**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА АНАЛИЗА ДАННЫХ И ТЕХНОЛОГИЙ ПРОГРАММИРОВАНИЯ

Направление: 09.03.03 – «Прикладная информатика»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**СИСТЕМА КОНТРОЛЯ ВЕРСИЙ**

**Работа завершена:**

Студент 4 курса

группы 09-051

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Мишин С.С.

**Работа допущена к защите:**

Научный руководитель

ст. преподаватель

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г.    \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сафина Л.И.

Казань-2024

Содержание

[Глоссарий 2](#_Toc164279791)

[Введение 3](#_Toc164279792)

[1. Анализ предметной области 4](#_Toc164279793)

[1.1. Особенности предметной области 4](#_Toc164279794)

[1.2. Обзор существующих решений 5](#_Toc164279795)

[1.3. Формирование технического задания 10](#_Toc164279796)

[2. Проектирование приложения 17](#_Toc164279797)

[2.1. Проектирование системы хранения данных 17](#_Toc164279798)

[2.2. Выбор инструментов и средств разработки 18](#_Toc164279799)

[3. Разработка веб-приложения 19](#_Toc164279800)

[3.1. Реализация системы хранения данных 19](#_Toc164279801)

[3.2. Реализация пользовательского интерфейса 19](#_Toc164279802)

[3.3. Реализация функциональной части приложения 24](#_Toc164279803)

[4.1. Тестирование пользовательского интерфейса 25](#_Toc164279804)

[4.2. Тестирование функциональной части приложения 25](#_Toc164279805)

[Заключение 29](#_Toc164279806)

[Список использованных источников 30](#_Toc164279807)

[Приложение 1 31](#_Toc164279808)

[Приложение 2 32](#_Toc164279809)

[Приложение 3 35](#_Toc164279810)

# Глоссарий

Blob – (binary large object) – бинарный объект, представляющий содержимое файла.

Hash - уникальный идентификатор объекта. Вычисляется на основе содержимого объекта, и он служит для обеспечения целостности данных и проверки изменений объекта.

Hash – функция – это функция, которая принимает входные данные произвольной длины и преобразует их в некоторое фиксированное значение фиксированной длины, известное как hash-значение или hash-сумма. Это значение обычно представляется в виде последовательности чисел и/или букв, которые выглядят как случайная строка.

IDE – (integrated development environment) интегрированная среда разработки. Блокнот для программистов.

VCS – (version control system) система контроля версий.

.gitignore – файл в котором описаны все файлы и папки, которые VCS должна игнорировать (не хранить изменения и вообще ничего с ними не делать).

Ветка (branch) - перемещаемый указатель на один из коммитов. Ветка указывает на последний коммит в своей цепочке коммитов, который называется «головным коммитом» (head commit). Создание новой ветки создает копию текущей ветки, в которой вы находитесь, и дает вам возможность работать над изменениями в изолированном пространстве без влияния на другие ветки.

Дерево – объект, который представляет собой папку или каталог файловой системы. Хранит имя каталога и ссылки на содержимое данного каталога.

Коммит – snapshot (копия хранимых данных) проекта в определенный момент времени. Коммит хранит в себе ссылку на дерево проекта, автора коммита, время создания, комментарий и ссылку на предыдущий коммит.

Репозиторий - центральное хранилище для всех файлов, данных и истории изменений проекта. В репозитории хранятся все версии файлов, а также информация об изменениях, коммитах, ветках и тегах. Может быть как локальным (на устройстве клиента), так и удаленным (на удаленном от клиента сервере).

Файл – любой объект файловой системы (кроме каталогов). Хранит имя и ссылку на Blob.

# Введение

В современном мире разработка программного обеспечения становится все более сложной и коллаборативной задачей, требующей эффективного управления изменениями в исходном коде. Системы контроля версий (VCS) играют ключевую роль в этом процессе, предоставляя механизмы отслеживания, управления и совместной работы над версиями программного продукта.

Тема разработки новой системы контроля версий становится важной, учитывая постоянное развитие технологий и появление новых требований к процессам разработки. Перспективы улучшения эффективности совместной работы, безопасности данных и интеграции с современными средами разработки делают актуальной не только оптимизацию уже существующих VCS, но и создание новых, инновационных систем.

Цель: Создать систему контроля версий.

Задачи:

1. Изучить требования к системе контроля версий

2. Проанализировать существующие системы контроля версий

3. Составить техническое задание

4. Спроектировать части будущей системы контроля версий

5. Реализовать спроектированную систему

# 1. Анализ предметной области

# 1.1. Особенности предметной области

Системы контроля версий являются инструментом, помогающим разработчикам отслеживать изменения в версиях своих файлов и организовывать совместную работу нескольких разработчиков над одним проектом. Предметная область, связанная с VCS задает определенные требования к данным системам.

Основными особенностями предметной области, которые следует учитывать при разработке VCS, являются:

1. Хранение данных файлов и папок (storing). Одной из основных проблем, с которой сталкиваешься при создании VCS – это принцип хранения данных и состояний.
2. Отслеживание изменений (tracking). Отслеживаются изменения файлов и папок, изменение их содержимого, имени, или расположения в файловой системе. Существуют различные способы сравнения содержимого файлов при помощи hash функций и diff-алгоритмов для выявления отличий.
3. История изменений (history). Просмотр всех версий файлов и самого проекта. Это возможно при перемещении к нужному коммиту. Необходима функция для просмотра истории коммитов.
4. Откат (reverting). Возможность откатить изменения до предыдущего состояния. Если изменения в коде привели к проблемам или ошибкам, разработчики могут использовать откат, чтобы вернуться к предыдущему рабочему состоянию проекта.
5. Игнорирование определенных файлов (ignoring). Не все файлы проекта обычно нужно отслеживать. Некоторые файлы слишком большие и не изменяются (например, библиотеки, файлы IDE), некоторые могут содержать конфиденциальные данные (пароли, переменные среды и т.д.). Такие файлы можно игнорировать, путем добавления их в .gitignore файл.
6. Работа в распределенном режиме (distributing). Это означает, что каждый клиент имеет полную копию репозитория. Это обеспечивает возможность работы независимо от сетевого подключения к центральному серверу.
7. Работа с удаленными репозиториями (remote repo). Возможность синхронизировать локальный репозиторий с удаленными репозиториями, такими как GitHub, GitLab или Bitbucket. Это позволяет разработчикам делиться своим кодом, работать в команде и делать резервные копии своего кода.
8. Ветвление (branching). Ветвление позволяет разработчикам работать над различными версиями проекта параллельно, а затем объединять их изменения обратно в основную ветку.
9. Слияние (merging): Это процесс объединения изменений из одной ветки с другой. После того как изменения в одной ветке были завершены и протестированы, их можно объединить с другой веткой, чтобы интегрировать эти изменения в основной поток разработки.
10. Отслеживание проблем и задач (tasks). Некоторые VCS интегрированы с системами управления задачами, такими как GitHub Issues или Jira. Это позволяет разработчикам связывать изменения в коде с определенными проблемами или задачами, что упрощает отслеживание прогресса и взаимодействие в команде.
11. Аудит и безопасность (security). VCS обеспечивают аудит изменений, что позволяет организациям следить за тем, кто и когда вносил изменения в код или другие файлы проекта. Также они могут предоставлять механизмы контроля доступа, чтобы управлять правами доступа к репозиториям и файлам.

# 1.2. Обзор существующих решений

1. Приложение «ЛидерТаск».

Рассмотрим особенности этого приложения. Часть пользовательского интерфейса представлена на рисунке 1.

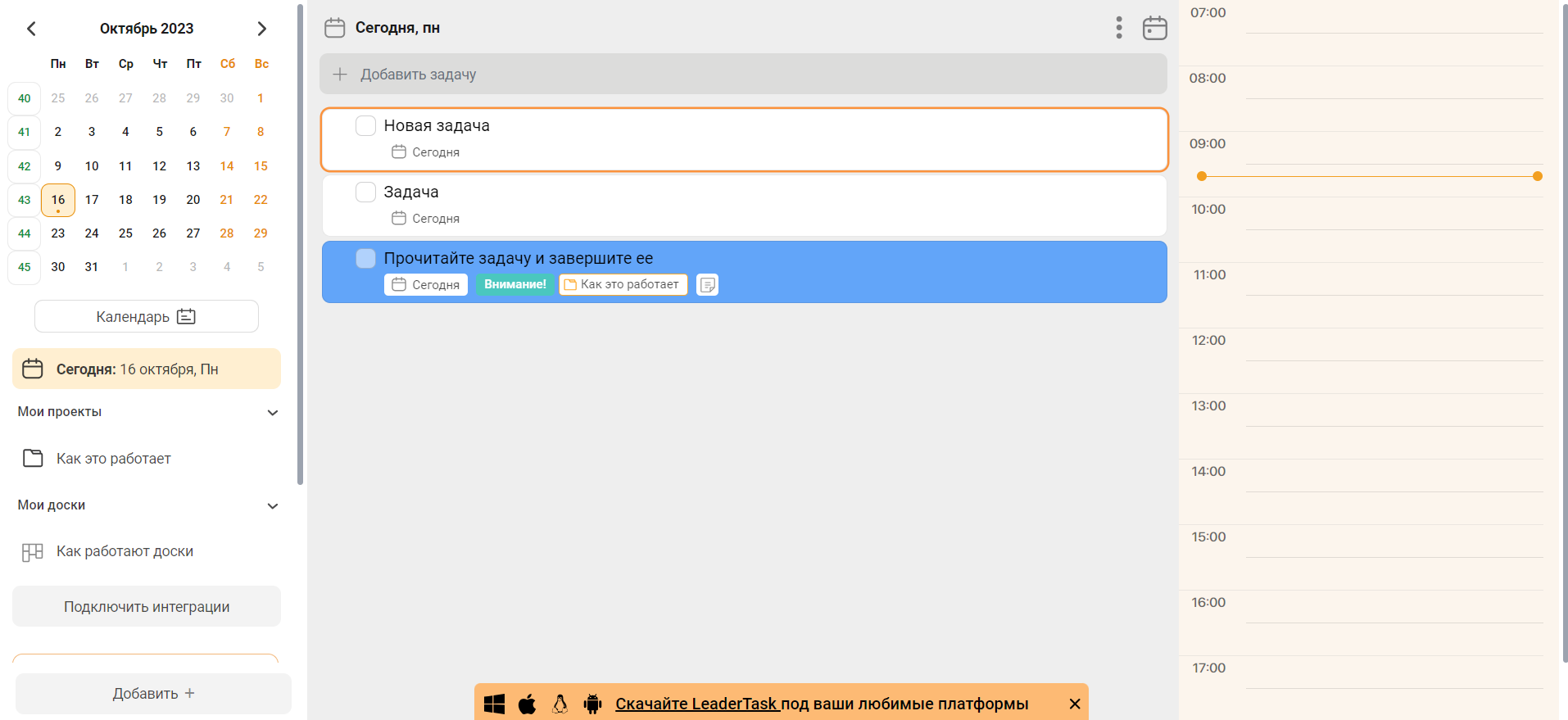


Рисунок 1. Скриншот приложения «ЛидерТаск»

Плюсы данного приложения:

Плюсы «ЛидерТаск»:

1. Удобный интерфейс: Пользовательский интерфейс «ЛидерТаск» интуитивно понятен и легок в использовании. Это позволяет пользователям быстро разобраться с приложением.
2. Поддержка множества платформ: «ЛидерТаск» доступен на различных платформах, включая веб-версию, мобильные приложения и настольное приложение. Это обеспечивает доступность и синхронизацию задач на разных устройствах.
3. Гибкость настроек: Пользователи могут настраивать задачи, группировать их, устанавливать приоритеты и дедлайны, что позволяет создавать персональные системы управления задачами.
4. Отчеты и статистика: «ЛидерТаск» предоставляет возможность генерации отчетов и анализа продуктивности, что помогает пользователям отслеживать, как они используют свое время.

Минусы «ЛидерТаск»:

1. Платное использование: Большинство функций «ЛидерТаск» доступны только в платных версиях. Это может ограничивать доступность для некоторых пользователей.
2. Недостаточная гибкость в бесплатной версии: Бесплатная версия «ЛидерТаск» имеет ограниченные возможности по настройке и синхронизации задач.
3. Недостаточно продвинутые функции проектного управления: В сравнении с некоторыми другими инструментами для проектного управления, «ЛидерТаск» может предоставлять более ограниченные возможности для сложных проектов.
4. Не всегда интуитивный процесс настройки: Некоторые пользователи могут столкнуться с трудностями в начальной настройке «ЛидерТаск» из-за многообразия опций и функций.

2. Приложение «Trello».

Часть пользовательского интерфейса представлена на рисунке 2.

