**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

**по дисциплине «Введение в нереляционные базы данных»**

**Тема: Графовый трекер задач**

| Студент гр. 2382 |  | Муравин Е.Е. |
| --- | --- | --- |
| Студент гр. 2382 |  | Федоров М.В. |
| Студент гр. 2382 |  | Бухарин М.А. |
| Преподаватель |  | Заславский М.М |

Санкт-Петербург

2025

**ЗАДАНИЕ**

| Студент Муравин Е.Е. 2382  Студент Федоров М.В. 2382  Студент Бухарин М.А. 2382 |
| --- |
|  |
| Тема работы:Графовый трекер задач |
| Исходные данные:  Задача - реализовать трекер задач (по аналогии с asana trello jira), который бы учитывал и активно использовал зависимости между задачами, а также нелинейность множества задач (например, задачи на стройке). Хранение, визуализация задач в виде таблицы и графа. Анализ графовых характеристик задач, построение списка приоритетов выполнения набора задач.  Используемая база данных – Neo4j. |
| Содержание пояснительной записки:  «Введение», «Сценарий использования», «Модель данных», «Разработанное приложение», «Выводы», «Приложения», «Литература» |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц. |
| Дата выдачи задания: 05.02.2025 |
| Дата сдачи реферата: 4.06.2025 |
| Дата защиты реферата: 04.06.2025   | Студент гр. 2382 |  | Муравин Е.Е. | | --- | --- | --- | | Студент гр. 2382 |  | Федоров М.В. | | Студент гр. 2382 |  | Бухарин М.А. | | Преподаватель |  | Заславский М.М | |

**АННОТАЦИЯ**

В рамках дисциплины «Введение в нереляционные базы данных» в составе команды было разработано приложение на тему «Графовый трекер задач». В качестве основной базы данных использовалась графовая СУБД Neo4j. Проект позволяет пользователю отслеживать задачи как в табличном формате, так и в виде графа с визуализацией связей между элементами.

**SUMMARY**

As part of the course "Introduction to Non-Relational Databases," our team developed an application titled "Graph-Based Task Tracker." The main database used was the graph database management system Neo4j. The project allows users to track tasks both in a tabular format and as a graph with visualized relationships between elements.**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_xxb88kr4uzaj)

[1. Актуальность решаемой проблемы 6](#_79czqr76v5c2)

[2. Постановка задачи 6](#_1pgr8yeeaeb)

[3. Предлагаемое решение 6](#_xzma5epqe493)

[4. Качественные требования к решению 6](#_fzj0b2dl7jsr)

[1. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ 8](#_gytzsc4armnv)

[1.1. Макет UI 8](#_phvylzyc1qx2)

[1.2. Сценарий использования 1](#_6aliw9jeucqs)4

[1.3. Преобладающая операция 2](#_c8sv6es0nn9l)0

[2. МОДЕЛЬ ДАННЫХ 2](#_m5x7joi081d1)1

[2.1. Нереляционная модель 2](#_ew75kerv11e8)1

[2.2. Реляционная модель 3](#_3f2l9uownm1g)5

[2.3. Сравнение моделей 4](#_p5vn5qb60eq)8

[3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ](#_wx2214metfw) 51

[3.1. Краткое описание](#_tuiez95d65gr) 51

[3.2. Использованные технологии](#_94mgr9s3vgw6) 51

[3.3. Снимки экрана приложения](#_858cgcpepwg) 52

[ВЫВОДЫ](#_xj67k5l6nk33) 59

[4.1. Достигнутые результаты](#_6wewrh4x2e6j) 59

[4.2. Недостатки и пути для улучшения полученного решения](#_vz720rkqxwid) 59

[4.3. Будущее развитие решения](#_x2les9g1zw2k) 60

[ЗАКЛЮЧЕНИЕ](#_s0awi49vl070) 61

[ПРИЛОЖЕНИЯ](#_nqa922yhcsyp) 62

ЛИТЕРАТУРА65

# ВВЕДЕНИЕ

## 1. Актуальность решаемой проблемы

## Классические трекеры задач часто не учитывают сложные зависимости между задачами, что снижает эффективность планирования в проектах с нелинейной структурой. Это особенно заметно в сферах, где задачи взаимосвязаны и требуют строгой последовательности выполнения.

## 2. Постановка задачи

Задача - реализовать трекер задач (по аналогии с asana trello jira), который бы учитывал и активно использовал зависимости между задачами, а также нелинейность множества задач (например, задачи на стройке). Хранение, визуализация задач в виде таблицы и графа. Анализ графовых характеристик задач, построение списка приоритетов выполнения набора задач.

## 3. Предлагаемое решение

## Графовый трекер задач — позволяет наглядно представлять структуру проекта, учитывать связи между задачами и выстраивать приоритеты на основе анализа графа. Такой подход обеспечивает более точное и гибкое управление задачами в реальных, нестандартных сценариях.

## 4. Качественные требования к решению

* Высокая производительность при работе с данными и визуализациях
* Удобный и понятный пользовательский интерфейс
* Возможность расширения функционала (импорт/экспорт данных, статистика, настройка графа)
* Надежность и устойчивость к ошибкам при взаимодействии с базой данных
* Актуальность и полнота отображаемой информации

