2. Страница логина:



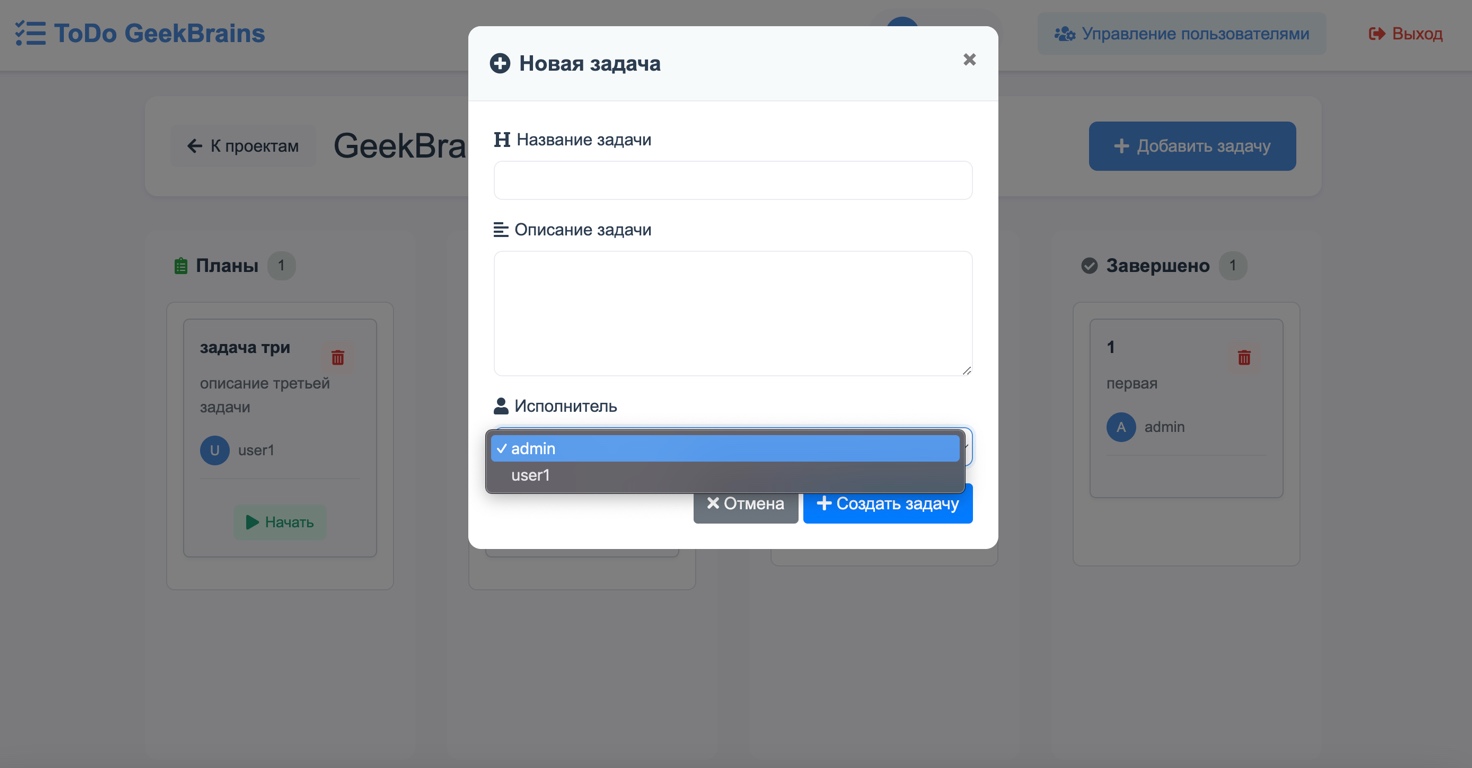
1. Раздел проекты:



1. Раздел задачи:



1. Добавление задачи:



1. Управление пользователями (для администратора) :