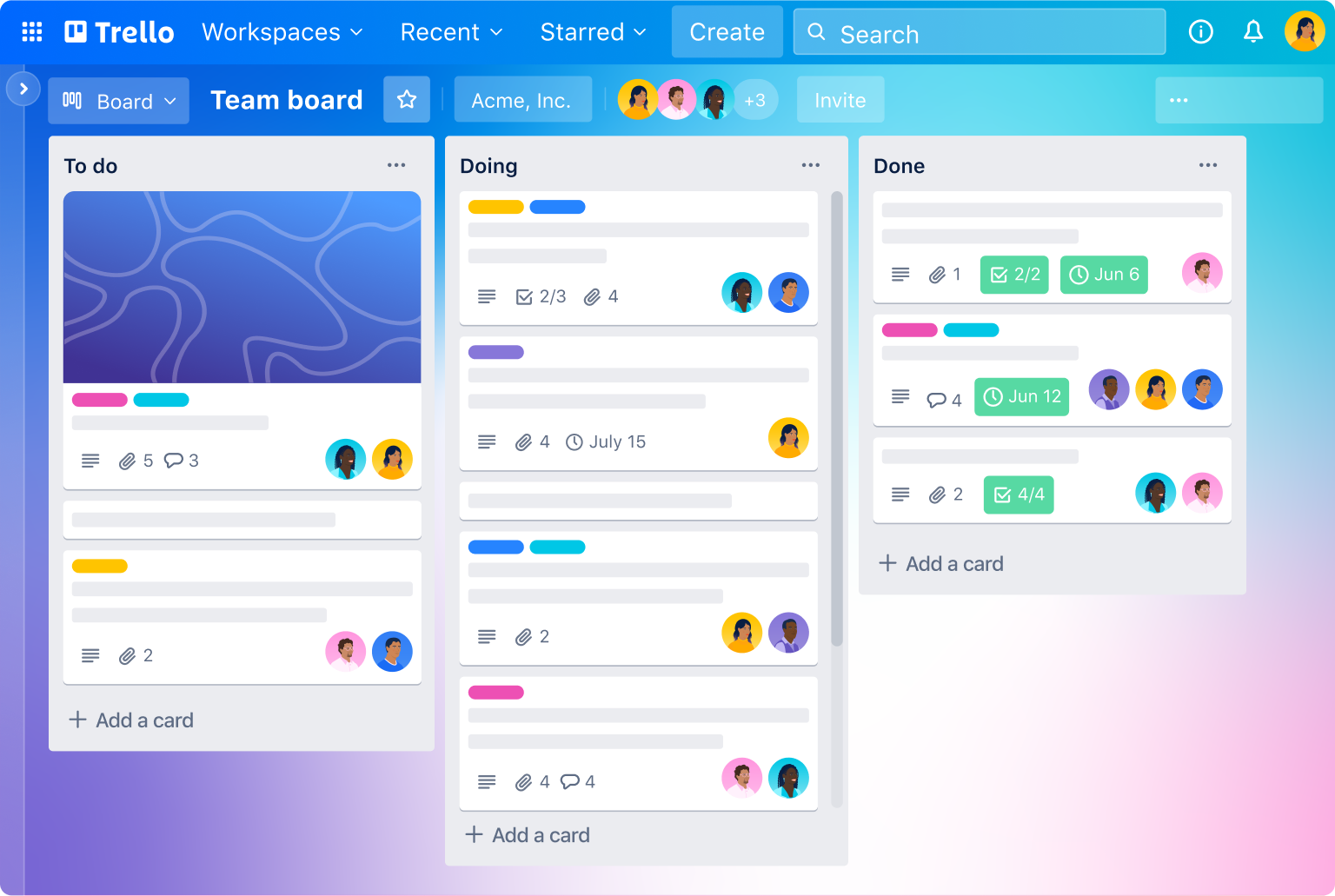


Рисунок 2. Скриншот приложения «Trello»

Плюсы «Trello»:

1. Интуитивный интерфейс: «Trello» предлагает простой и интуитивно понятный интерфейс, основанный на системе карточек и досок. Это делает его легким в освоении даже для новых пользователей.
2. Бесплатная версия: «Trello» предоставляет бесплатный доступ с базовыми функциями, что делает его доступным для многих пользователей.
3. Гибкость в организации задач: Пользователи могут создавать карточки, группировать их в списки, присваивать метки и сроки выполнения. Это обеспечивает гибкость в организации задач и проектов.

Минусы «Trello»:

1. Недоступность в России
2. Ограничения в бесплатной версии: Бесплатная версия «Trello» имеет ограничения по количеству вложений, интеграций и другим функциям, что может быть недостаточно для более сложных проектов.
3. Недостаточная автоматизация: «Trello» может потребовать дополнительных действий для автоматизации процессов, в отличие от некоторых других инструментов управления задачами.

3. Далее рассмотрим приложение «Todoist».

Часть пользовательского интерфейса представлена на рисунке 3.

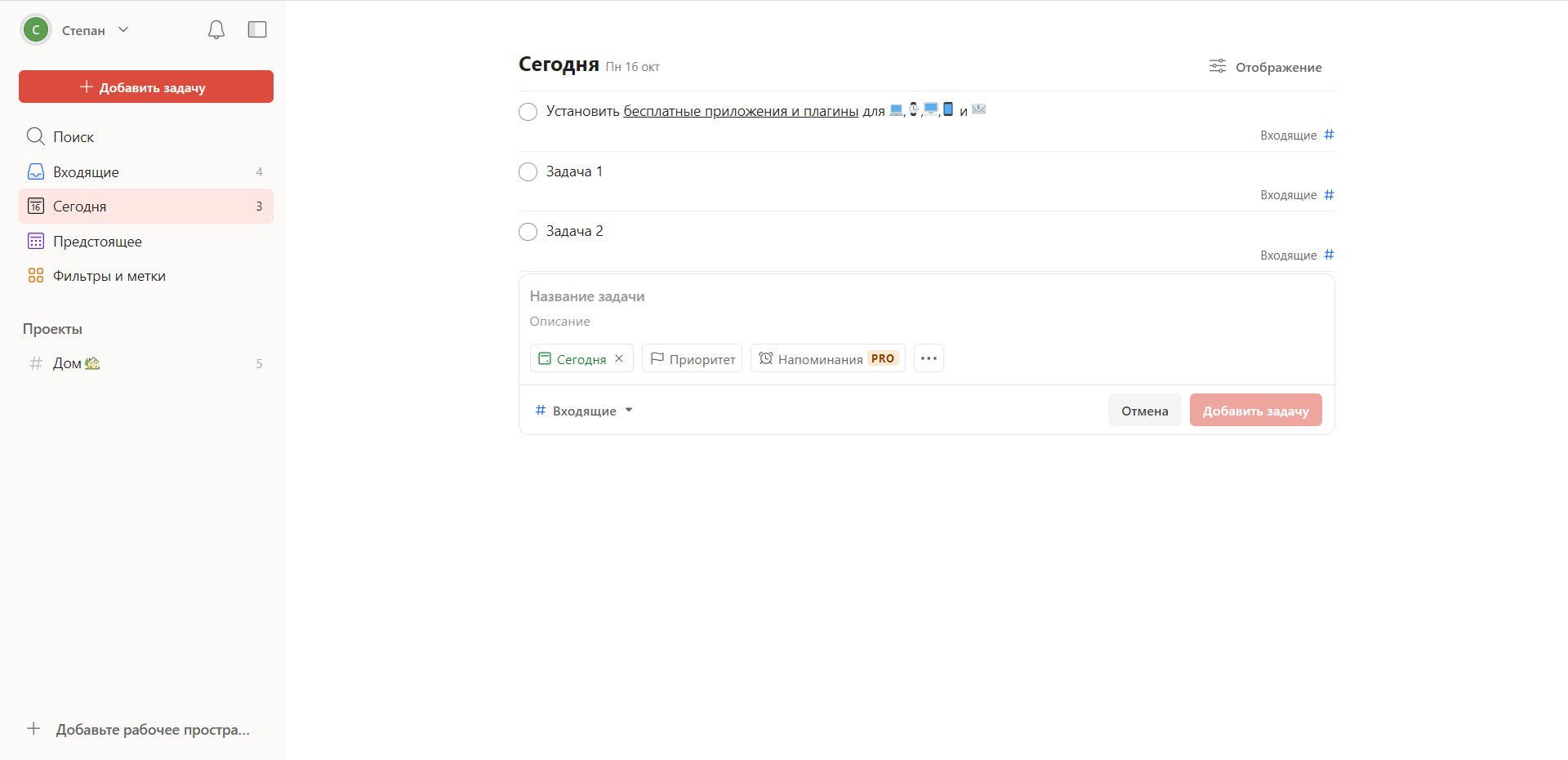


Рисунок 3. Скриншот приложения «Todoist»

Плюсы «Todoist»:

1. Интуитивный интерфейс: «Todoist» предлагает простой и понятный интерфейс, что делает его легким в освоении для новых пользователей.
2. Поддержка множества платформ: «Todoist» доступен на разных устройствах, включая веб-версию, мобильные приложения и настольное приложение, и позволяет синхронизировать задачи между ними.
3. Умные фильтры и поиск: «Todoist» предоставляет возможность создавать умные фильтры и использовать мощный поиск для быстрого доступа к задачам.

Минусы «Todoist»:

1. Ограничения в бесплатной версии: Бесплатная версия «Todoist» имеет ограничения по количеству проектов и функциональности, и для доступа ко всем возможностям требуется подписка.
2. Не всегда подходит для сложных проектов: Для более сложных проектов и задач, требующих дополнительных функций управления проектами, «Todoist» может оказаться ограниченным.
3. Недостаточные инструменты для командной работы: В сравнении с некоторыми другими инструментами, «Todoist» может предоставлять более ограниченные возможности для командной работы над проектами.

Исходя из обзора рынка планировщиков задач и инструментов управления временем, можно сделать следующие выводы:

1. Рынок планировщиков задач разнообразен: Существует множество планировщиков задач с различными функциональными возможностями и подходами к управлению задачами. Это позволяет пользователям выбирать инструмент, наиболее соответствующий их потребностям и стилю работы.

2. Интуитивность и доступность ключевы: Многие успешные планировщики задач отличаются простым и интуитивным интерфейсом, что делает их доступными для широкой аудитории, включая новых пользователей.

3. Бесплатные версии и подписки: Многие планировщики предоставляют бесплатные версии со значительными ограничениями, а дополнительные функции доступны через платные подписки. Это делает инструменты доступными для разных бюджетов.

4. Гибкость и настраиваемость: Пользователи ценят возможность настройки планировщика задач в соответствии с их собственными потребностями, включая организацию задач, установку приоритетов, меток и сроков выполнения.

5. Аналитика и отчеты становятся более важными: Возможность анализа и мониторинга продуктивности важна для пользователей, что отражается в развитии функционала для создания отчетов и статистики.

Вывод состоит в том, что рынок планировщиков задач предлагает множество вариантов для пользователей с различными потребностями. Важно также учитывать, что лучший инструмент для одного пользователя может не подходить другому, и выбор должен зависеть от конкретных требований. В конечном итоге, для выделения среди конкурентов и создания успешного планировщика задач, важно предложить уникальные функции и решения, учесть потребности пользователей и предоставить высококачественный продукт с акцентом на удобстве и эффективности использования, а также сделать бесплатную версию достаточно функциональной, чтобы не отпугивать обычных пользователей необходимостью подписки.

# 1.3. Формирование технического задания

1.3.1. Предмет разработки

Предметом разработки является веб-приложение «Планировщик задач», предоставляющее онлайн площадку для создания, редактирования и управления задачами.

Назначение веб-приложения:

* Предоставление пользователям возможности создания, редактирования и удаления задач;
* Предоставление пользователям возможности создания и удаления собственных категорий для задач;
* Возможность редактирования личных данных в профиле пользователя;
* Возможность подключения уведомлений о приближающихся задачах;
* Возможность просмотра статистики по количеству задач в категориях и их выполнении;

Цель разработки веб-приложения "Планировщик задач" заключается в предоставлении инструмента, который помогает пользователям эффективно управлять своим временем и задачами, повышать производительность и улучшать качество жизни, как в рабочем, так и в личном контексте.

1.3.2. Требования к графическому интерфейсу системы

Требования к графическому интерфейсу (ГИ) веб-приложения "Планировщик задач" являются критическими, поскольку удобство использования и привлекательный дизайн могут существенно повлиять на опыт пользователя. Вот список ключевых требований к ГИ:

1. Интуитивность: ГИ должен быть интуитивно понятным, что позволит пользователям быстро разобраться в его функциональности без необходимости в долгой инструкции.

2. Простота: Дизайн должен быть минималистичным и не перегруженным лишними элементами. Это сделает приложение более понятным и удобным.

3. Цветовая палитра и стиль: Выбор подходящей цветовой палитры и стиля, соответствующие общей атмосфере приложения и легко воспринимаемые.

4. Типография: Использование четкого и читаемого шрифта для текста и заголовков, чтобы улучшить читаемость.

5. Расположение элементов: Размещение элементов интерфейса логично и удобно для пользователя.

6. Адаптивный дизайн: ГИ должен быть адаптивным и подстраиваться под разные разрешения экранов, обеспечивая хороший пользовательский опыт на различных устройствах.

7. Интерактивность: Наличие анимаций и интерактивных элементов, чтобы сделать приложение более привлекательным и динамичным.

8. Иконки и графика: Использование качественных иконок и графики, которые ясно отображают функциональность элементов.

9. Пользовательский опыт (UX): Необходимо учесть потребности и ожидания пользователей в процессе взаимодействия с приложением, упрощая и оптимизируя процессы.

10. Инструкции и подсказки: Предоставление пользователю информацию и подсказки о функциональности приложения, особенно если есть сложные или неочевидные функции.

Эти требования помогут создать графический интерфейс, который не только будет привлекательным и современным, но и будет способствовать легкости и комфорту использования веб-приложения "Планировщик задач".

1.3.3. Функциональные требования

User Stories - короткие и простые описания функциональности продукта с точки зрения конечного пользователя:

* UC1-1 Регистрация в веб-приложении.

Как новый пользователь, я хочу зарегистрироваться в приложении, чтобы создать свой профиль и начать пользоваться сервисом.

Критерии приемки:

1. Пользователь должен иметь возможность выбрать опцию "Регистрация" на главном экране приложения.
2. Пользователь должен ввести свои данные, включая логин, адрес электронной почты и пароль.
3. При вводе адреса электронной почты, система должна проверить его уникальность.
4. После успешной регистрации, пользователь попадает в личный кабинет.

* UC1-2 Авторизация в приложении.

Как зарегистрированный пользователь, я хочу иметь возможность войти в приложение, используя свой логин и пароль.

Критерии приемки:

1. Пользователь должен иметь возможность ввести свой логин и пароль на экране входа.
2. Система должна проверить правильность введенных данных.
3. Пользователь может воспользоваться функцией восстановления пароля, если забыл его.

* UC2-1 Создание задачи.

Как пользователь, я хочу создавать новые задачи, чтобы планировать и управлять ими.

Критерии приемки:

1. Пользователь должен иметь возможность создавать новую задачу, указывая ее название, описание, срок выполнения и категорию.
2. Созданная задача отображается в указанной категории

* UC2-2 Редактирование задачи.