# 1. СЦЕНАРИЙ ИСПОЛЬЗОВАНИЯ

## Макет UI

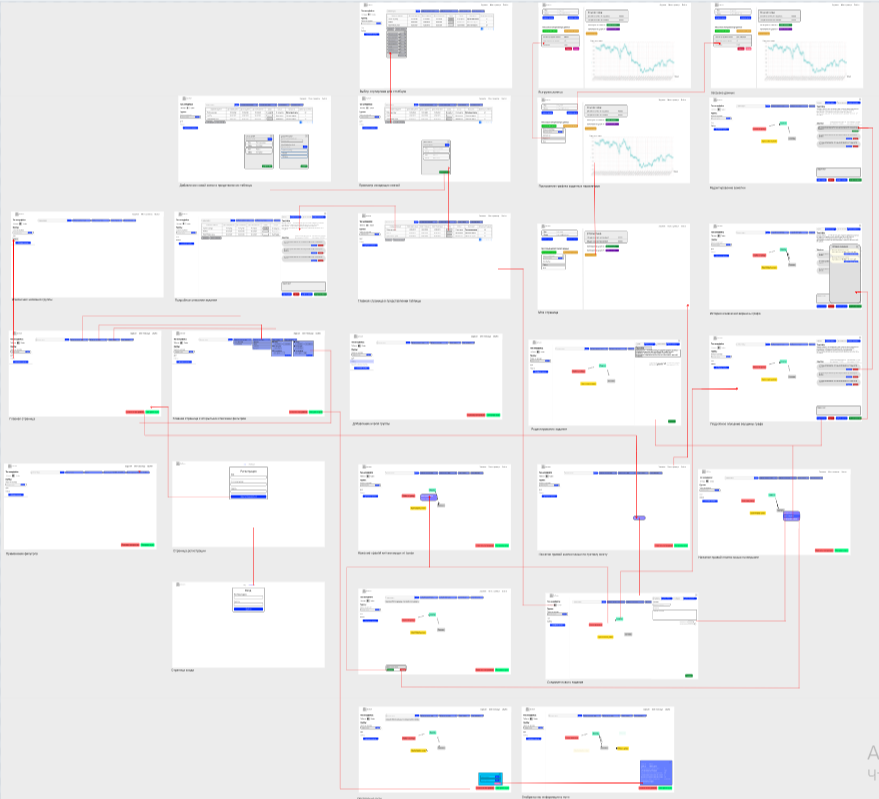


Рисунок 1.1.1 – Макет UI

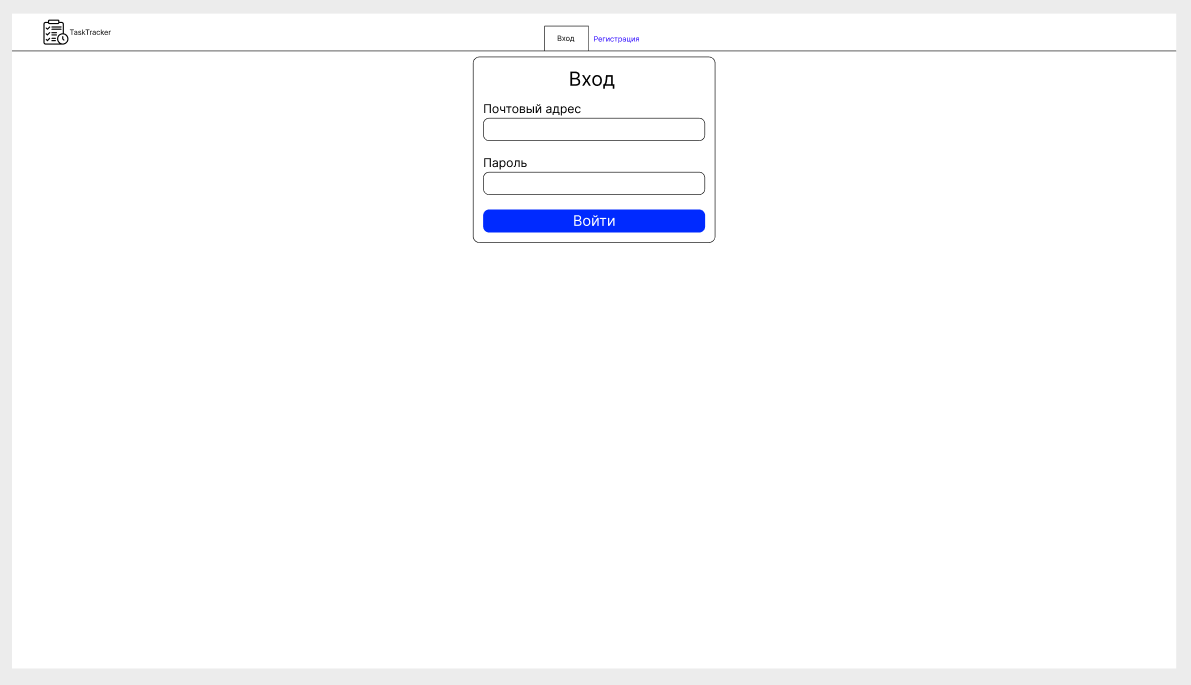


Рисунок 1.1.2 – Страница входа

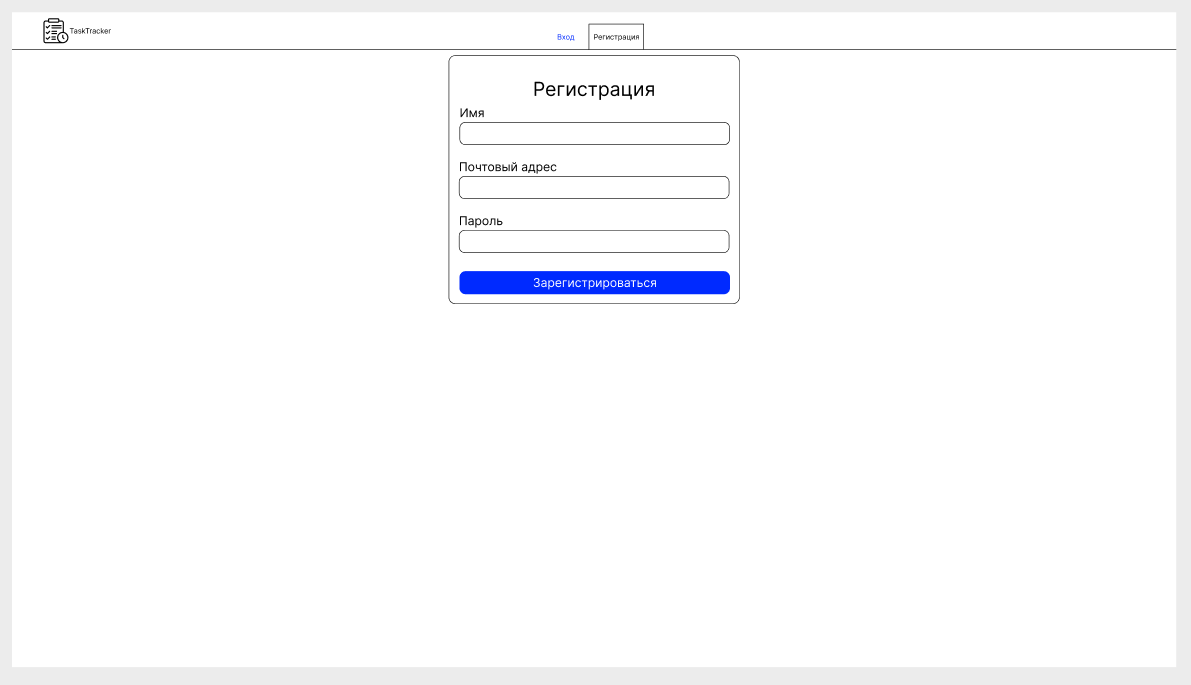


Рисунок 1.1.3 – Страница регистрации

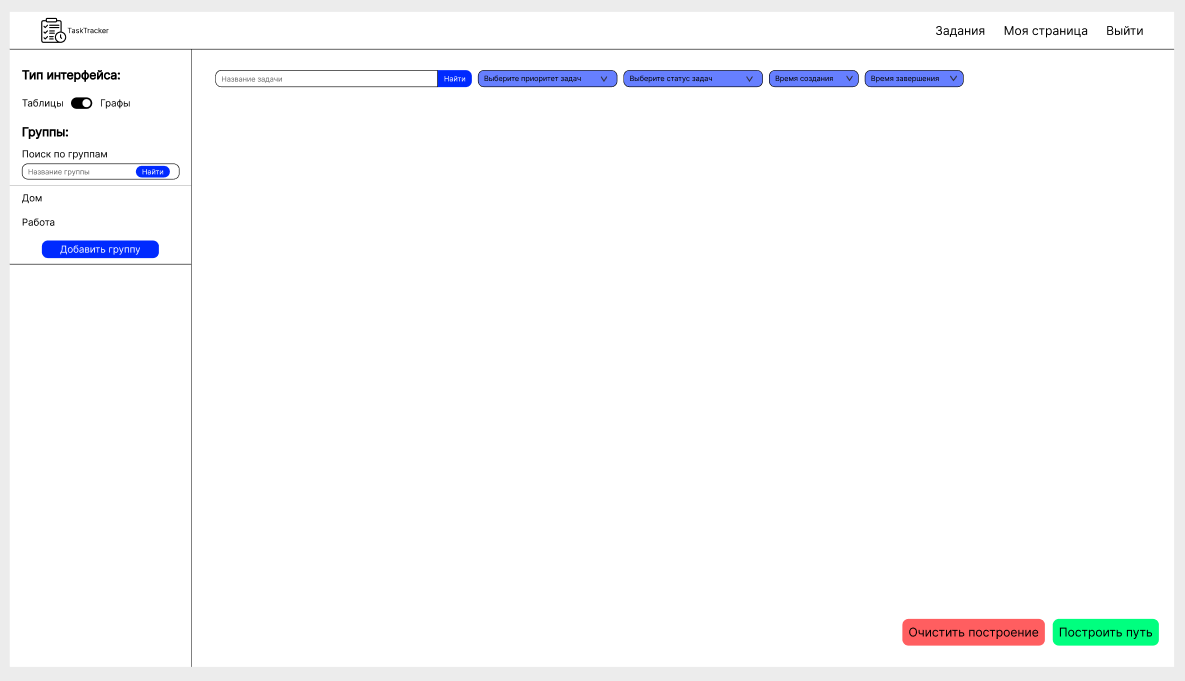


Рисунок 1.1.4 – Главная страница

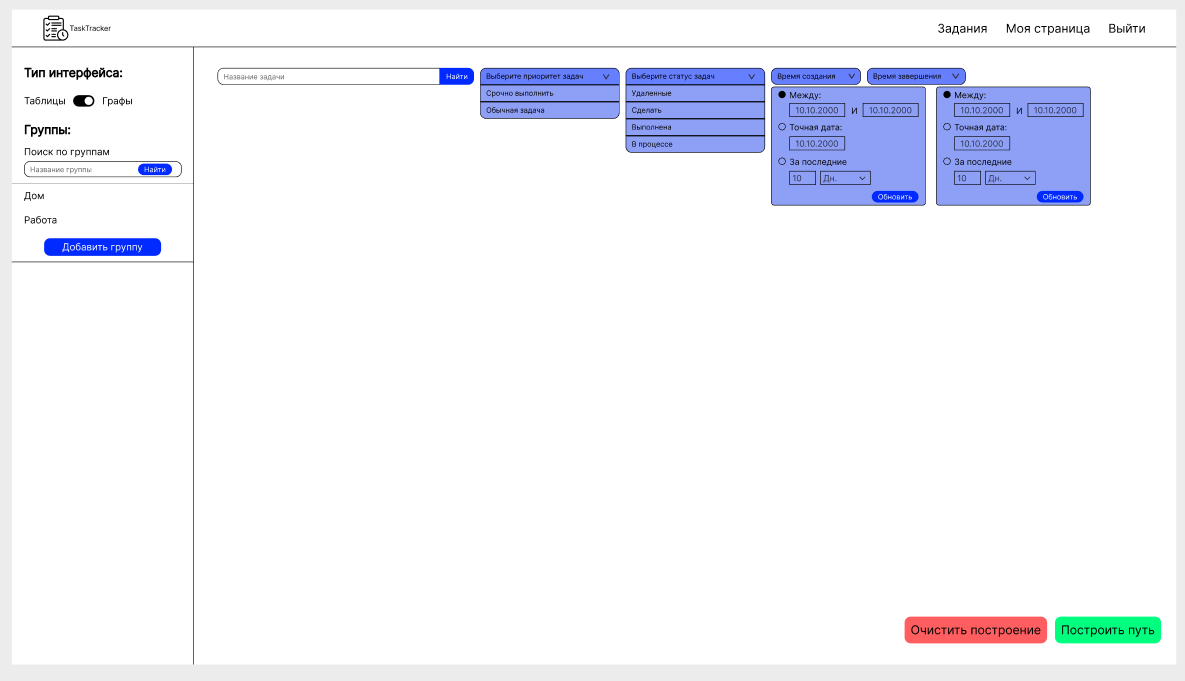


Рисунок 1.1.5 – Фильтры на главной странице

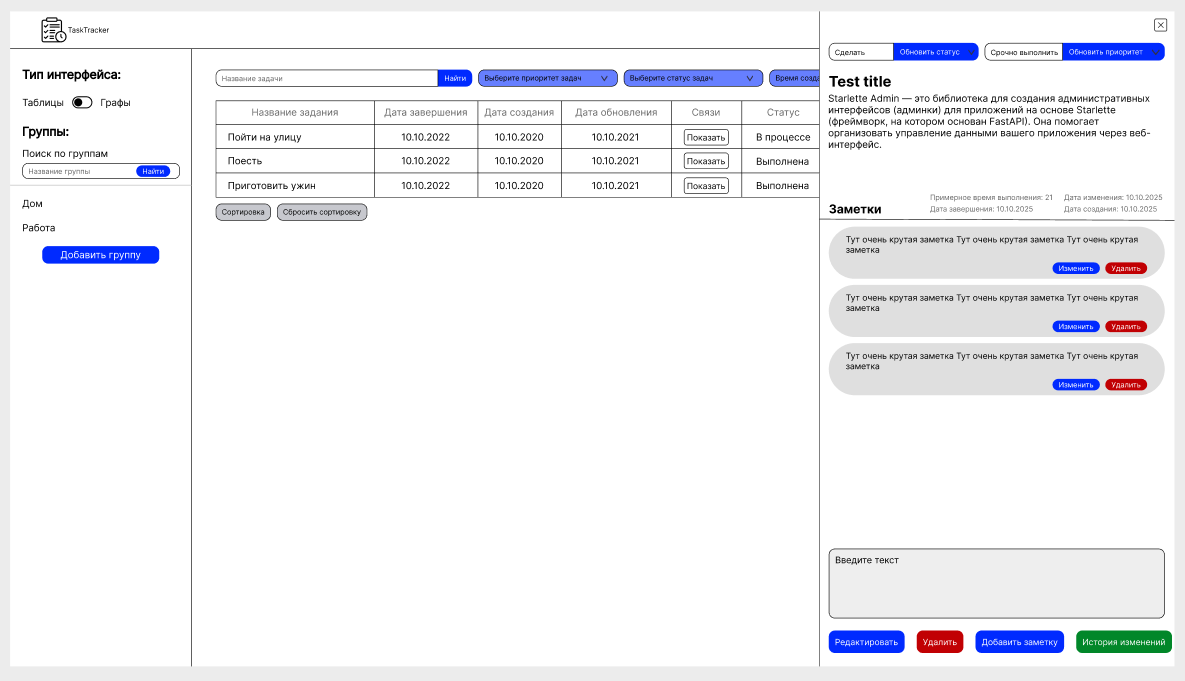


Рисунок 1.1.6 – Табличное представление заданий с подробным описанием

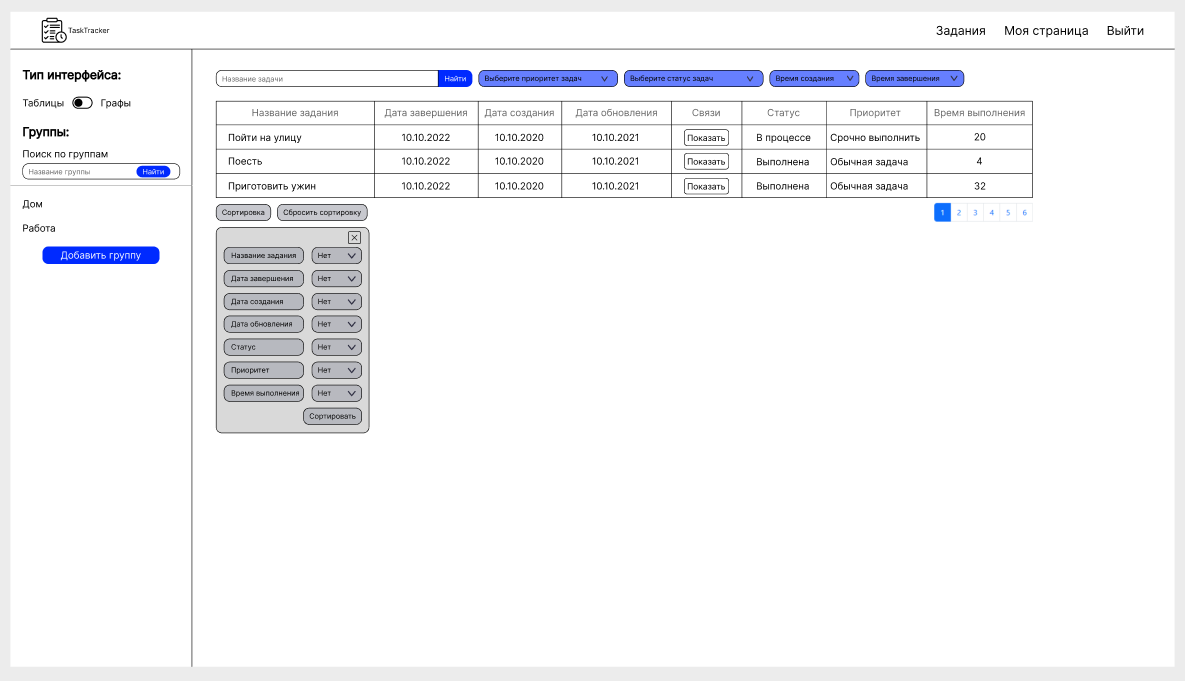


Рисунок 1.1.7 – Выбор сортировки для столбцов

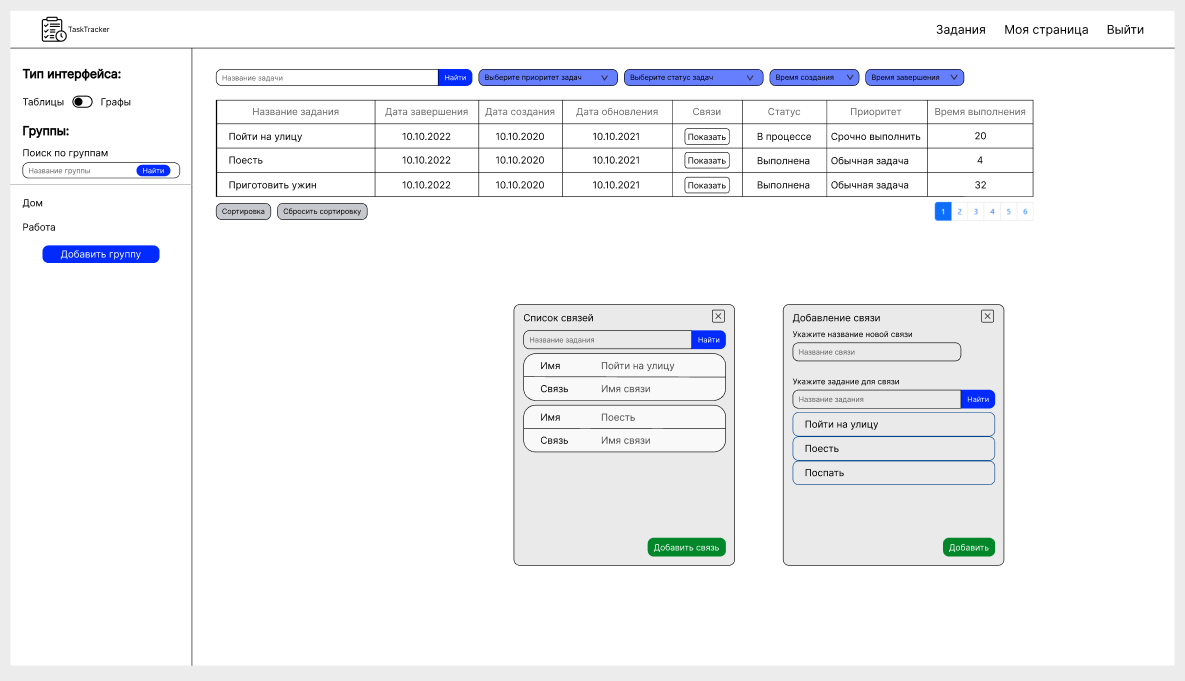


Рисунок 1.1.8 – Управление связями между задачами

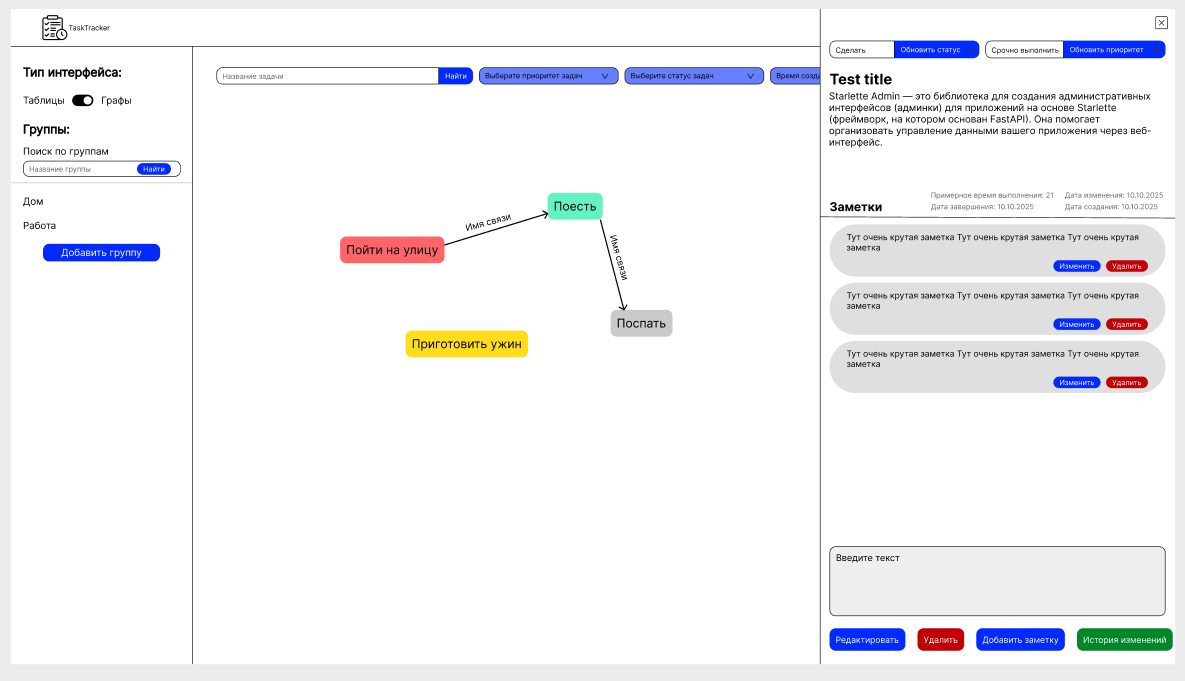


Рисунок 1.1.9 – Графовое представление заданий с подробным описанием

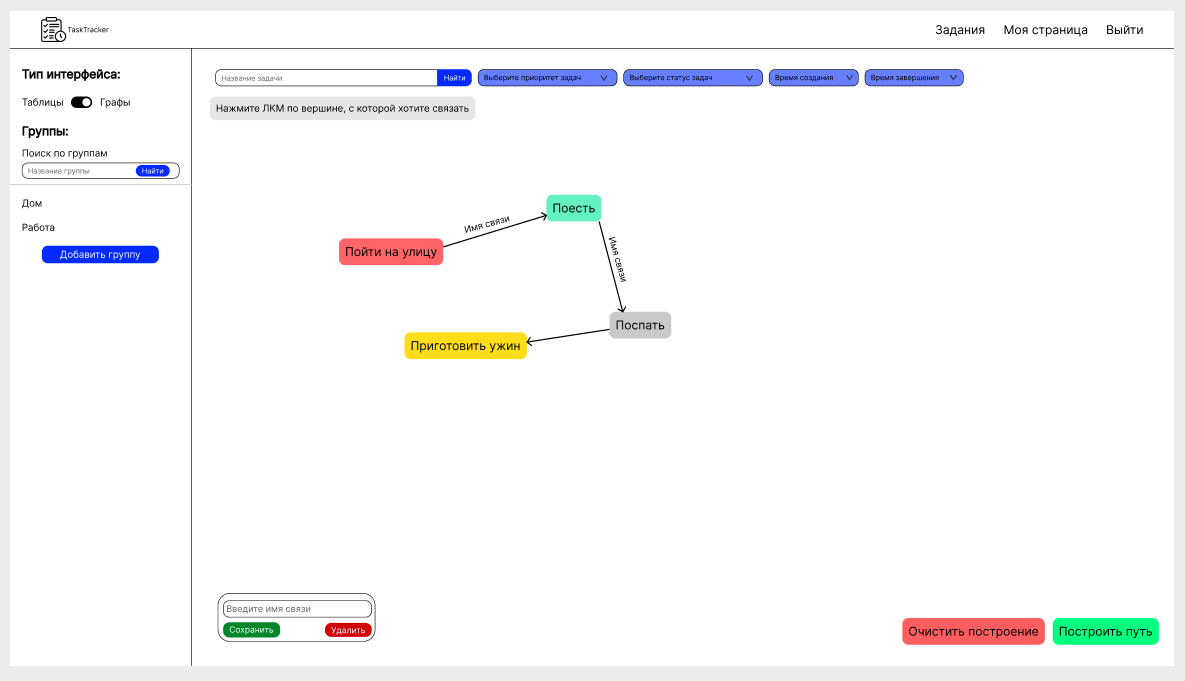


Рисунок 1.1.10 – Создание связи между заданиями

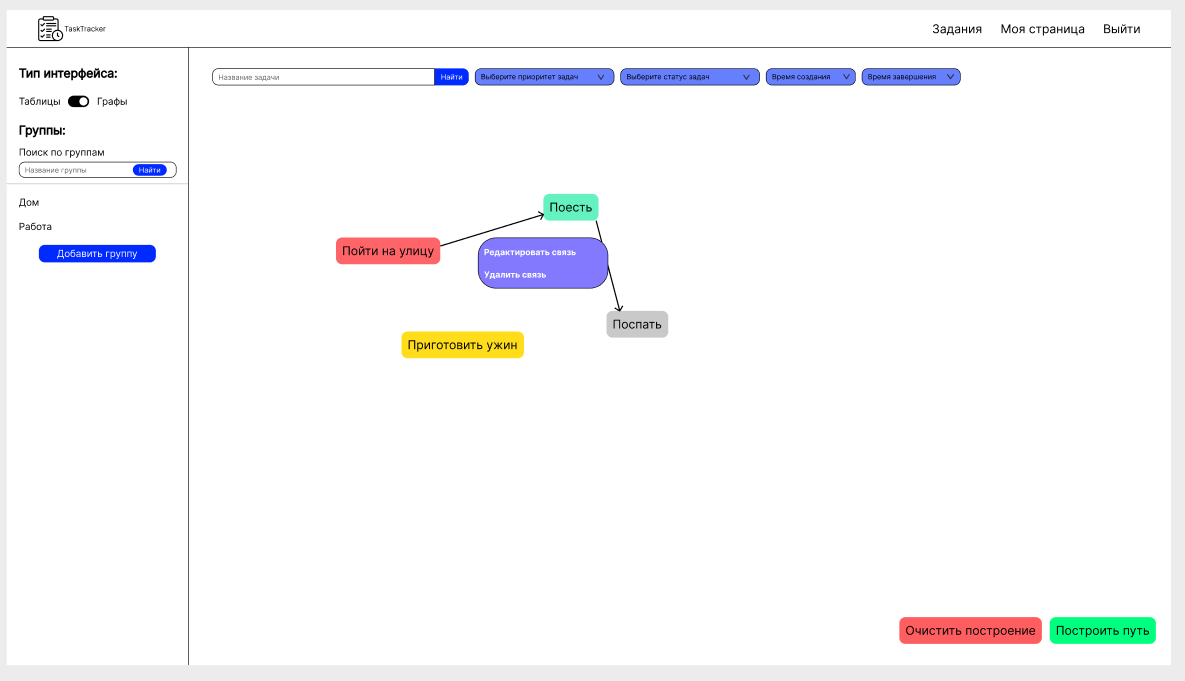


Рисунок 1.1.11 – Управление связями между заданиями

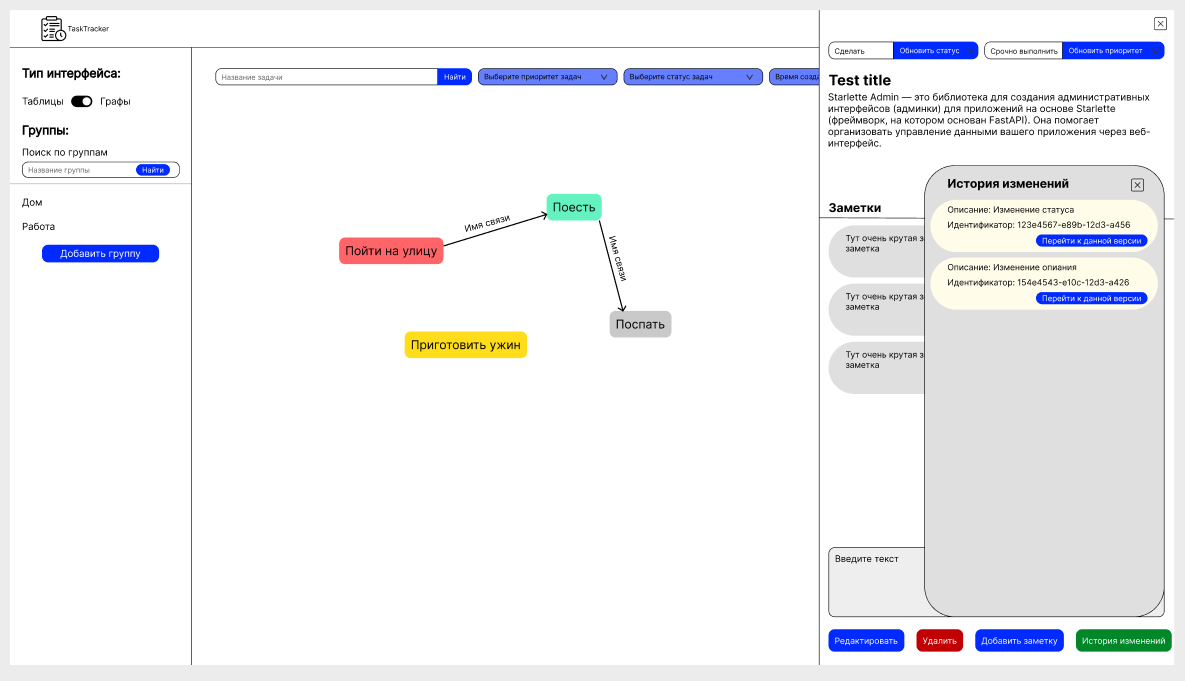


Рисунок 1.1.12 – История изменения вершины графа

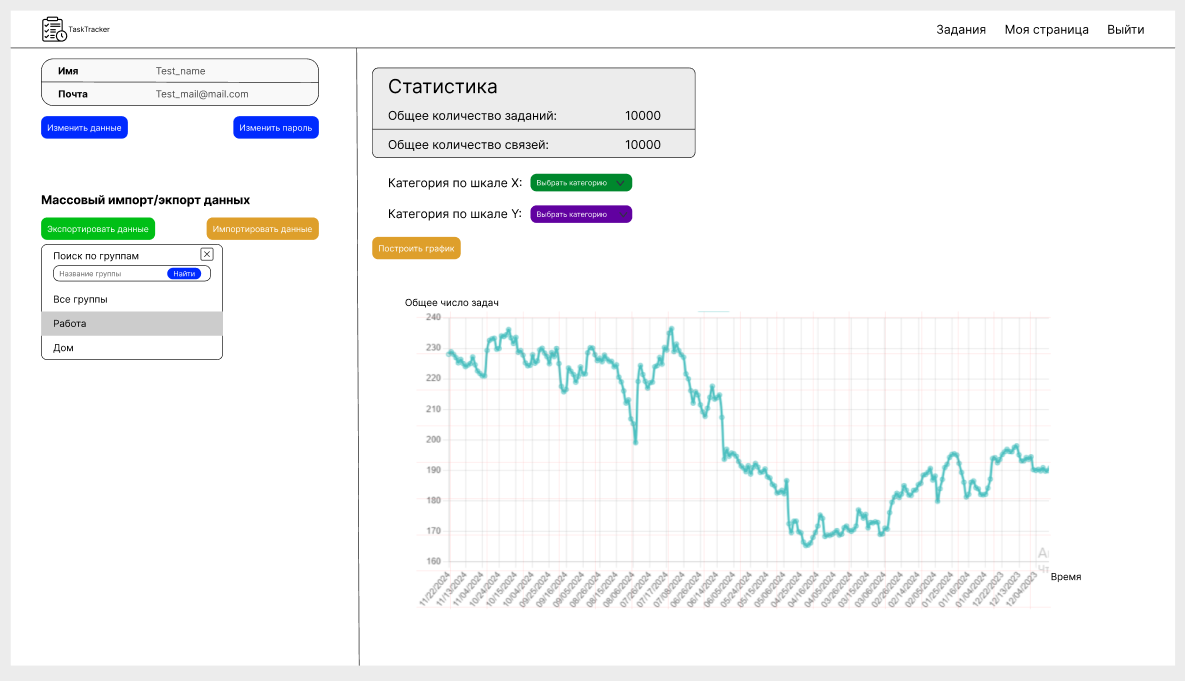


Рисунок 1.1.13 – Построение графиков

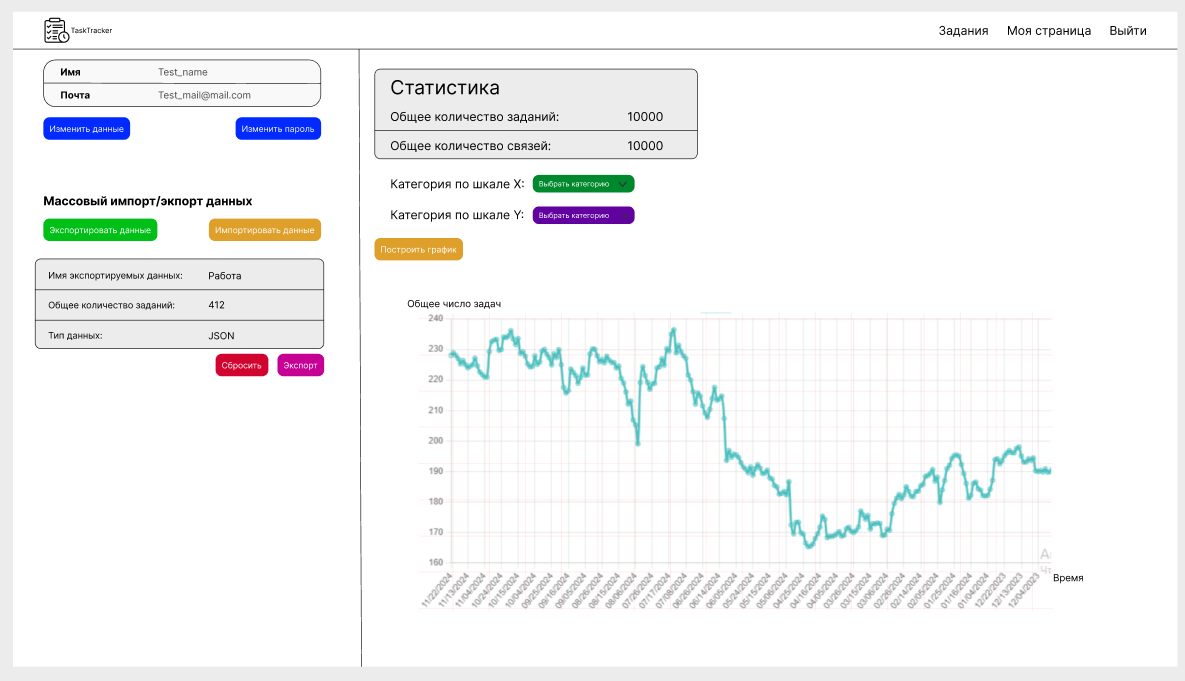


Рисунок 1.1.14 – Загрузка и выгрузка данных

## Сценарий использования

## Действующее лицо: Пользователь

### Основной сценарий:

1. **Пользователь заходит на сайт и видит Страницу входа**

**1А)** Пользователь вводит верные данные и нажимает “Войти” → переход на Главную страницу.

**1Б)** Пользователь нажимает “Регистрация” → переход на страницу Регистрации.

**1Б.1)** Пользователь вводит данные и нажимает “Зарегистрироваться” → переход на Главную страницу.

1. **Пользователь находится на Главной странице**