Как пользователь, я хочу иметь возможность редактировать созданные задачи.

Критерии приемки:

1. Пользователь может изменить название, описание, срок выполнения, категорию задачи.

* UC2-3 Удаление задачи.

Как пользователь, я хочу иметь возможность удалять задачи.

Критерии приемки:

1. Пользователь должен иметь возможность удалить задачу, что приведет к ее полному удалению из системы.

* UC3-1 Создание категории для задач.

Как пользователь, я хочу создавать категории, чтобы организовывать задачи по определенным критериям.

Критерии приемки:

1. Пользователь должен иметь возможность создавать новые категории и задавать им название.
2. Пользователь может присваивать задачам созданные категории.

* UC3-2 Удаление категории для задач.

Как пользователь, я хочу иметь возможность удалять созданные категории.

Критерии приемки:

1. Пользователь должен иметь возможность удалить категорию, что приведет к удалению всех задач, связанных с данной категорией.

* UC4-1 Редактирование данных пользователя.

Как пользователь, я хочу иметь возможность редактировать свои личные данные.

Критерии приемки:

1. Пользователь может изменить свою почту, фото профиля, и прочие личные данные.
2. Обновленные данные должны отобразиться в профиле после сохранения.

* UC5-1 Подключение уведомлений о задачах в Telegram.

Как пользователь, я хочу иметь возможность получать уведомления о задачах в мессенджере Telegram.

Критерии приемки:

1. Пользователь должен иметь возможность настроить интеграцию с Telegram.
2. Пользователь получает уведомления о задачах в своем чате в Telegram о предстоящих задачах.

* UC6-1 Просмотр статистики задач.

Как пользователь, я хочу иметь возможность просматривать статистику выполненных и невыполненных задач.

Критерии приемки:

1. Пользователь может видеть статистику выполненных задач, а также те, которые находятся в работе.
2. Пользователь может просматривать статистику задач по категориям и по дате.

Эти функциональные требования помогут определить основной функционал приложения и обеспечат пользователей необходимыми функциями для управления задачами и профилем в системе.

Use Case диаграмма показана на диаграмме на рисунке 4.

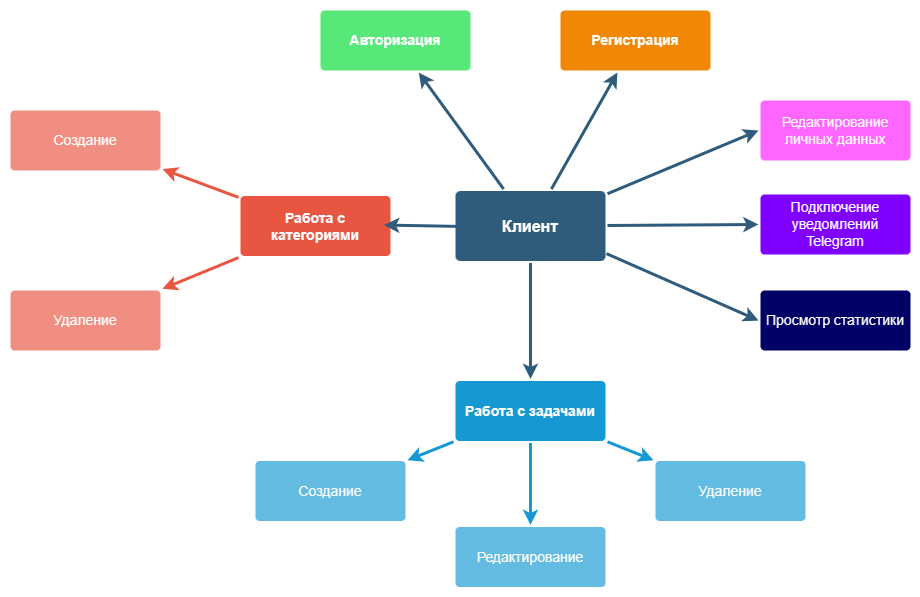


Рисунок 4. Диаграмма вариантов использования

1.3.4. Нефункциональные требования

Для приложения определены следующие нефункциональные требования:

* Безопасность: Приложение должно обеспечивать безопасность данных пользователей и соблюдать стандарты безопасности, включая защиту от несанкционированного доступа, шифрование данных и механизмы аутентификации.
* Доступность: Приложение должно быть доступным для пользователей в любое время, с минимальными периодами недоступности.
* Адаптивность: Приложение должно открываться и быть удобным при использовании на любых устройствах с любым размером экрана.
* Производительность: Приложение должно обеспечивать высокую производительность и отзывчивость даже при большом количестве пользователей и задач.
* Масштабируемость: Система должна быть легко масштабируемой для возможности увеличения числа пользователей и задач без значительных изменений в архитектуре.
* Совместимость: Приложение должно быть совместимо с различными браузерами (Chrome, Firefox, Safari, Edge) и мобильными устройствами (iOS, Android).
* Удобство использования (Usability): Дизайн интерфейса должен быть интуитивным, с учетом лучших практик пользовательского опыта (UX). Пользователи должны легко осваивать приложение.
* Скорость загрузки: Страницы приложения должны быстро загружаться, даже при медленном интернет-соединении.

# 2. Проектирование приложения

# 2.1. Проектирование системы хранения данных

В качестве базы данных выбор был отдан SQLite. SQLite - это встраиваемая БД, которая не требует отдельного сервера и настройки. Ее использование обычно не требует сложных операций по установке и настройке. Это упрощает процесс разработки и развертывания приложения. Также из плюсов можно отметить легкость обслуживания, поддержки, отказоустойчивость.

Пользователи будут хранится в User, дополнительная информация о них в Profile, список категорий в Category и список задач в ToDo.

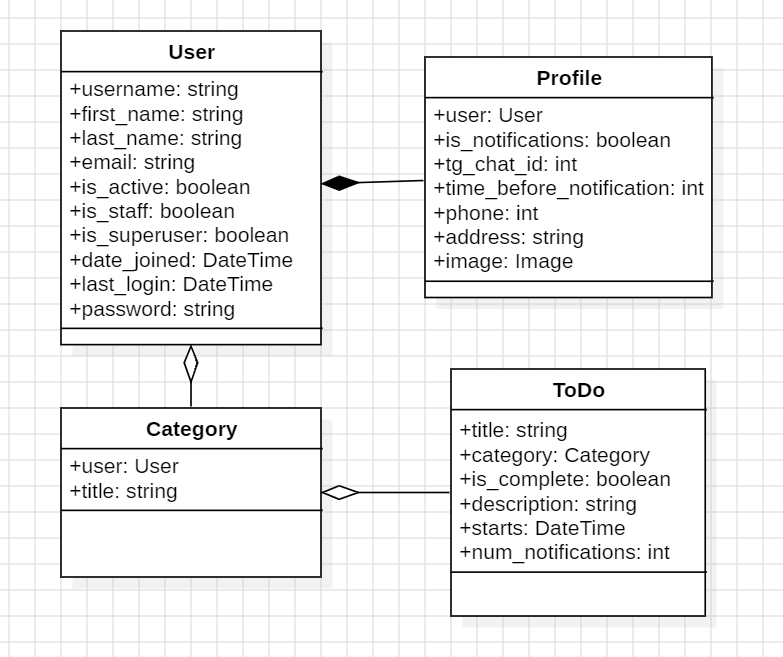


Рисунок 5. Структура БД

# 2.2. Выбор инструментов и средств разработки

В качестве средств для разработки были выбраны:

1. Язык программирования – Python.
2. Django – фреймворк для создания веб-приложений.
3. Интегрированная среда разработки Pycharm.
4. Sqlite – БД для хранения данных.
5. MDBootstrap - библиотека, предоставляющая набор стильных и адаптивных компонентов пользовательского интерфейса (UI) в соответствии с Material Design.
6. Chart.js - библиотека для создания интерактивных и красочных графиков и диаграмм в веб-приложениях.
7. Aiogram - это фреймворк для разработки Telegram-ботов.
8. Celery - асинхронная очередь задач для Python. Он позволяет выполнять задачи в фоновом режиме, улучшая производительность вашего приложения.
9. Selenium – для автоматизированного тестирования.

# 3. Разработка веб-приложения

# 3.1. Реализация системы хранения данных

В Django для создания таблиц в базе данных SQLite используется механизм миграций (migrations). Миграции позволяют вам определить структуру вашей базы данных с использованием моделей Django и затем автоматически создать или изменить таблицы базы данных в соответствии с вашими моделями. Сначала для каждой таблицы необходимо создать модель, после чего можно применить миграции и эти таблицы будут созданы в БД. Листинг кода создания таблиц можно посмотреть в Приложении 1.

# 3.2. Реализация пользовательского интерфейса

Для разработки графического интерфейса были использованы HTML, CSS, JavaScript и библиотека MDBootstrap. С помощью HTML создается структура страницы, далее, добавляются стили с помощью CSS и классов из MDBootstrap, а с помощью JavaScript создаются динамические элементы и события.

При открытии веб-приложения отобразится страница входа с полями для ввода данных, кнопка войти и ссылка для регистрация нового аккаунта (рис 6.).

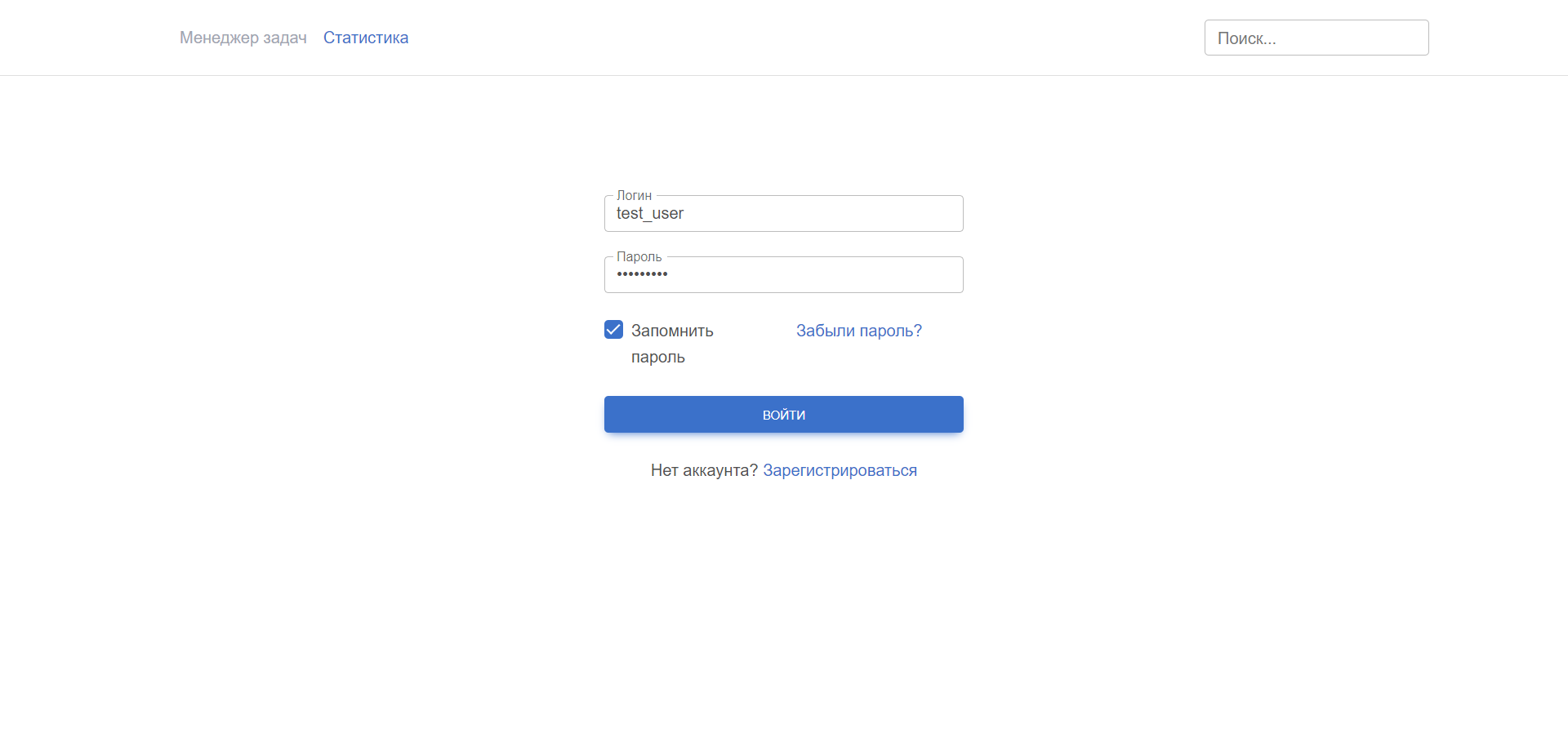


Рисунок 6. Страница входа

Чтобы попасть на главную страницу нужно пройти авторизацию или регистрацию, страница которой выглядит следующим образом (рис 7). На ней находятся поля для данных аккаунта и кнопка регистрации.