**2А)** Пользователь нажимает “Выберите приоритет задач” → разворачивается окно для фильтрации по приоритету задач.

**2А.1)** Пользователь настраивает фильтры и нажимает “Обновить” → показанные задачи фильтруются в соответствии с выбранными настройками.

**2Б)** Пользователь нажимает “Выберите статус задач” → разворачивается окно для фильтрации по статусу задач.

**2Б.1)** Действие, аналогичное описанному в пункте 2А.1.

**2В)** Пользователь нажимает “Время создания” → разворачивается окно для фильтрации по времени создания задач.

**2В.1)** Действие, аналогичное описанному в пункте 2А.1.

**2Г)** Пользователь нажимает “Время завершения” → разворачивается окно для фильтрации по времени, к которому задача должна быть завершена.

**2Г.1)** Действие, аналогичное описанному в пункте 2А.1.

**2Д)** Пользователь нажимает “Добавить группу” → появляется поле ввода названия новой группы.

**2Е)** Пользователь нажимает “Моя страница” → переход на вкладку “Моя страница”.

**2E.1)** Пользователь выбирает категории и нажимает “Построить график” → появляется график со статистикой по заданным параметрам.

**2E.2)** Пользователь нажимает “Изменить данные” → появляется форма для изменения данных.

**2E.2А)** Пользователь изменяет свои данные и сохраняет изменения.

**2E.3)** Пользователь нажимает “Изменить пароль” → появляется форма для изменения пароля.

**2E.3А)** Пользователь изменяет пароль и сохраняет изменения.

**2Е.4)** Пользователь нажимает “Экспортировать данные” и выбирает одну или несколько групп, данные из которых требуется экспортировать → появляется сводка по экспортируемым данным, пользователь указывает тип экспортируемых данных происходит экспорт данных выбранных групп файл.

**2Е.4А)** Пользователь нажимает на “Экспорт” → данные экспортируются в выбранный файл.

**2Е.5)** Пользователь нажимает “Импортировать данные” и выбирает json файл с данными для импорта → появляется информационное табло, происходит импорт данных из выбранного файла.

**2Е.5А)** Пользователь нажимает на “Импорт” → данные импортируются из выбранного файла.

**2Е.6)** Пользователь нажимает “Задания” → переход на Главную страницу.

**2Ж)** Пользователь нажимает ПКМ на пустом месте основного поля в центре экрана → появляется всплывающее окно с предложением добавить новое задание.

**2Ж.1)** Пользователь нажимает на окно добавления нового задания → справа появляется окно с формой для ввода информации о задании.

**2Ж.1А)** Пользователь вводит информацию о задании и сохраняет новое задание.

**2З)** Пользователь имеет возможность переключаться между имеющимися группами заданий.

**2И)** Пользователь делает двойной клик по названию группы → название группы выделяется и вместо названия появляется форма для изменения названия группы.

1. **Пользователь находится на Главной странице и создал несколько заданий**

**3А)** Пользователь нажимает ПКМ на задание → появляется всплывающее окно с предложениями “Добавить связь”, “Удалить задание” и “Редактировать задание”.

**3А.1)** Пользователь нажимает “Добавить связь” → в левом нижнем углу появляется окно с формой для ввода названия связи, кнопками сохранения и отмены действия.

**3А.1А)** Пользователь вводит название связи, нажимает на другое задание, с которым хочет создать связь, затем нажимает “Сохранить” → появляется ребро-связь, соединяющее задания.

**3А.2)** Пользователь нажимает “Удалить задание” → задание и связи, ведущие к нему, удаляются.

**3А.3)** Пользователь нажимает “Редактировать задание” → появляется форма для редактирования задания.

**3А.3А)** Пользователь редактирует информацию о задании и нажимает “Сохранить” → задание обновляется и теперь содержит новую информацию.

**3Б)** Пользователь нажимает ПКМ на ребро-связь → появляется всплывающее окно с предложениями “Редактировать связь” и “Удалить связь”.

**3Б.1)** Пользователь нажимает “Редактировать связь” → появляется форма для редактирования связи.

**3Б.1А)** Пользователь редактирует связь и нажимает “Сохранить” → информация о связи обновляется.

**3Б.2)** Пользователь нажимает “Удалить связь” → связь удаляется.

**3В)** Пользователь нажал ЛКМ на задание → справа появилось окно с подробным описанием задания.