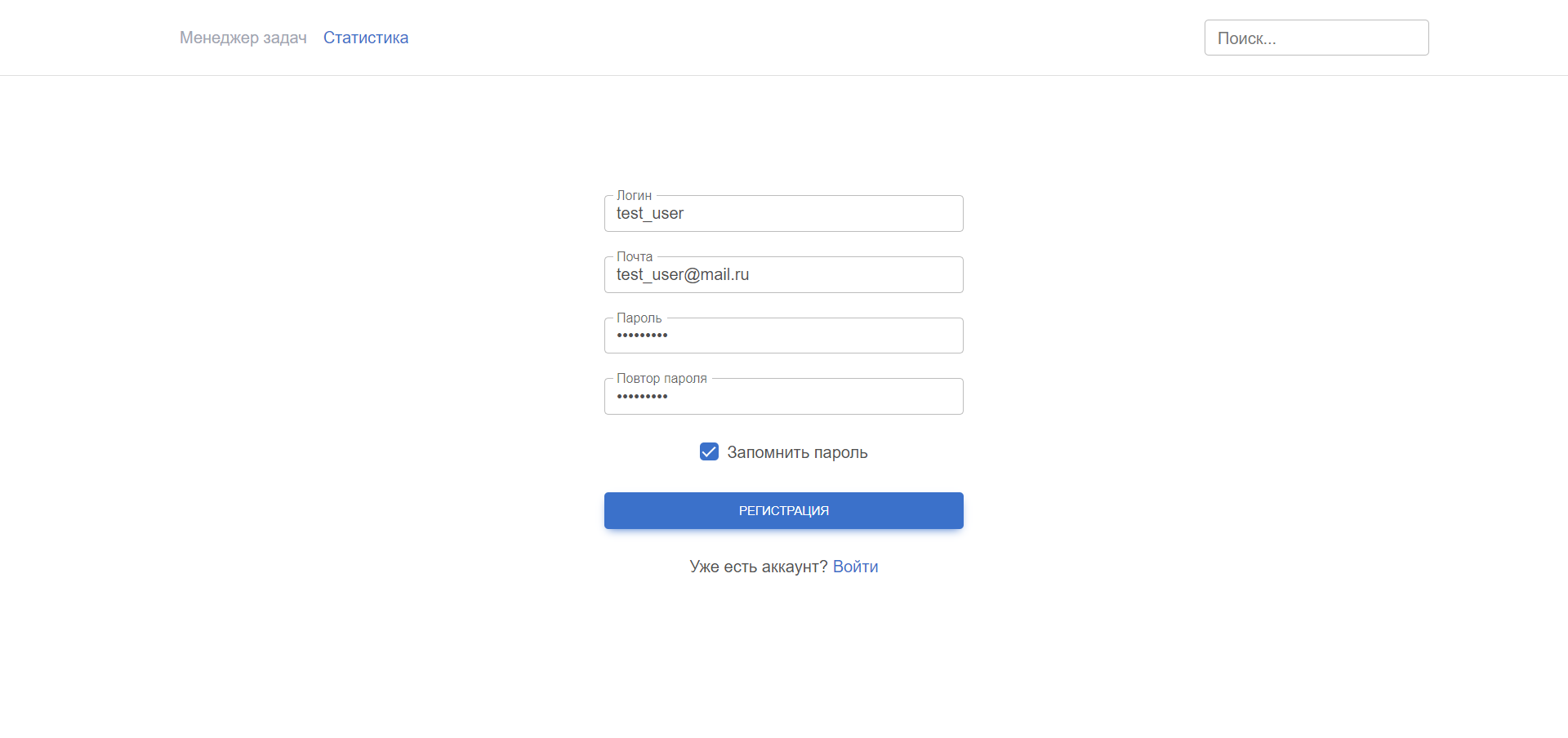


Рисунок 7. Страница регистрации.

На рисунке 8 – страница личного профиля. После регистрации пользователь попадает сюда. Здесь он может просмотреть свои личные данные, а также загрузить свое фото профиля.

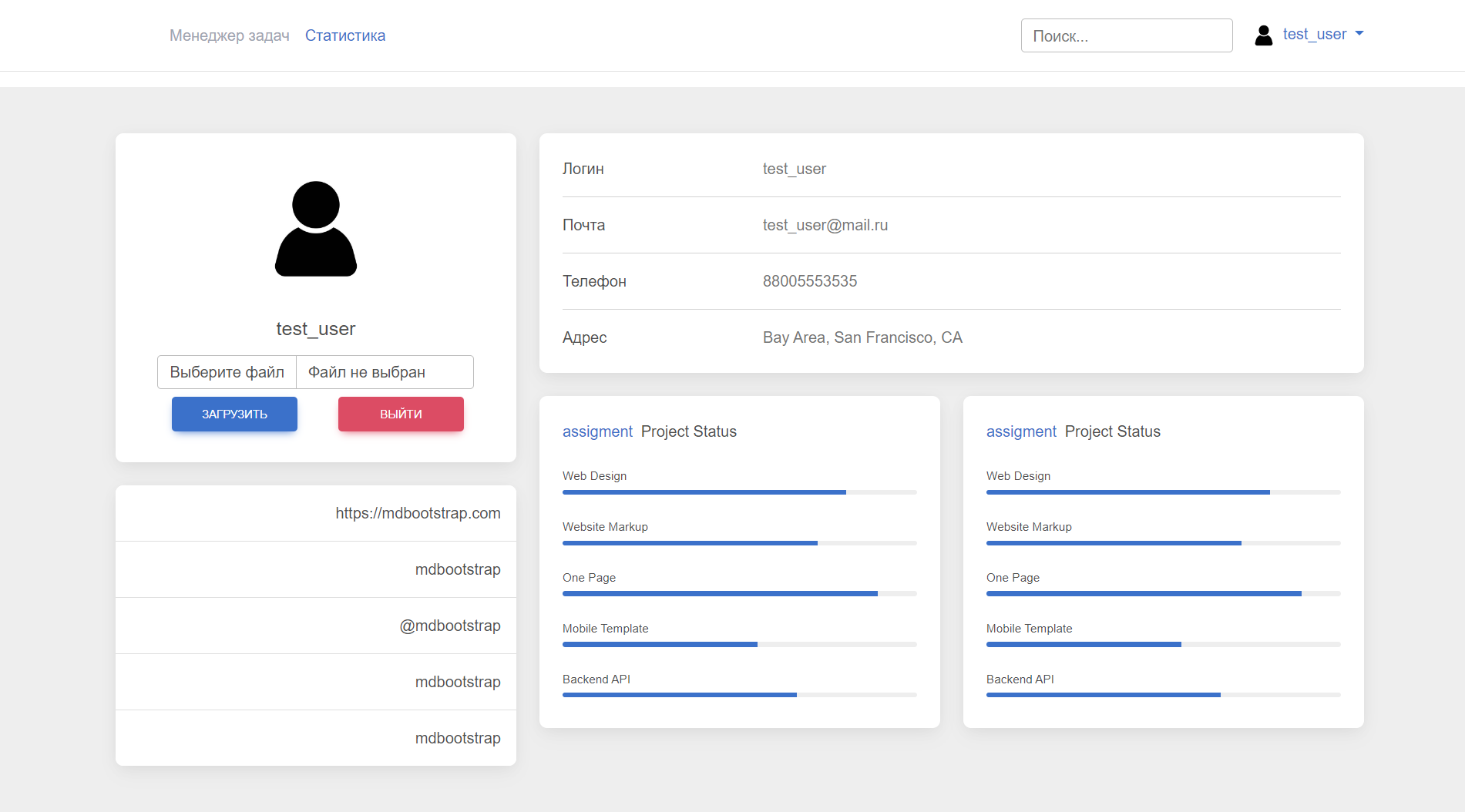


Рисунок 8. Личный профиль

Нажав на логин в верхнем правом углу, откроется выпадающий список и пользователь может перейти на страницу настроек, где можно изменить личные данные и настроить оповещения в Telegram.

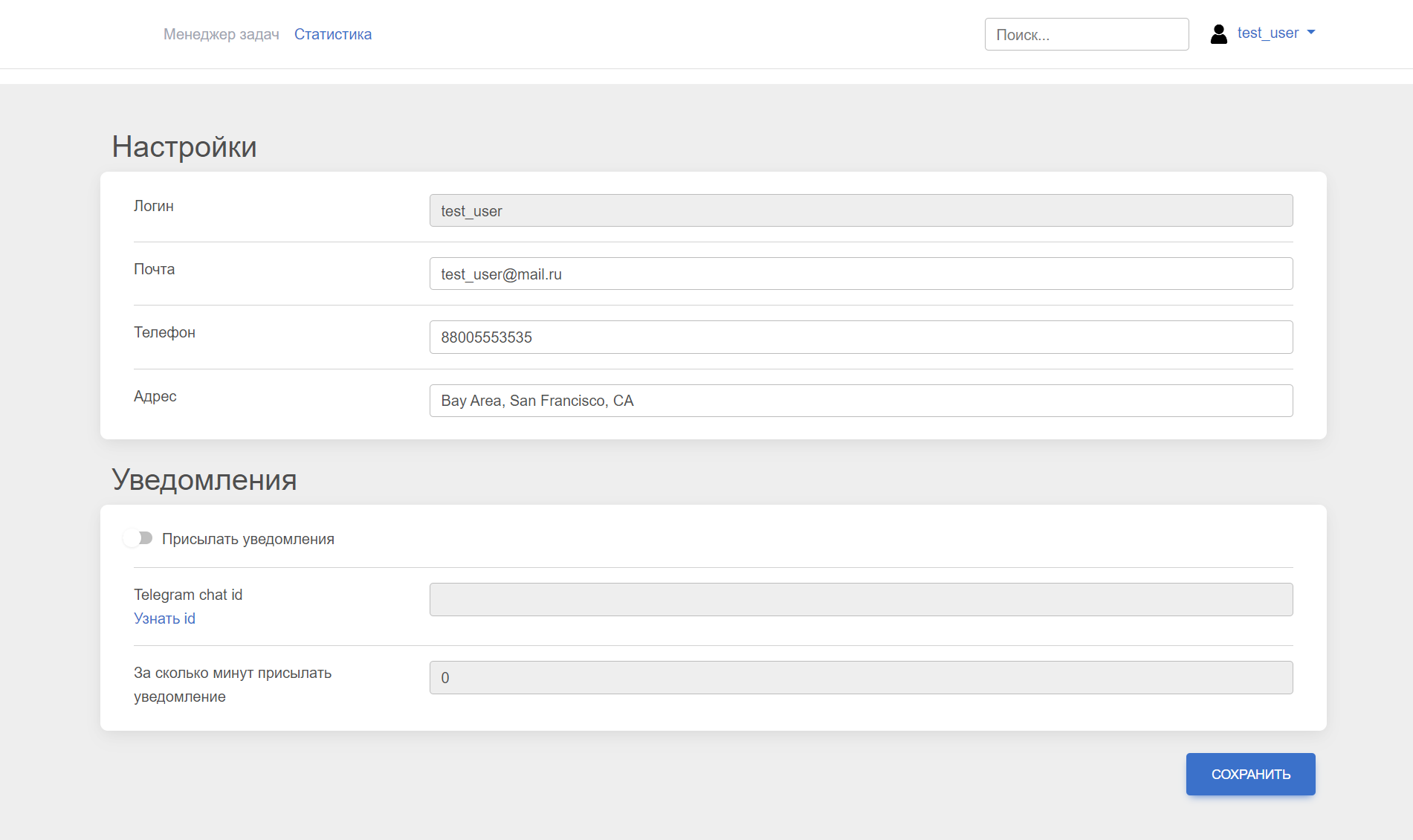


Рисунок 9. Страница настроек.

На главную страницу можно попасть нажав в верхнем левом углу «Менеджер задач». Здесь можно добавлять, редактировать, удалять задачи и категории задач.