**3В.1)** Пользователь нажимает “Редактировать” → появляется форма редактирования задания.

**3В.2)** Пользователь нажимает “Удалить” → задание удаляется.

**3В.3)** Пользователь вводит текст в форму и нажимает “Добавить заметку” → появляется новая заметка.

**3В.4)** Пользователь нажимает “Изменить” у заметки → текст заметки превращается в поле редактирования.

**3В.4А)** Пользователь изменяет заметку и нажимает “Сохранить” → заметка обновляется.

**3В.5)** Пользователь нажимает “Удалить” у заметки → заметка удаляется.

**3В.6)** Пользователь нажимает “История изменений” → появляется окно со списком всех версий задачи.

**3В.6А)** Пользователь выбирает версию и нажимает “Перейти к данной версии” → задание переходит на выбранную версию.

**3Д)** Пользователь нажимает “Построить путь” → открывается окно для настройки построения пути между двумя вершинами.

\*\*3Д.1) Пользователь вводит названия двух вершин и на графах → отображается путь, все вершины не из пути становятся слабовидными. Серые вершины - завершенные задания.

**3Е)** Пользователь нажимает “Очистить построение” → построение пути очищается, граф задач возвращается к своему стандартному виду.

**3Ж)** Пользователь нажимает на ползунок выбора типа интерфейса → представление меняется с графового на табличное.

1. **Пользователь находится в табличном режиме**

**4А)** Пользователь нажимает “Показать” в столбце “Связи” → появляется окно со списком связей.

**4А.1)** Пользователь нажимает “Добавить связь” → появляется окно создания связи.

**4А.1А)** Пользователь вводит данные и нажимает “Добавить” → связь добавляется.

**4Б)** Пользователь нажимает на строку нужного задания → появляется окно, описанное в пункте 4В.

**4В)** Пользователь нажимает “Сортировка” → появляется окно для выбора параметров для сортировки по строкам.

**4В.1)** Пользователь выставляет нужные параметры сортировки и нажимает “Сортировать” → столбцы таблицы сортируются в соответствии с выставленными параметрами.

**4Д)** Пользователь нажимает “Сбросить сортировку” → выставленные настройки сортировки сбрасываются.

**4Е)** Пользователь нажимает на одну из кнопок пагинатора таблицы → открывается соответствующая страница пагинатора.

### Альтернативный сценарий:

1. **Ошибка при входе**

**1А)** Введены неверные данные → появляется уведомление о некорректности.

1. **Ошибка при регистрации**

**2А)** Имя пользователя уже используется.

**2Б)** Почта уже зарегистрирована.

**2В)** Слабый пароль.

1. **Ошибка при изменении данных**

**3А)** Введены данные, уже используемые другим аккаунтом.

1. **Ошибка при изменении пароля**

**4А)** Введён слабый пароль.

1. **Ошибка в случае, если пользователь хочет импортировать файл с неверными данными или файл, формат которого не поддерживается**

**5А)** Предоставлены неверные данные для импорта.

1. **Выход из аккаунта**

**6А)** Пользователь нажимает “Выйти” → переход на Страницу входа.

1. **Перезагрузка страницы в режиме настройки импорта/экспорта данных**

**7А)** Информация о импорте/экспорте сбрасывается.

1. **Отмена действий**

**8А)** Пользователь отменяет любое из действий: добавление, редактирование, изменение версии, изменение связи, изменение параметров фильтрации, изменение параметров сортировки, изменение параметров построения пути. Окно закрывается, изменения не применяются.

**1.3. Преобладающие операции**

В данном сценарии основную нагрузку будут составлять операции чтения данных, поскольку пользователь, как правило, чаще просматривает уже существующие задачи — как в табличном представлении, так и в виде графа.

При создании новой задачи система предварительно загружает текущий список задач, а после добавления — повторно обновляет данные из базы. Таким образом, чтение данных из базы происходит чаще, чем операции создания или обновления, что делает акцент на оптимизацию именно запросов на выборку.

# 2. МОДЕЛЬ ДАННЫХ

**2.1 Нереляционная модель**

**A. Графическое представление модели**

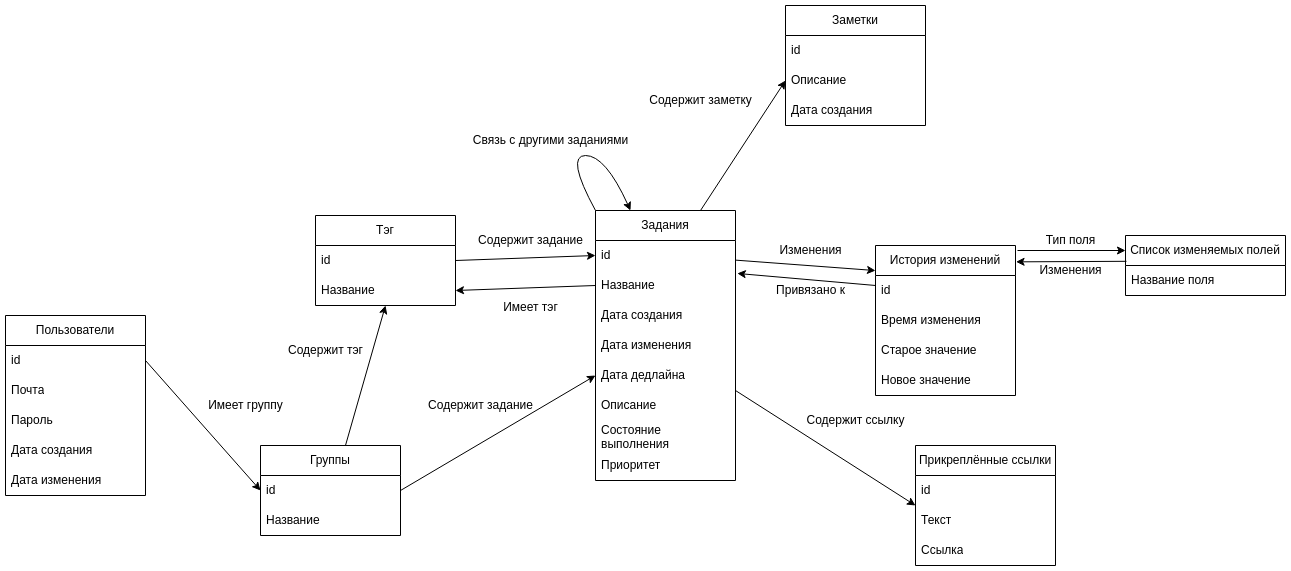


Рисунок 2.1.1 — Графическое представление нереляционной модели

**B. Сущности модели**

Пользователь (user):

* Почта (email)
* Пароль (password)
* Дата создания (createdAt)
* Дата изменения (modifiedAt)
* Имеет группу — связь с group

Группа (group):

* Название (name)
* Содержит задание — связь с task
* Содержит тег — связь с tag

Тег (tag):

* Название (name)
* Содержит задание — связь с task

Задача (task):

* Название (title)
* Описание (content)
* Дата создания (createdAt)
* Дата дедлайна (deadline)
* Дата изменения (change)
* Состояние выполнения (status)
* Приоритет (priority)
* Имеет тег — связь с tag
* Связь с другими заданиями — связь с task
* Содержит заметки — связь с note
* Имеет историю изменений — связь с changeHistory
* Содержит прикреплённые ссылки — связь с attachment

Заметка (note):

* Описание (content)
* Дата создания (createdAt)

Прикреплённая ссылка (attachment)

* Текст (text)
* Ссылка (URL)

История изменений (changeHistory)

* Время изменения (modifiedAt)
* Старое значение (oldValue)
* Новое значение (newValue)
* Связана с задачей — связь с task
* Ссылается на тип поля — связь с fieldType

Список допустимых названий полей (fieldType)

* Название поля (fieldName)
* Связан с историей изменений — связь с changeHistory

**C. Подсчет объема данных**

Размеры сущностей и связей

Размер полей:

* char = 1 байт
* int, datetime = 8 байт
* UUID = 16 байт
* Служебные данные узла Neo4j ≈ UUID + int + char[10] = 34 байта
* Каждая связь ≈ UUID + int + char[10] = 34 байт
* Для связи будем считать чистыми данными = 10 байт
* Служебные чистые данные узла будем считать = 0 байт

Обозначения:

* Количество пользователей: Nu
* Количество групп у одного пользователя: Ng
* Количество заданий: Nt
* Количество заметок у каждого задания: Nn
* Количество тегов у группы: Ntag
* Количество прикрепленных ссылок у одного задания: Na
* История изменений: Nh
* Среднее количество связей исходящих от задания: Nr
* Среднее количество тегов у задания: Ntagc

Количество всех связей для одного пользователя в общем виде:

* У пользователя Ng групп
* Каждое задание находится в группе - Nt связей
* У каждой группы по Ntag тегов - Ntag \* Ng связей
* У каждого задания по Nn заметок - Nn \* Nt связей
* Каждое задание изменяется Nh раза - Nh \* Nt связей (туда и обратно Nh \* Nt \* g2)
* Каждая история привязана к изменяемому полю - Nh \* Nt связей (туда и обратно Nh \* Nt \* 2)
* Количество заданий с ссылками - Nt \* Na
* Количество связей между заданиями - Nt \* Nr
* Количество заданий с тегами - Nt \* Ntagc
* Итого: Количество связей = Nt∗(1+Nn+4Nh+Na+Nr+Ntagc)+Ng∗(1+Ntag)

Средние значения:

* Количество пользователей: 1000 = Nu
* Количество групп у одного пользователя: 3 = Ng
* Количество заданий: 15 = Nt
* Количество заметок у каждого задания: 2 = Nn
* Количество тегов у группы: 1 = Ntag
* Количество прикрепленных ссылок у одного задания: 0.5 = Na
* История изменений: 2 = Nh
* Среднее количество связей исходящих от задания: 2 = Nr
* Среднее количество тегов у задания: 1 = Ntagc

Размеры сущностей

Таблица 2.1.1 — Пользователь (user)

| Поле | Тип | Размер |
| --- | --- | --- |
| email | char[50] | 50 |
| password | char[30] | 30 |
| createdAt | datetime | 8 |
| modifiedAt | datetime | 8 |
| Служебные данные | системные | 34 |

* Итого на 1 пользователя: 130 байта
* Итого на 1 пользователя чистый: 96 байта
* Общий вид: Nu \* 130 байт
* В среднем 1000 пользователей: 127 КБ
* В среднем 1000 пользователей чистый: 94 К

Таблица 2.1.2 — Группа (group)

| Поле | Тип | Размер |
| --- | --- | --- |
| name | char[30] | 30 |
| Служебные | системные | 34 |

Таблица 2.1.3 — Задача (task)

| Поле | Тип | Размер |
| --- | --- | --- |
| title | char[50] | 50 |
| content | char[500] | 500 |
| createdAt | datetime | 8 |
| change | datetime | 8 |
| deadline | datetime | 8 |
| status | char[50] | 50 |
| priority | char[50] | 50 |
| Служебные | системные | 34 |

* Итого на 1 задачу: 708 байта Итого на 1 задачу чистый: 674 байта
* Общий вид: Nu \* Nt \* 708 байт
* На 15 000 задач: 10.1 МБ
* На 15 000 задач чистый: 9.64 МБ

Таблица 2.1.4 — Тег (tag)

| Поле | Тип | Размер |
| --- | --- | --- |
| content | char[100] | 100 |
| createAt | datetime | 8 |
| Служебные | системные | 34 |

* Итого на 1 группу: 64 байт
* Итого на 1 группу чистый: 30 байт
* Общий вид: Nu \* Ng \* Ntag \* 64 байтИтог на 3000 тегов: 187 КБ
* Итог на 3000 тегов чистый: 88 КБ

Таблица 2.1.5 — Заметка (note)

| Поле | Тип | Размер |
| --- | --- | --- |
| content | char[100] | 100 |
| createAt | datetime | 8 |
| Служебные | системные | 34 |

* Итого на 1 заметку: 142 байт
* Итого на 1 заметку чистый: 108 байт
* Общий вид: Nu \* Nt \* Nn \* 142 байт
* На 30 000 заметок: 3.8 МБ
* На 30 000 заметок чистый: 2.9 МБ