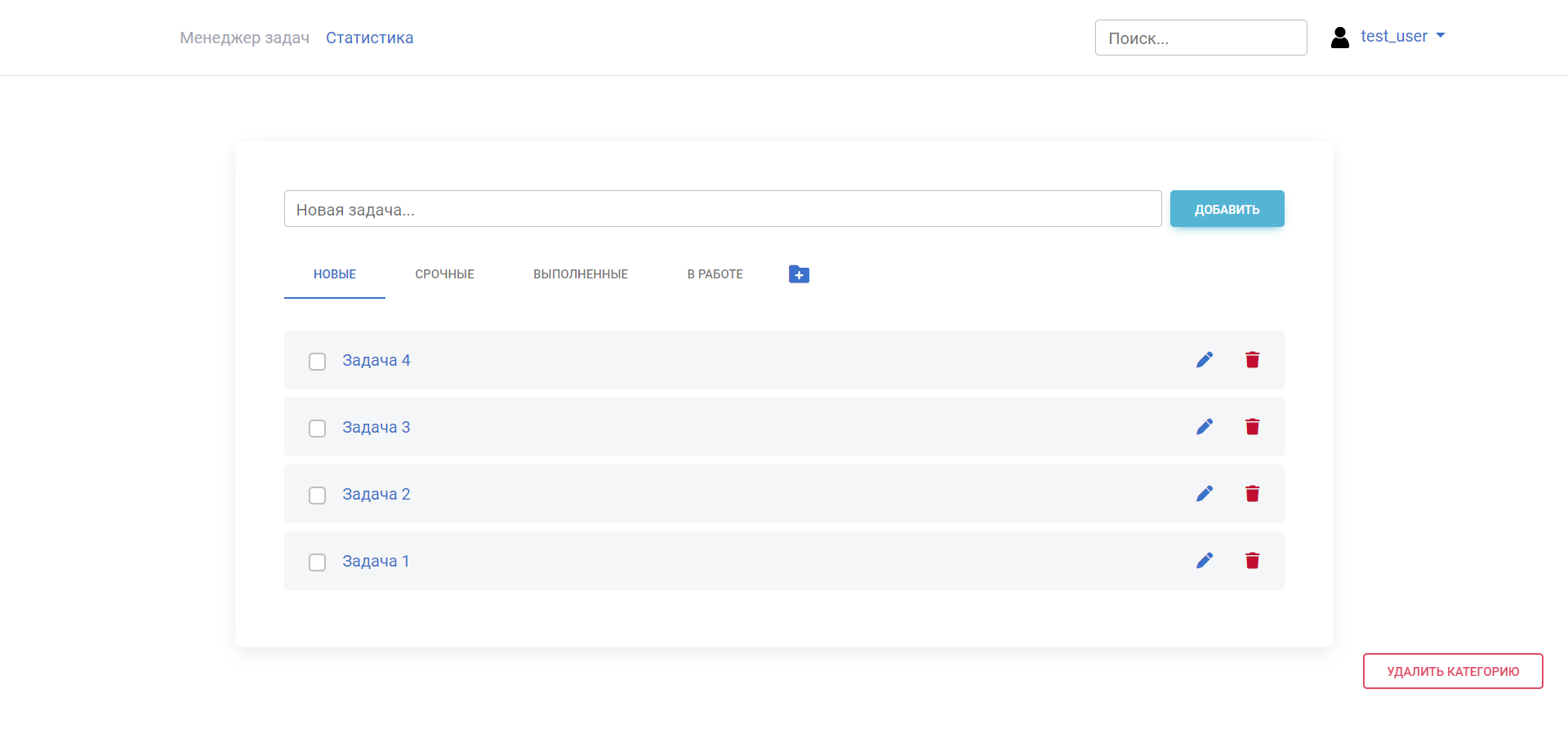


Рисунок 10. Главная страница

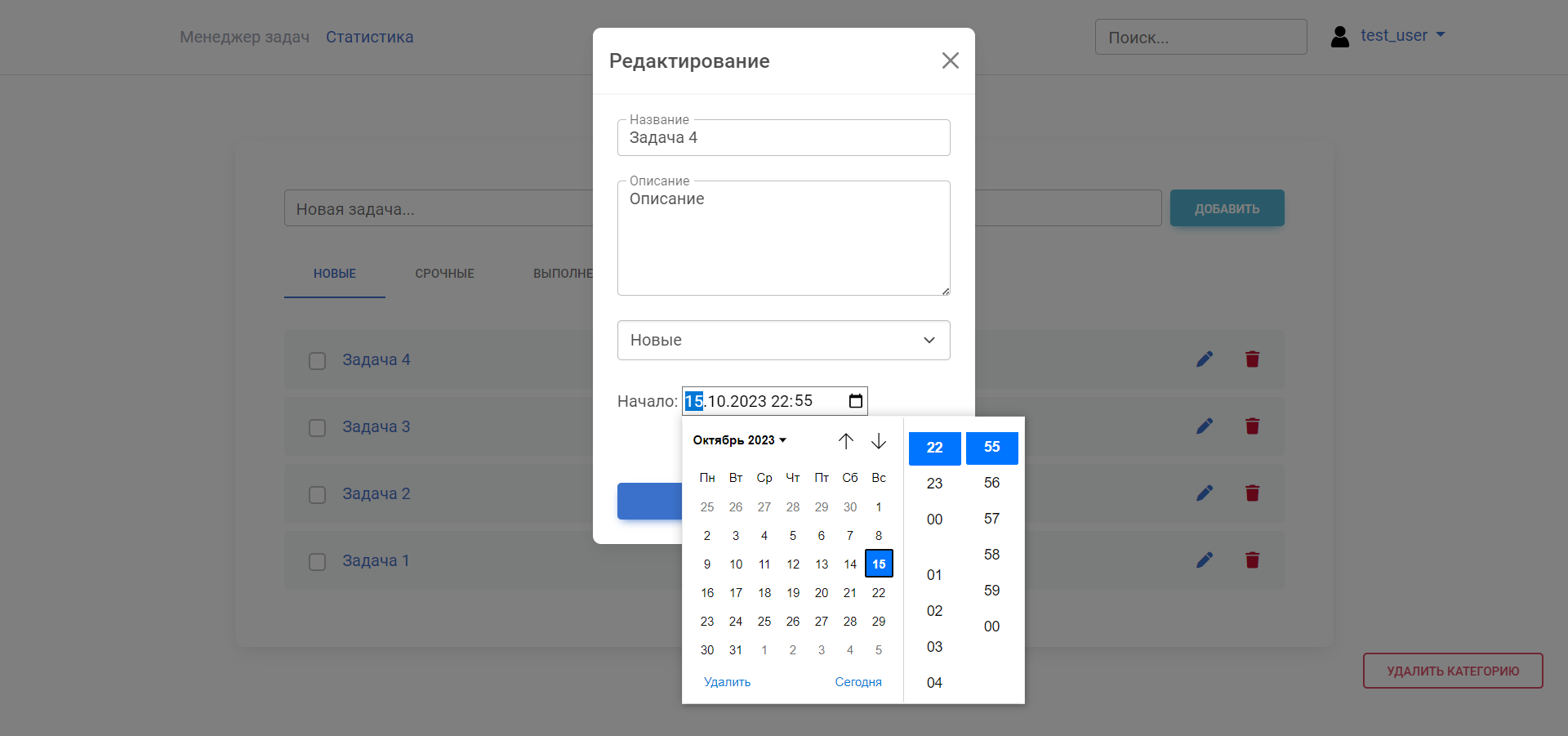
При нажатии на задачу открывается меню ее редактирования (рис 11).

Рисунок 11. Редактирование задачи

При нажатии на иконку папки с плюсом рядом с категориями открывается меню создания новой категории (рис 12).

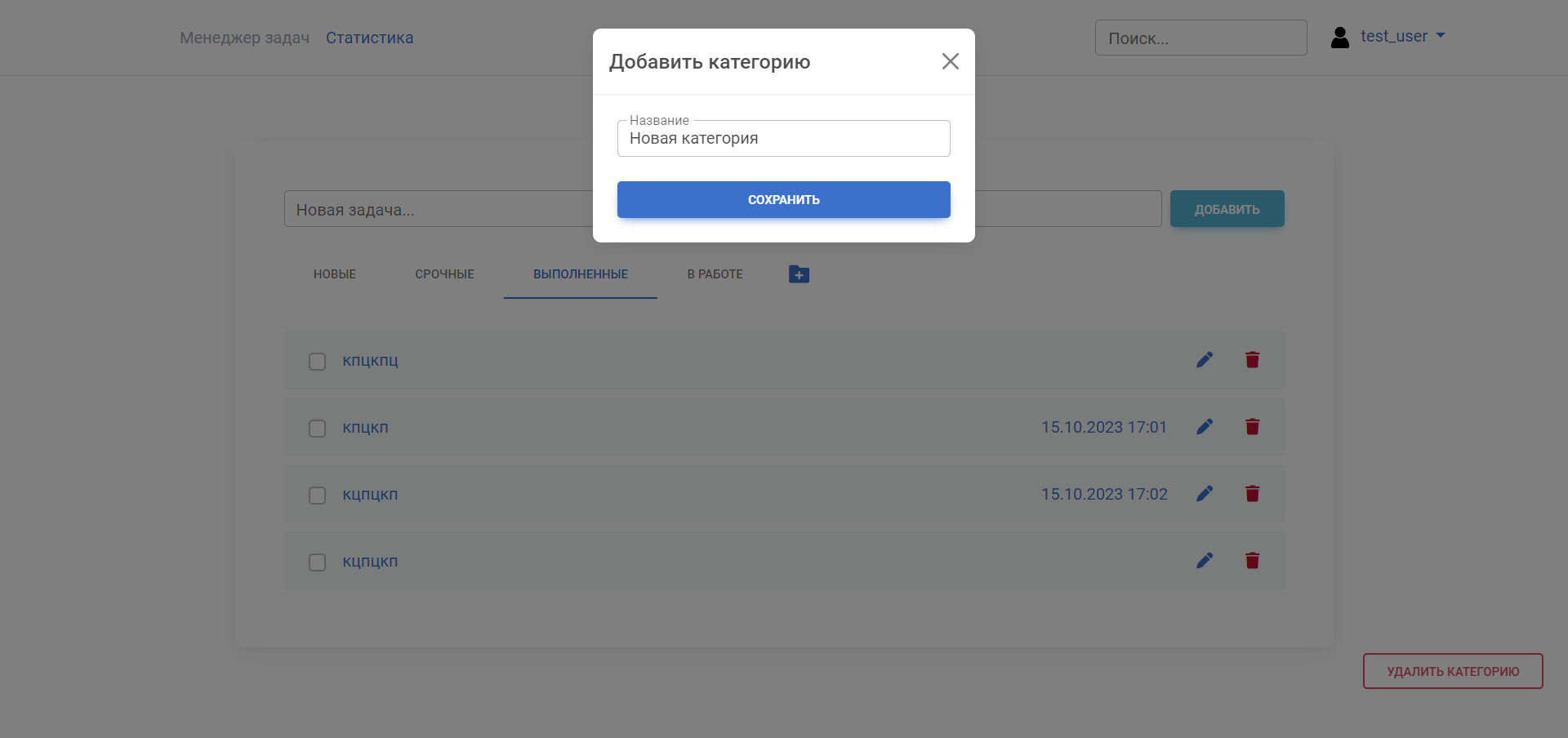


Рисунок 12. Добавление новой категории

Чтобы просмотреть статистику задач нужно нажать по ссылке «Статистика» в левом верхнем углу (рис 13).

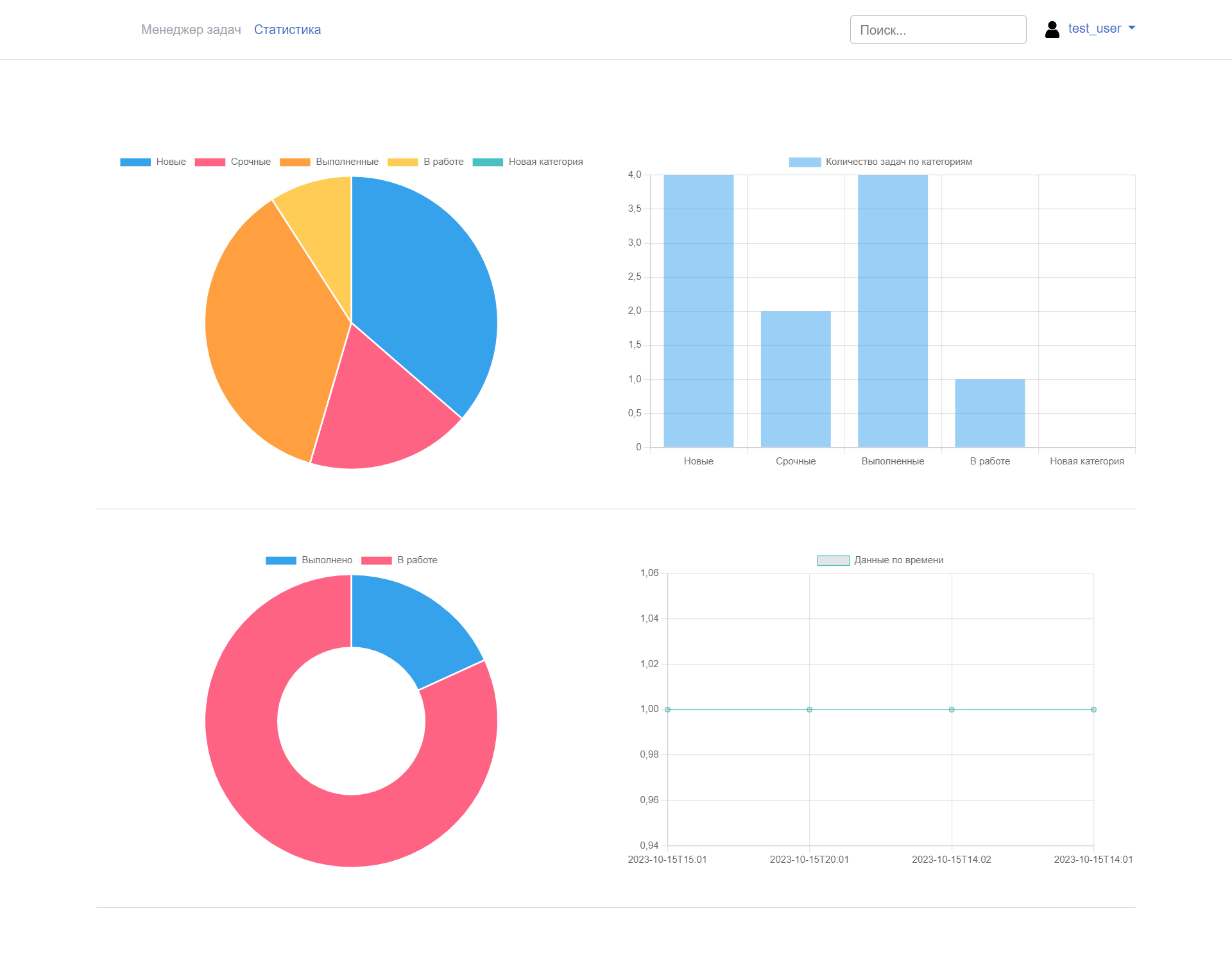


Рисунок 13. Страница статистики

# 3.3. Реализация функциональной части приложения

Для разработки функциональной части приложения были использован язык программирования Python, фреймворк для создания веб-приложений Django, Chart.js для создания инфографики, Aiogram для отправки уведомлений в Telegram, Celery для создания фоновых задач. Листинг кода создания таблиц можно посмотреть в Приложении 2.

4. Тестирование приложения

# 4.1. Тестирование пользовательского интерфейса

Тестирование пользовательского интерфейса было проведено вручную, были проведены тесты в различных браузерах с разными размерами экрана, которые показали что веб-сервис достаточно адаптивен и хорошо выглядит на любых устройствах и платформах.

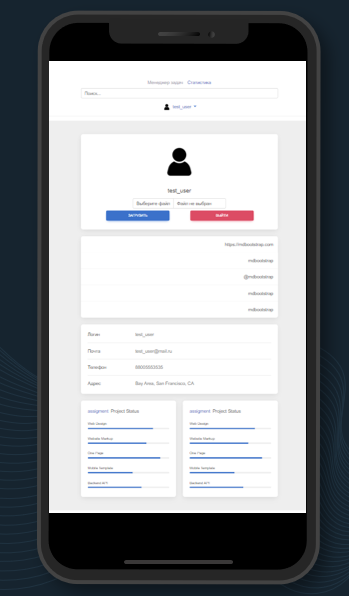
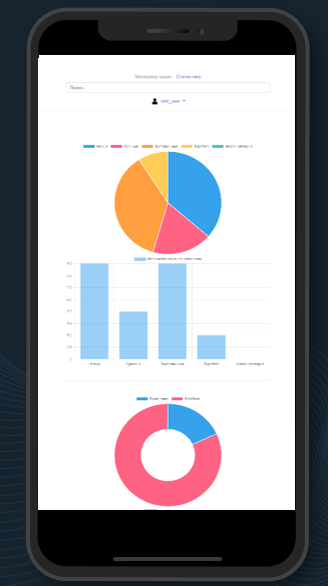
 

Рисунок 14. Веб-приложение в мобильной версии.