Таблица 2.1.6 — Прикреплённая ссылка (attachment)

| Поле | Тип | Размер |
| --- | --- | --- |
| text | char[100] | 100 |
| URL | char[100] | 100 |
| Служебные | системные | 34 |

* Итого на 1 ссылку: 234 байт
* Итого на 1 ссылку чистый: 200 байт
* Общий вид: Nu \* Nt \* Na \* 234 байт
* На 7 500 ссылок: 1.7 МБ
* На 7 500 ссылок чистый: 1.4 МБ

Таблица 2.1.7 — История изменений (changeHistory)

| Поле | Тип | Размер |
| --- | --- | --- |
| modifiedAt | datetime | 8 |
| oldValue | char[100] | 100 |
| newValue | char[100] | 100 |
| Служебные | системные | 34 |

* Итого на 1 запись: 242 байт
* Итого на 1 запись чистый: 208 байт
* Общий вид: Nu \* Nt \* Nh \* 242 байт
* На 30 000 записей: 6.9 МБ
* На 30 000 записей чистый: 5.9 МБ

Таблица 2.1.8 — Общий итог

| Сущность | Объём (КБ) | Чистый объем (КБ) |
| --- | --- | --- |
| Пользователи | 127 | 94 |
| Группы | 187 | 88 |
| Задачи | 10342 | 9871 |
| Заметки | 3891 | 2969 |
| Ссылки | 1740 | 1433 |
| Теги | 187 | 88 |
| История изменений | 7065 | 6041 |

* Размер без учета связей: ~22.9 Мб
* Размер без учета связей чисты: ~20.1 Мб
* Количество связей у 1000 пользователей: 223500 Размер связей: ~7.2 Мб
* Размер связей чистый: ~2.1 Мб
* Общий размер: ~30.1 МБ (узлы + связи) Общий размер чистый: ~22.2 МБ (узлы + связи)

Выразим объем данных через количество пользователей в системе по уже описанным средним значениям:V(Nu) = 31639 \* Nu байт

**D. Избыточность данных**

Для связей и узлов были убраны данные по типу uuid и id, которые генерируеются автоматически.

Для связей были оставлены названия в виде char[10], для узлов названия учетны не были.

* Общий размер: ~30.1 МБ (узлы + связи)
* Общий размер чистый: ~22.2 МБ (узлы + связи)

Тогда избыточность данных равна: R(Nu) = 30.1 / 22.2 = 1.35

**E. Направление роста модели при увеличении количества объектов каждой сущности**

При увеличении количества объектов каждая модель сущности будет расти линейно.

**F. Примеры данных**

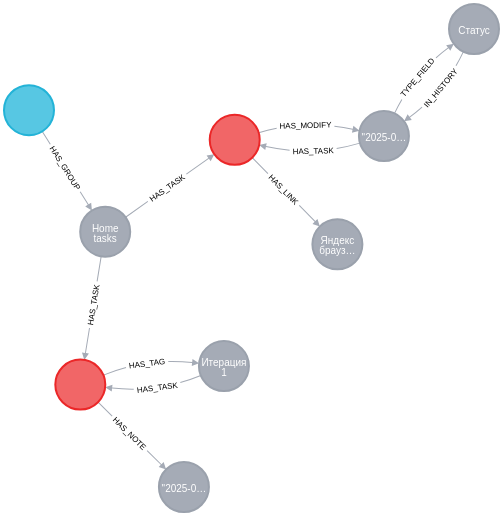


Рисунок 2.1.6 — SVG формат

**JSON формат:**

[

{

"n": {

"identity": 0,

"labels": [

"Note"

],

"properties": {

"createdAt": "2025-04-11T19:44:04.018000000Z",

"Content": "Итерация 1"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:0"

}

},

{

"n": {

"identity": 1,

"labels": [

"attachment"

],

"properties": {

"Text": "Яндекс браузер",

"URL": "https://ya.ru/?npr=1&utm\_referrer=https%3A%2F%2Fyandex.ru%2F"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:1"

}

},

{

"n": {

"identity": 2,

"labels": [

"History"

],

"properties": {

"modifiedAt": "2025-04-10T19:44:04.018000000Z",

"OldValue": "Важное",

"NewValue": "Очень важное"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:2"

}

},

{

"n": {

"identity": 3,

"labels": [

"Type"

],

"properties": {

"FieldName": "Статус"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:3"

}

},

{

"n": {

"identity": 4,

"labels": [

"Task"

],

"properties": {

"Status": "В процессе",

"createdAt": "2025-04-11T19:44:04.018000000Z",

"Content": "Нужно посмотреть телевизор",

"Priority": "Очень важное",

"change": "2025-04-11T19:44:04.018000000Z",

"Title": "Просмотр телевизора"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:4"

}

},

{

"n": {

"identity": 5,

"labels": [

"Tag"

],

"properties": {

"Content": "Итерация 1"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:5"

}

},

{

"n": {

"identity": 7,

"labels": [

"User"

],

"properties": {

"createdAt": "2025-04-11T19:44:03.902000000Z",

"password": "veryhardhashpassword",

"modifiedAt": "2025-04-11T19:44:03.902000000Z",

"email": "helloworld@gmail.com"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:7"

}

},

{

"n": {

"identity": 8,

"labels": [

"Group"

],

"properties": {

"name": "Home tasks"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:8"

}

},

{

"n": {

"identity": 9,

"labels": [

"Task"

],

"properties": {

"Status": "В процессе",

"createdAt": "2025-04-11T19:44:04.018000000Z",

"Content": "Нужно убрать весь дом",

"Priority": "Важное",

"change": "2025-04-11T19:44:04.018000000Z",

"Title": "Уборка"

},

"elementId": "4:21b543f2-a2ae-46ca-8874-0e5c42b8e9bc:9"

}

}

]

**G. Примеры запросов**

**Пользователь регистрируется**

* Существует ли уже такой email

MATCH (u:user {Email: "some@gmail.com"}) RETURN t

* Создаем нового пользователя

CREATE (:user {

email: "helloworld@gmail.com",

password: "veryhardhashpassword",

createdAt: datetime(),

modifiedAt: datetime()

});

Количество запросов: 2

Задействованные сущности: user

**Пользователь логинится**

* Существует ли уже такой email

MATCH (u:user {email: "some@gmail.com"}) RETURN t

Количество запросов: 1

Задействованные сущности: user

**Пользователь добавляет новую группу**

MATCH (u:user {email: "helloworld@gmail.com"})

CREATE (:group {

name: "Home tasks"

})<-[:HAS\_GROUP]-(u)

Количество запросов: 2

Задействованные сущности: group, user

**Пользователь добавляет новое задание в некоторой группе "some group"**

MATCH (g:group {name: "some group"})

CREATE (watch:task {

title: "Просмотр телевизора",

content: "Нужно посмотреть телевизор",

createdAt: datetime(),

change: datetime(),

dedline: null,

status: "В процессе",

priority: "Очень важное"

})<-[:HAS\_TASK]-(g)

Количество запросов: 2

Задействованные сущности: group, task

**Пользователь создает новый тег**

CREATE (t:tag {

content: "Итерация 1"

})

Количество запросов: 1

Задействованные сущности: tag

**Пользователь закрепляет некоторый тег к заданию**

MATCH (tag:tag {content: "Итерация 1"}), (t:task {title: "some task"})

CREATE (tag)-[:HAS\_TASK]->(t)

CREATE (tag)<-[:HAS\_TAG]-(t)

Количество запросов: 3

Задействованные сущности: tag, task

**Пользователь прикрепляет ссылку к заданию**

MATCH (t:task {title: "some task"})

CREATE (l:attachment{

Text: "Яндекс браузер",

URL: "https://ya.ru/?npr=1&utm\_referrer=https%3A%2F%2Fyandex.ru%2F"

})<-[:HAS\_LINK]-(t)

Количество запросов: 2

Задействованные сущности: task, attachment

**Пользователь изменяет статус у задания**

* Изменяется значение у сущности задания

MATCH (t:task {title: "some task"})

SET t.status = "Завершено"

RETURN t

Количество запросов: 1

Задействованные сущности: task

* Изменяется значение у сущности задания

MATCH (task:task {title: "Уборка"})

CREATE (h:history {

modifiedAt: datetime(),

oldValue: "В процессе",

newValue: "Завершено"

})<-[:HAS\_MODIFY]-(task)

CREATE (t:type {fieldName: "Статус"})

CREATE (task)<-[:HAS\_TASK]-(h)

CREATE (h)-[:TYPE\_FIELD]->(t)

CREATE (h)<-[:IN\_HISTORY]-(t)

Количество запросов: 6

Задействованные сущности: task, history, fieldType

**Пользователь изменяет статус у задания**

MATCH (one:Task {Title: "Просмотр телевизора"})-[]-(g:Task)

RETURN one, g

Количество запросов: 1

Задействованные сущности: task

**Пользователь ищет путь от одной задачи до другой**

MATCH (start:Task {Title: "Просмотр телевизора"}),

(end:Task {Title: "Что-то крутое"})

MATCH path = shortestPath((start)-[\*..N]->(end))

RETURN path

Количество запросов: N

Задействованные сущности: task

**Пользователь ищет топ-10 незавершенных задач**

MATCH (u:user {email: "helloworld@gmail.com"})-[\*]->(t:task)

WHERE t.status <> 'Выполнена'