# 4.2. Тестирование функциональной части приложения

Тестирование функциональной части приложения было выполнено в автоматизированном режиме с помощью Selenium.

Тесты:

1. Регистрация
2. Вход
3. Изменение личных данных
4. Включение уведомлений
5. Создание задач
6. Редактирование задач
7. Удаление задач
8. Создание категории
9. Удаление категории
10. Просмотр статистики

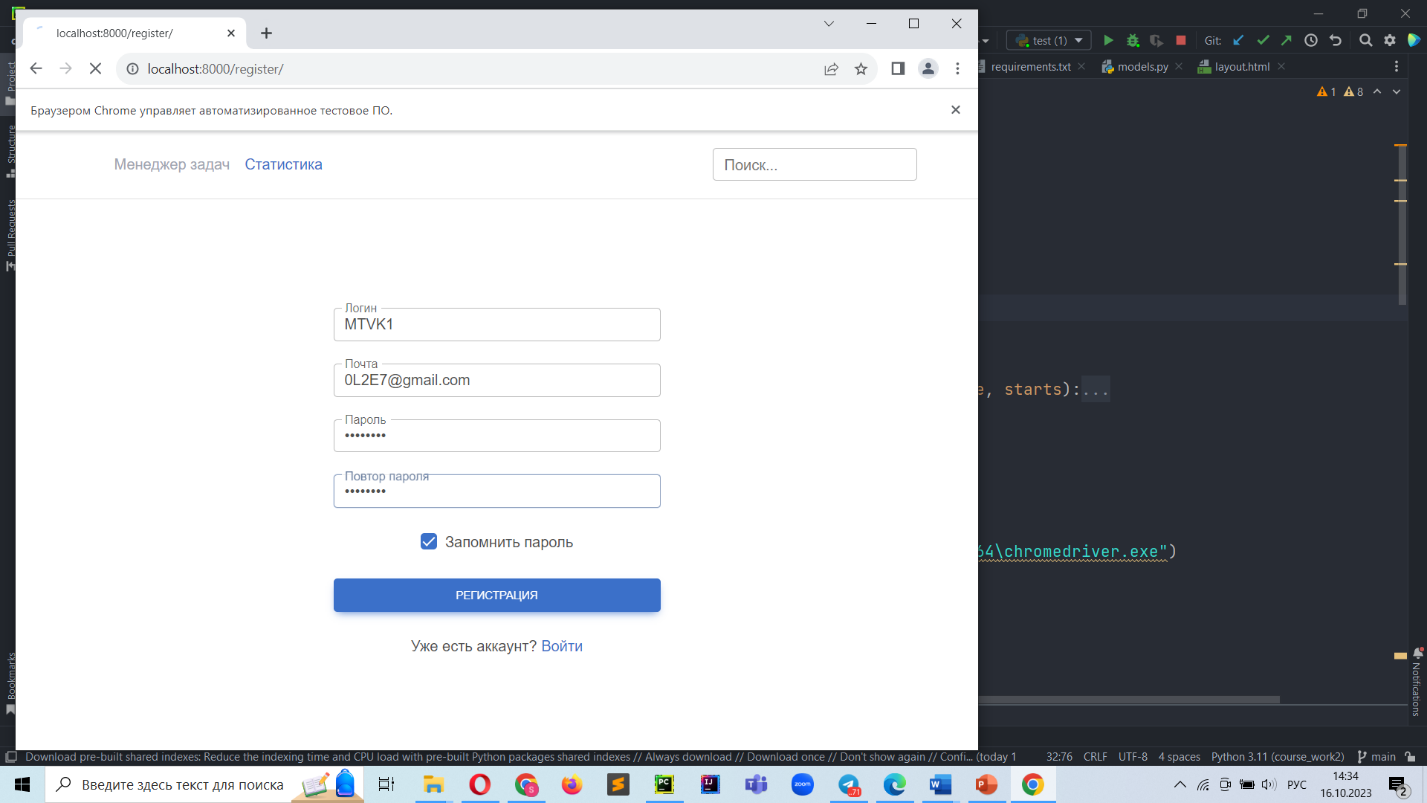


Рисунок 15. Тест регистрации.

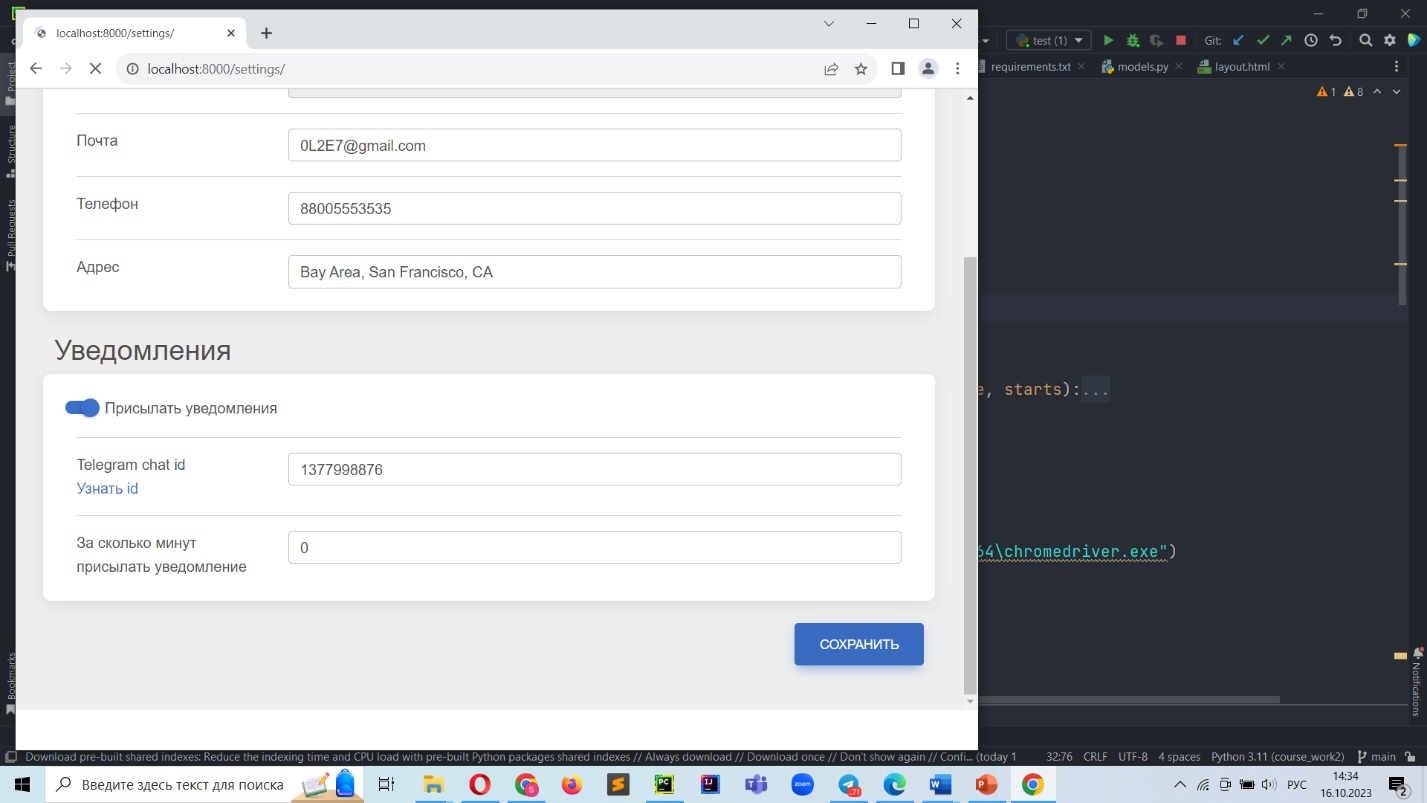


Рисунок 16. Тест включения уведомлений.

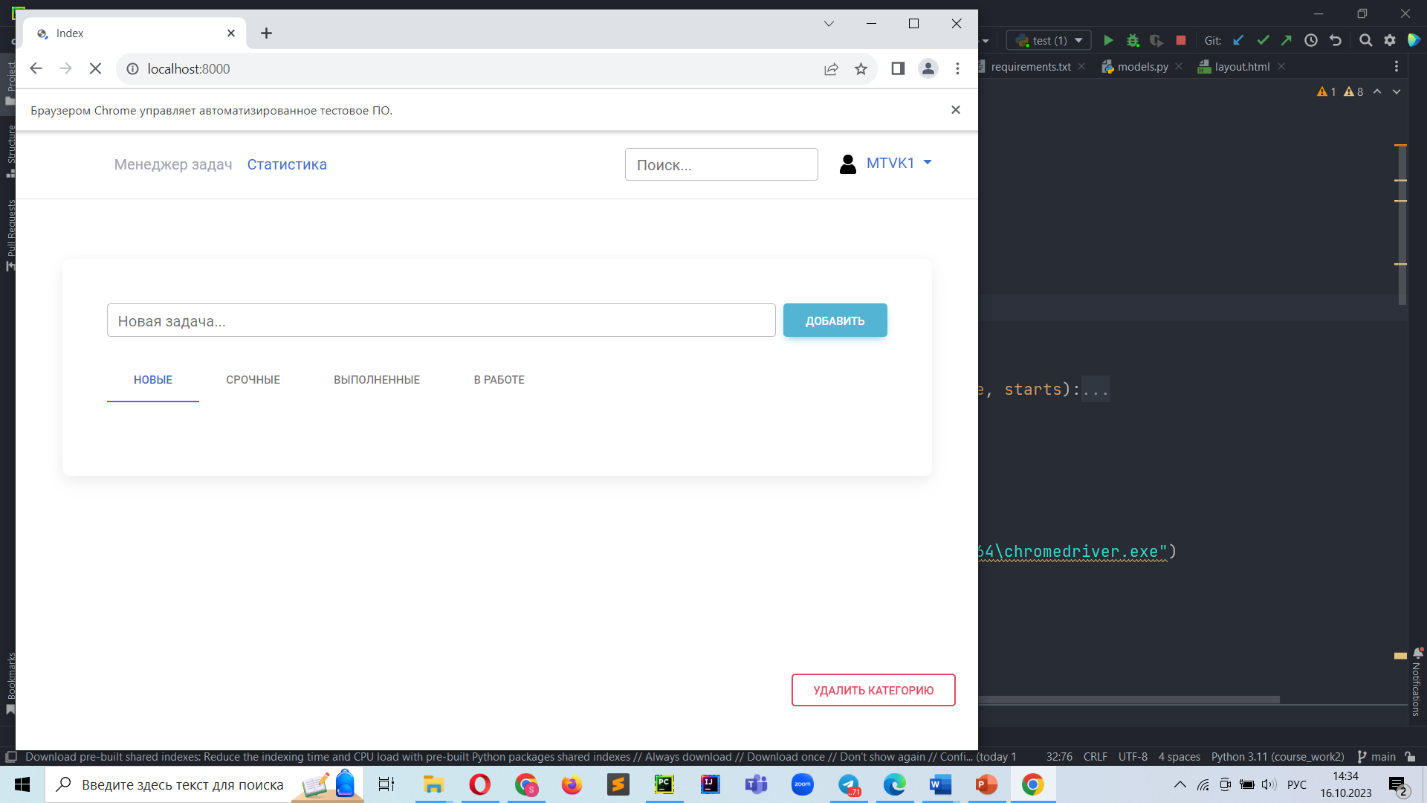
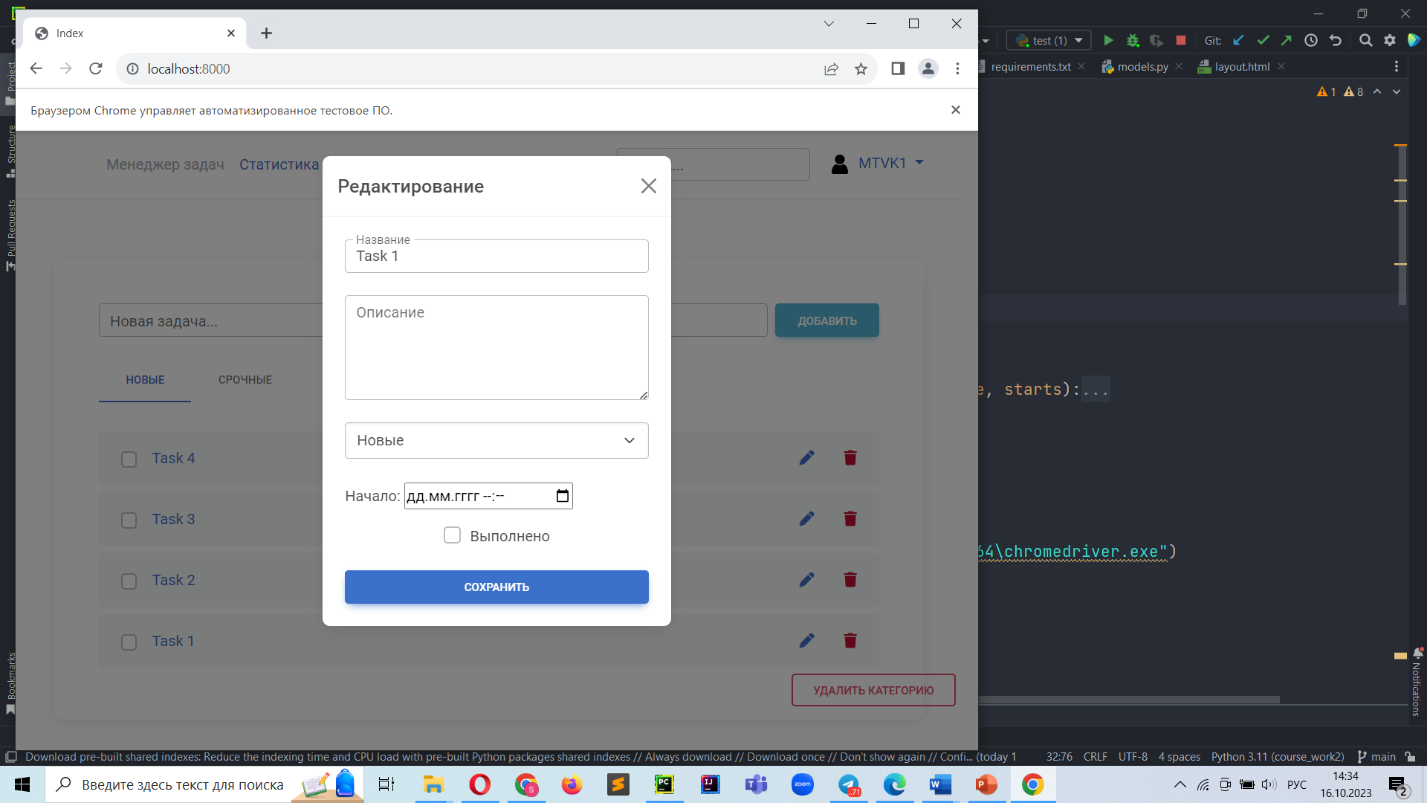
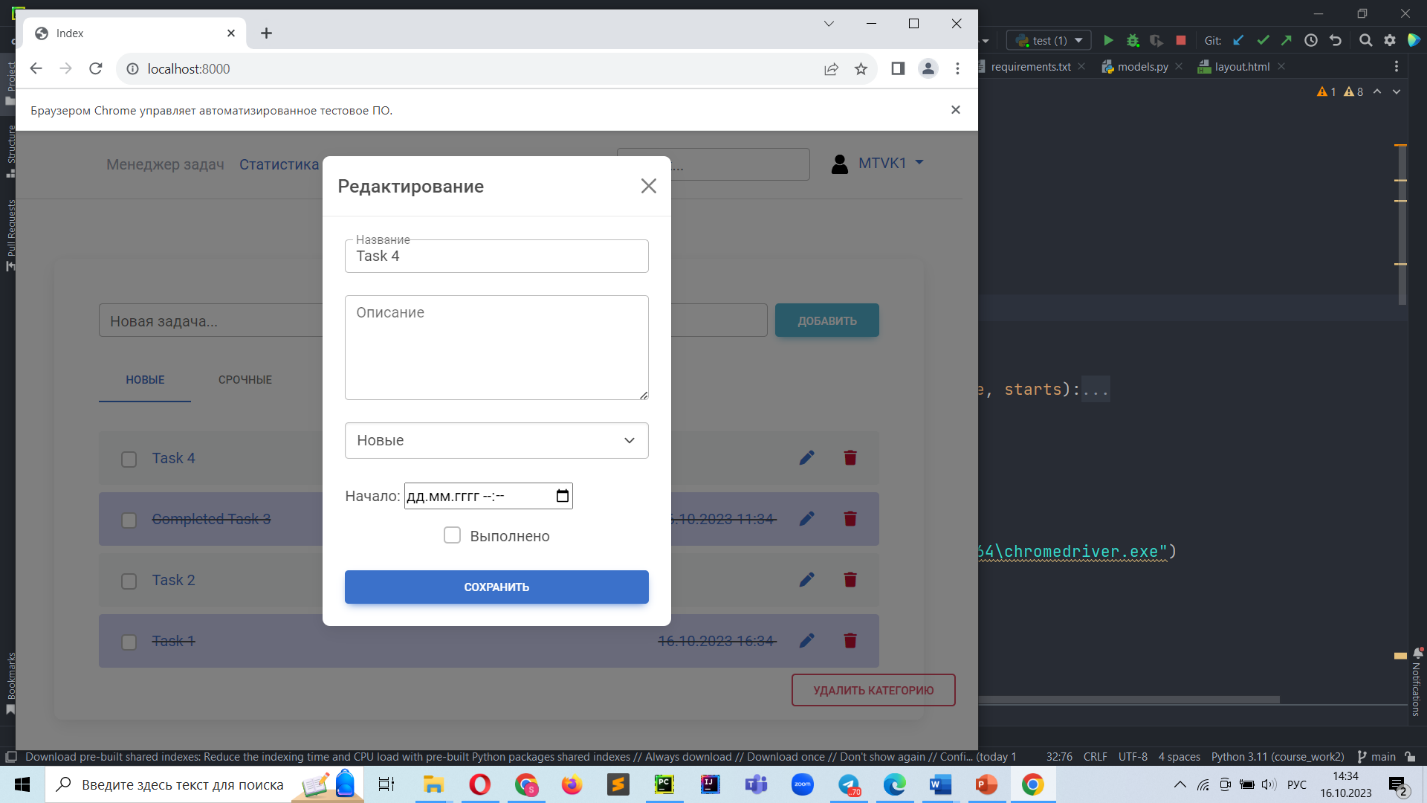


Рисунок 17. Тест главной страницы.

Рисунок 18. Тест добавления задач.

Рисунок 19. Тест редактирования задач.

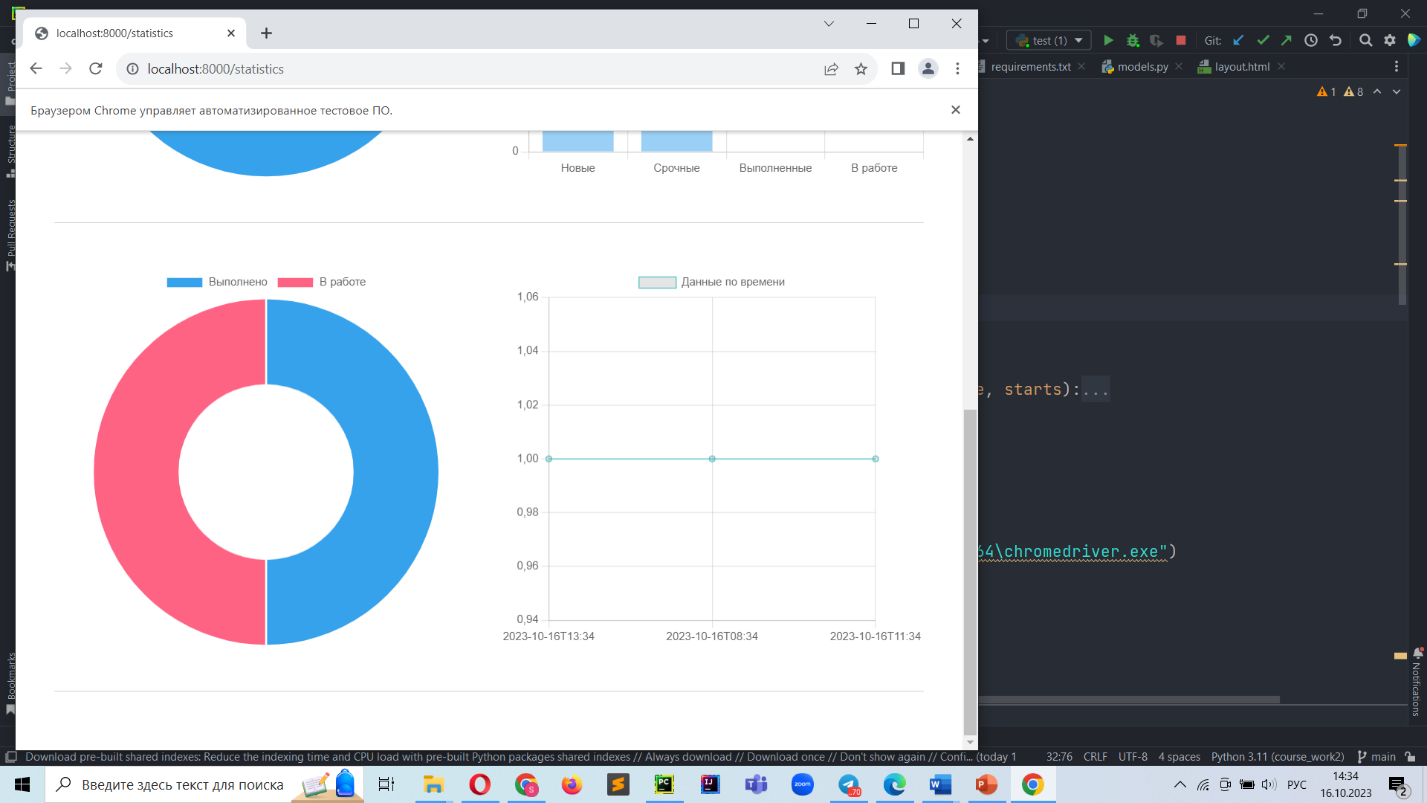


Рисунок 20. Тест просмотра статистики.

Также часть тестов произведена в ручном режиме, чтобы полностью охватить те кейсы, которые не были проверены при автоматизированном тестировании. Такой метод тестирования позволяет проверить функциональность и удобство использования программного продукта с точки зрения конечного пользователя.

# Заключение

В мире, насыщенном информацией и быстрым темпом жизни, эффективное управление временем и задачами становится неотъемлемой частью нашего успеха и благополучия. В ходе исследования и разработки данной курсовой работы, мы глубоко погрузились в мир современных инструментов планирования задач, а также рассмотрели их плюсы и минусы.

В ходе курсовой работы было разработано веб-приложение для управления списком дел и задач, которое позволяет пользователям экономить и организовывать свое время.

Приложение имеет понятный и удобный и функциональный интерфейс, совместимо с разными версиями браузеров, обеспечивает безопасность и конфиденциальность данных пользователей.

Для разработки приложения были использованы следующие технологии: язык программирования Python, фреймворк Django для создания веб-приложений, Sqlite для хранения данных, MDBootstrap для стильного дизайна, Chart.js для создания красочных графиков и диаграмм, Aiogram для уведомлений в Telegram, Celery для управления фоновыми задачами (отправление уведомлений), Selenium для автоматизированного тестирования.

Приложение было протестировано на разных устройствах с разными размерами экранов и получило положительные отзывы от потенциальных пользователей. Разработанное приложение может быть использовано людьми для личного использования с целью организации времени и задач, оно может также быть использовано в рабочем пространстве и для управления проектами. Многие офисные сотрудники и фрилансеры могут воспользоваться таким приложением для повышения эффективности работы. Планировщик задач может быть использован для управления проектами в различных областях, включая информационные технологии, строительство, маркетинг, образование и другие. Он поможет распределить задачи, следить за сроками и оценивать прогресс.

# Список использованных источников

1. Официальная документация Python. [Электронный ресурс] /docs.python.org - Python - Режим доступа: https://docs.python.org/ - Дата обращения: 05.02.2023.
2. Официальная документация Django. [Электронный ресурс] /docs.djangoproject.com - Django - Режим доступа: https://docs.djangoproject.com/ - Дата обращения: 10.01.2023.
3. Руководство по использованию SQLite в Django. [Электронный ресурс] /docs.djangoproject.com - Django - Режим доступа: https://docs.djangoproject.com/en/4.0/topics/db/sqlite3/ - Дата обращения: 15.02.2023.
4. "Создание графического интерфейса с использованием HTML, CSS и JavaScript". [Электронный ресурс] /w3schools.com - W3Schools - Режим доступа: https://www.w3schools.com/ - Дата обращения: 20.03.2023.
5. Интеграция уведомлений в веб-приложение с использованием aiogram. [Электронный ресурс] /docs.aiogram.dev - Aiogram - Режим доступа: https://docs.aiogram.dev/ - Дата обращения: 25.04.2023.
6. Создание графиков и диаграмм с использованием Chart.js. [Электронный ресурс] /chartjs.org - Chart.js - Режим доступа: https://www.chartjs.org/ - Дата обращения: 30.04.2023.
7. Документация Celery. [Электронный ресурс] /docs.celeryproject.org - Celery - Режим доступа: https://docs.celeryproject.org/ - Дата обращения: 15.04.2023.
8. Документация Selenium. [Электронный ресурс] /selenium.dev - Selenium - Режим доступа: https://www.selenium.dev/ - Дата обращения: 10.04.2023.

# Приложение 1

Листинг кода для создания таблиц в БД

**from** django **import** forms  
**from** django.contrib.auth.forms **import** UserCreationForm  
**from** django.contrib.auth.models **import** User  
**from** django.db **import** models  
  
  
**class** Category(models.Model):  
 user = models.ForeignKey(User, on\_delete=models.CASCADE, null=**False**)  
 title = models.CharField(**'Название'**, max\_length=100)  
  
 **class** Meta:  
 verbose\_name = **'Категория задания'** verbose\_name\_plural = **'Категории заданий'  
  
 def** \_\_str\_\_(self):  
 **return** self.title  
  
  
**class** ToDo(models.Model):  
 title = models.CharField(**'Название'**, max\_length=500, null=**False**)  
 category = models.ForeignKey(Category, on\_delete=models.CASCADE)  
 is\_complete = models.BooleanField(**'Завершено'**, default=**False**)  
 description = models.TextField(default=**''**)  
 starts = models.DateTimeField(null=**True**)  
 num\_notifications = models.IntegerField(default=0)  
  
 **class** Meta:  
 verbose\_name = **'Задание'** verbose\_name\_plural = **'Задания'  
  
 def** \_\_str\_\_(self):  
 **return** self.title  
  
  
**class** Profile(models.Model):  
 user = models.OneToOneField(User, null=**True**, on\_delete=models.CASCADE)  
 is\_notifications = models.BooleanField(**'Присылать уведомления'**, default=**False**)  
 tg\_chat\_id = models.IntegerField(null=**True**)  
 time\_before\_notification = models.IntegerField(default=0)  
 phone = models.IntegerField(default=88005553535)  
 address = models.CharField(max\_length=200, default=**"Bay Area, San Francisco, CA"**)  
 image = models.ImageField(upload\_to=**'images'**, default=**'images/default.jpg'**)  
  
 **def** \_\_str\_\_(self):  
 **return** str(self.user)  
  
  
**class** RegisterForm(UserCreationForm):**class** Meta(UserCreationForm.Meta):  
 model = User  
 fields = (**"username"**, **"email"**)  
  
  
**class** ProfileForm(forms.ModelForm):  
 **class** Meta:  
 model = Profile  
 fields = [**'image'**]

# Приложение 2

Листинг кода функций отображения страниц сайта

**from** django.contrib.auth **import** authenticate, login  
**from** django.contrib.auth.decorators **import** login\_required  
**from** django.shortcuts **import** render, redirect  
**from** django.urls **import** reverse\_lazy  
**from** django.utils.datastructures **import** MultiValueDictKeyError  
**from** django.views.decorators.http **import** require\_http\_methods  
**from** django.views.generic **import** FormView  
  
**from** .models **import** ToDo, Category, RegisterForm, Profile, ProfileForm  
  
  
@login\_required  
**def** index(request):  
 categories\_list = Category.objects.filter(user\_id=request.user.id)  
 todo\_list = ToDo.objects.all()[::-1]  
 **return** render(request, **'todoapp/index.html'**, {**'title'**: **'Index'**, **'categories\_list'**: categories\_list, **'todo\_list'**: todo\_list})  
  
  
@require\_http\_methods([**'POST'**])  
**def** add(request):  
 title = request.POST[**'title'**]  
 category\_id = request.POST[**'category'**]  
 todo = ToDo(title=title, category=Category.objects.get(id=category\_id))  
 todo.save()  
 **return** redirect(**'index'**)  
  
  
@require\_http\_methods([**'POST'**])  
**def** add\_category(request):  
 title = request.POST[**'name'**]  
 category = Category(user=request.user, title=title)  
 category.save()  
 **return** redirect(**'index'**)  
  
  
@require\_http\_methods([**'POST'**])  
**def** delete\_category(request):  
 category\_id = request.POST[**'category'**]  
 category = Category.objects.get(id=category\_id)  
 category.delete()  
 **return** redirect(**'index'**)  
  
  
@require\_http\_methods([**'POST'**])  
**def** update(request, todo\_id):  
 todo = ToDo.objects.get(id=todo\_id)  
 todo.title = request.POST[**'title'**]  
 todo.description = request.POST[**'description'**]  
 todo.category\_id = request.POST[**'category'**]  
 **if** request.POST[**'starts'**] == **''**:  
 todo.starts = **None  
 else**:  
 todo.starts = request.POST[**'starts'**] + **'+03:00'** todo.is\_complete = **True if** request.POST.get(**'is\_complete'**, **False**) **else False** todo.save()  
 **return** redirect(**'index'**)  
  
  
**def** delete(request, todo\_id):  
 todo = ToDo.objects.get(id=todo\_id)  
 todo.delete()  
 **return** redirect(**'index'**)  
  
  
@login\_required  
**def** profile(request):  
 **if** request.method == **'POST'**:  
 form = ProfileForm(request.POST, request.FILES)  
 **if** form.is\_valid():  
 profile = Profile.objects.get(user=request.user)  
 profile.image = form.cleaned\_data[**'image'**]  
 profile.save()  
 **return** redirect(**'profile'**)  
 **else**:  
 form = ProfileForm()  
 **return** render(request, **'todoapp/profile.html'**, {**'form'**: form})  
  
  
**class** RegisterView(FormView):  
 form\_class = RegisterForm  
 template\_name = **'registration/register.html'** *# success\_url = reverse\_lazy('todolist:profile')* success\_url = reverse\_lazy(**'profile'**)  
  
 **def** form\_valid(self, form):  
 form.save()  
 form\_valid = super().form\_valid(form)  
 **if** form\_valid:  
 *#get the username and password* username = self.request.POST[**'username'**]  
 password = self.request.POST[**'password1'**]  
 *#authenticate user then login* user = authenticate(username=username, password=password)  
 login(self.request, user)  
 *# after user creating, running signals.py, where creates profile for user* **return** redirect(**'profile'**)  
 **return** form\_valid  
  
  
@login\_required  
**def** settings(request):  
 **return** render(request, **'todoapp/settings.html'**)  
  
  
@require\_http\_methods([**'POST'**])  
**def** update\_settings(request):  
 user = request.user  
 user.email = request.POST[**'email'**]  
 user.save()  
  
 profile = Profile.objects.get(user=request.user)  
 profile.address = request.POST[**'address'**]  
 profile.phone = request.POST[**'phone'**]  
 profile.is\_notifications = **True if** request.POST.get(**'is\_notifications'**, **False**) **else False  
 try**:  
 profile.tg\_chat\_id = int(request.POST[**'tg\_chat\_id'**])  
 profile.time\_before\_notification = int(request.POST[**'time\_before\_notification'**])  
 **except** ValueError:  
 **return** redirect(**'settings'**)  
 **except** MultiValueDictKeyError:  
 **pass** profile.save()  
 **return** redirect(**'settings'**)  
  
  
**def** statistics(request):  
 categories = Category.objects.filter(user\_id=request.user.id)  
 labels = []  
 values = []  
 **for** c **in** categories:  
 todos = ToDo.objects.filter(category=c)  
 labels.append(c.title)  
 values.append(len(todos))  
 categories\_data = {  
 **"labels"**: labels,  
 **"values"**: values,  
 }  
 todos = []  
 **for** c **in** categories:  
 todos.extend(ToDo.objects.filter(category=c))  
  
 completed\_tasks = len(list(filter(**lambda** todo: todo.is\_complete, todos)))  
 completed = {  
 **"labels"**: [**'Выполнено'**, **'В работе'**],  
 **"values"**: [completed\_tasks, len(todos) - completed\_tasks]  
 }  
  
 todo\_starts = [todo.starts **for** todo **in** todos **if** todo.starts **is not None**]  
 tasks\_by\_hour = {}  
 *# Перебираем время выполнения задач и разбиваем их по часам* **for** todo **in** todo\_starts:  
 *# Получаем час из времени выполнения* hour = todo.strftime(**"%Y-%m-%dT%H"**)  
 tasks\_by\_hour[hour] = tasks\_by\_hour.get(hour, 0) + 1  
  
 tasks\_by\_minute = {}  
 **for** todo **in** todo\_starts:  
 minute = todo.strftime(**"%Y-%m-%dT%H:%M"**)  
 tasks\_by\_minute[minute] = tasks\_by\_minute.get(minute, 0) + 1  
  
 tasks\_by\_day = {}  
 **for** todo **in** todo\_starts:  
 day = todo.strftime(**"%Y-%m-%d"**)  
 tasks\_by\_day[day] = tasks\_by\_day.get(day, 0) + 1  
  
 **if** len(tasks\_by\_day) > 3:  
 todo\_times = tasks\_by\_day  
 **elif** len(tasks\_by\_hour) > 3:  
 todo\_times = tasks\_by\_hour  
 **else**:  
 todo\_times = tasks\_by\_minute  
  
 todo\_times = {  
 **"labels"**: list(todo\_times.keys()),  
 **"values"**: list(todo\_times.values())  
 }  
  
 **return** render(request, **'todoapp/statistics.html'**,  
 {**"categories"**: categories\_data,  
 **"completed"**: completed,  
 **"todo\_times"**: todo\_times  
 })

# Приложение 3

Листинг кода для автоматизированного тестирования

**import** datetime  
**import** os  
**import** random  
**import** string  
**import** time  
  
**from** selenium **import** webdriver  
**from** selenium.webdriver **import** Keys  
**from** selenium.webdriver.chrome.service **import** Service  
**from** selenium.webdriver.common.by **import** By  
**from** selenium.webdriver.support **import** expected\_conditions **as** EC  
**from** selenium.webdriver.support.select **import** Select  
**from** selenium.webdriver.support.wait **import** WebDriverWait  
**from** dotenv **import** load\_dotenv  
  
load\_dotenv()  
  
CHROME\_DRIVER = os.environ.get(**'CHROME\_DRIVER'**)  
  
  
**def** register(login, password, email):  
 login\_input = driver.find\_element(By.ID, **'form2Example1'**)  
 login\_input.clear()  
 login\_input.send\_keys(login)  
  
 email\_input = driver.find\_element(By.ID, **'form2Example2'**)  
 email\_input.clear()  
 email\_input.send\_keys(email)  
  
 password\_input = driver.find\_element(By.ID, **'form2Example3'**)  
 password\_input.clear()  
 password\_input.send\_keys(password)  
  
 repeat\_password\_input = driver.find\_element(By.ID, **'form2Example4'**)  
 repeat\_password\_input.clear()  
 repeat\_password\_input.send\_keys(password)  
  
 driver.find\_element(By.XPATH, **'//button[text()="Регистрация"]'**).click()  
  
  
**def** change\_task(old\_title, title, description, category, is\_complete, starts):  
 task\_link = driver.find\_element(By.LINK\_TEXT, old\_title)  
 task\_link.click()  
 task\_id = task\_link.get\_attribute(**'data-mdb-target'**)[1:]  
  
 time.sleep(2)  
  
 modal = driver.find\_element(By.ID, task\_id)  
 title\_input = modal.find\_element(By.NAME, **'title'**)  
 title\_input.clear()  
 title\_input.send\_keys(title)  
  
 description\_input = modal.find\_element(By.NAME, **'description'**)  
 description\_input.clear()  
 description\_input.send\_keys(description)  
  
 category\_input = Select(modal.find\_element(By.CLASS\_NAME, **'form-select'**))  
 category\_input.select\_by\_visible\_text(category)  
  
 **if** is\_complete:  
 modal.find\_element(By.NAME, **'is\_complete'**).click()  
  
 starts\_input = modal.find\_element(By.NAME, **'starts'**)  
 driver.execute\_script(**f"arguments[0].value = '{**starts**}';"**, starts\_input)  
  
 time.sleep(1)  
 save\_button = WebDriverWait(modal, 1).until(EC.element\_to\_be\_clickable((By.NAME, **'save-btn'**)))  
 save\_button.click()  
  
  
**def** settings(tg\_chat\_id):  
 dropdown = WebDriverWait(driver, 3).until(EC.presence\_of\_element\_located((By.CLASS\_NAME, **'dropdown'**)))  
 dropdown.click()  
 driver.find\_element(By.LINK\_TEXT, **"Настройки"**).click()  
  
 is\_notifications = WebDriverWait(driver, 3).until(EC.presence\_of\_element\_located((By.NAME, **'is\_notifications'**)))  
 driver.execute\_script(**"arguments[0].scrollIntoView();"**, is\_notifications)  
 time.sleep(1)  
 is\_notifications.click()  
 time.sleep(1)  
 tg\_chat\_id\_input = driver.find\_element(By.NAME, **'tg\_chat\_id'**)  
 tg\_chat\_id\_input.clear()  
 tg\_chat\_id\_input.send\_keys(tg\_chat\_id)  
  
 driver.find\_element(By.XPATH, **'//button[text()="Сохранить"]'**).click()  
  
  
s = Service(executable\_path=CHROME\_DRIVER)  
driver = webdriver.Chrome(service=s)  
  
  
**try**:  
 *# driver.maximize\_window()* driver.get(**'http://localhost:8000'**)  
  
 driver.find\_element(By.LINK\_TEXT, **"Зарегистрироваться"**).click()  
 login = **''**.join(random.choice(string.ascii\_uppercase + string.digits) **for** \_ **in** range(5))  
 password = **''**.join(random.choice(string.ascii\_uppercase + string.digits) **for** \_ **in** range(8))  
 email = **''**.join(random.choice(string.ascii\_uppercase + string.digits) **for** \_ **in** range(5)) + **"@gmail.com"** register(login, password, email)  
 settings(tg\_chat\_id=1377998876)  
  
 task\_manager\_page = WebDriverWait(driver, 1).until(EC.presence\_of\_element\_located((By.LINK\_TEXT, **"Менеджер задач"**)))  
 task\_manager\_page.click()  
  
 **for** i **in** range(1, 5):  
 new\_task\_input = driver.find\_element(By.ID, **'form2'**)  
 new\_task\_input.send\_keys(**f'Task {**i**}'**)  
 new\_task\_input.send\_keys(Keys.ENTER)  
  
 time.sleep(1)  
  
 task\_starts = (datetime.datetime.now() + datetime.timedelta(hours=2)).strftime(**"%Y-%m-%dT%H:%M"**)  
 change\_task(**'Task 1'**, **'Task 1'**, **'Some description'**, **'Новые'**, **True**, task\_starts)  
 time.sleep(1)  
  
 task\_starts = (datetime.datetime.now() - datetime.timedelta(hours=3)).strftime(**"%Y-%m-%dT%H:%M"**)  
 change\_task(**'Task 3'**, **'Completed Task 3'**, **'Some description'**, **'Новые'**, **True**, task\_starts)  
 time.sleep(1)  
  
 task\_starts = datetime.datetime.now().strftime(**"%Y-%m-%dT%H:%M"**)  
 change\_task(**'Task 4'**, **'Updated Task 4'**, **'Some description'**, **'Срочные'**, **False**, task\_starts)  
 time.sleep(1)  
  
 tab = WebDriverWait(driver, 1).until(EC.element\_to\_be\_clickable((By.LINK\_TEXT, **'СРОЧНЫЕ'**)))  
 tab.click()  
 time.sleep(1)  
  
 task\_manager\_page = WebDriverWait(driver, 1).until(EC.presence\_of\_element\_located((By.LINK\_TEXT, **"Статистика"**)))  
 task\_manager\_page.click()  
 time.sleep(1)  
  
 myDoughnutChart = driver.find\_element(By.ID, **"myDoughnutChart"**)  
 driver.execute\_script(**"arguments[0].scrollIntoView();"**, myDoughnutChart)  
  
 time.sleep(5)  
**except** Exception **as** e:  
 print(e)  
**finally**:  
 driver.close()  
 driver.quit()