ORDER BY t.createdAt

LIMIT 10

RETURN t

Количество запросов: 3

Задействованные сущности: task, user

**2.2 Реляционная модель**

**А. Графическое представление модели**

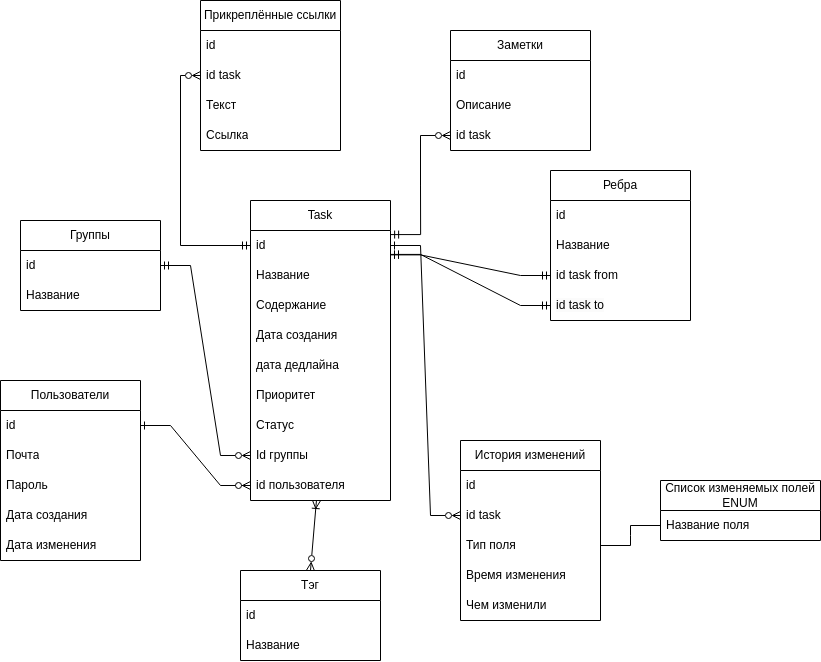


Рисунок 2.2.1 — Графическое представление реляционной модели

**B. Сущности модели**

Пользователь (user):

* Идентификатор (id)
* Почта (email)
* Пароль (password)
* Дата создания (createdAt)
* Дата изменения (modifiedAt)

Группа (group):

* Идентификатор (id)
* Название (name)
* Идентификатор пользователя (userId)

Тег (tag):

* Идентификатор (id)
* Название (name)

Задача (task):

* Идентификатор (id)
* Название (title)
* Описание (content)
* Дата создания (createdAt)
* Дата дедлайна (deadline)
* Дата изменения (change)
* Состояние выполнения (status)
* Приоритет (priority)
* Идентификатор группы (groupId)

Заметка (note):

* Идентификатор (id)
* Описание (content)
* Дата создания (createdAt)
* Идентификатор задания (taskId)

Прикреплённая ссылка (attachment):

* Идентификатор (id)
* Идентификатор задания (taskId)
* Текст (text)
* Ссылка (URL)

История изменений (changeHistory):

* Идентификатор (id)
* Время изменения (modifiedAt)
* Старое значение (oldValue)
* Новое значение (newValue)
* Тип поля (fieldType)
* Идентификатор задания (taskId)

Размеры сущностей

Таблица 2.2.1 — Пользователь (user)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| email | char[50] | 50 |
| password | char[30] | 30 |
| createdAt | datetime | 8 |
| modifiedAt | datetime | 8 |

* Итого на 1 пользователя: 104 байта
* Итого на 1 пользователя чистый: 96 байта
* Общий вид: Nu \* 104 байт
* В среднем 1000 пользователей: 101 КБ
* В среднем 1000 пользователей чистый: 94 КБ

Таблица 2.2.2 — Группа (group)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| name | char[30] | 30 |
| userId | UUID | 8 |

* Итого на 1 группу: 46 байт
* Итого на 1 группу чистый: 30 байт
* Общий вид: Nu \* Ng \* 46 байт
* Итог на 3000 групп: 135 КБ
* Итог на 3000 групп чистый: 88 КБ

Таблица 2.2.3 — Тег (tag)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| name | char[30] | 30 |

* Итого на 1 группу: 38 байт
* Итого на 1 группу чистый: 30 байт
* Общий вид: Nu \* Ng \* Ntag \* 38 байт
* Итог на 3000 тегов: 111 КБ
* Итог на 3000 тегов чистый: 88 КБ

Таблица 2.2.4 — Задача (task)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| title | char[50] | 50 |
| content | char[500] | 500 |
| createdAt | datetime | 8 |
| change | datetime | 8 |
| deadline | datetime | 8 |
| status | char[50] | 50 |
| priority | char[50] | 50 |
| groupId | UUID | 8 |

* Итого на 1 задачу: 690 байта Итого на 1 задачу чистый: 674 байта
* Общий вид: Nu \* Nt \* 690 байт
* На 15 000 задач: 9.8 МБ
* На 15 000 задач чистый: 9.64 МБ

Таблица 2.2.5 — Заметка (note)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| content | char[100] | 100 |
| createAt | datetime | 8 |
| taskId | UUID | 8 |

* Итого на 1 заметку: 124 байт
* Итого на 1 заметку чистый: 108 байт
* Общий вид: Nu \* Nt \* Nn \* 124 байт
* На 30 000 заметок: 3.5 МБ
* На 30 000 заметок чистый: 3.1 МБ

Таблица 2.2.6 — Прикреплённая ссылка (attachment)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| taskId | UUID | 8 |
| text | char[100] | 100 |
| URL | char[100] | 100 |

* Итого на 1 ссылку: 216 байт
* Итого на 1 ссылку чистый: 200 байт
* Общий вид: Nu \* Nt \* Na \* 216 байт
* На 7 500 ссылок: 1.54 МБ
* На 7 500 ссылок чистый: 1.4 МБ

Таблица 2.2.7 — История изменений (changeHistory)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| taskId | UUID | 8 |
| modifiedAt | datetime | 8 |
| oldValue | char[100] | 100 |
| newValue | char[100] | 100 |
| fieldType | char[50] | 50 |

* Итого на 1 запись: 274 байт
* Итого на 1 запись чистый: 258 байт
* Общий вид: Nu \* Nt \* Nh \* 274 байт
* На 30 000 записей: 7.84 МБ
* На 30 000 записей чистый: 7.38 МБ

Таблица 2.2.8 — Ребра (edge)

| Поле | Тип | Размер |
| --- | --- | --- |
| id | UUID | 8 |
| taskFromId | UUID | 8 |
| taskToId | UUID | 8 |
| title | char[50] | 50 |

* Итого на 1 запись: 74 байт
* Итого на 1 запись чистый: 50 байт
* Общий вид: Nu \* Nt \* Nr \* 74 байт
* На 30 000 записей: 2.1 МБ
* На 30 000 записей чистый: 1.4 МБ

Таблица 2.2.9 — Общий итог

| Сущность | Объём (КБ) | Чистый объем (КБ) |
| --- | --- | --- |
| Пользователи | 101 | 94 |
| Группы | 135 | 88 |
| Задачи | 10035 | 9871 |
| Заметки | 3584 | 3174 |
| Ссылки | 1576 | 1433 |
| Теги | 111 | 88 |
| История изменений | 8028 | 7557 |
| Ребра | 2150 | 1433 |

* Общий размер: ~24.4 МБ
* Общий размер чистый: ~23.2 МБ

Выразим объем данных через количество пользователей в системе по уже описанным средним значениям: V(Nu) = 25782 \* Nu байт

**С. Избыточность данных**

Для вычисления чистых данных были убраны размеры идентификаторов сущностей и идентификаторов для связи с другими группами

* Общий размер: ~24.4 МБ
* Общий размер чистый: ~23.2 МБ

Тогда избыточность данных равна: R(Nu) = 24.4 / 23.2 = 1.05

**D. Направление роста модели при увеличении количества объектов каждой сущности.**

При увеличении количества объектов каждая модель сущности будет расти линейно.

**F. Примеры данных**

Таблица 2.2.10 — Таблица user

| id | Почта | Пароль | Дата создания | Дата изменения |
| --- | --- | --- | --- | --- |
| 1 | ivan@example.com | hash123 | 2025-04-10 10:00:00 | 2025-04-10 10:00:00 |
| 2 | olga@example.com | hash456 | 2025-04-11 12:15:00 | 2025-04-11 12:20:00 |

Таблица 2.2.11 — Таблица group

| id | Название | id пользователя |
| --- | --- | --- |
| 1 | Frontend | 1 |
| 2 | Backend | 1 |

Таблица 2.2.12 — Таблица task

| **id** | **Название** | **Содержание** | **Дата создания** | **Дата дедлайна** | **Дата изменения** | **Приоритет** | **Состояние выполнения** | **id группы** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Сделать API | Реализовать REST | 2025-04-10 11:00:00 | 2025-04-12 18:00:00 | 2025-04-11 09:00:00 | Высокий | В процессе | 2 |
| 2 | Верстка формы | Форма логина | 2025-04-10 11:30:00 | 2025-04-13 12:00:00 | 2025-04-11 08:00:00 | Средний | Не начато | 1 |

Таблица 2.2.13 — Таблица tag

| id | Название |
| --- | --- |
| 1 | urgent |
| 2 | frontend |

Таблица 2.2.14 — Таблица task\_tag

| id\_task | id \_tag |
| --- | --- |
| 1 | 2 |
| 2 | 2 |

Таблица 2.2.15 — Таблица note

| id | Описание | Дата создания | id task |
| --- | --- | --- | --- |
| 1 | Добавить валидацию | 2025-04-10 12:00:00 | 1 |
| 2 | Перепроверить стили | 2025-04-11 10:00:00 | 2 |

Таблица 2.2.16 — Таблица attachment

| id | id task | Текст | Ссылка |
| --- | --- | --- | --- |
| 1 | 1 | Документация | <https://docs.example.com/api> |
| 2 | 2 | Макет Figma | <https://figma.com/mockup> |

Таблица 2.2.17 — Таблица edge

| id | Название | id task from | id task to |
| --- | --- | --- | --- |
| 1 | Зависимость | 1 | 2 |

**G. Примеры запросов**

**Пользователь регистрируется**

* Существует ли уже такой email

SELECT id FROM user WHERE email = 'helloworld@gmail.com';

* Создаем нового пользователя

INSERT INTO user (id, email, password, createdAt, modifiedAt)

VALUES (

UUID(),

'helloworld@gmail.com',

'veryhardhashpassword',

NOW(),

NOW()

);

Количество запросов: 2

Задействованные сущности: user

**Пользователь логинится**

* Существует ли уже такой email

SELECT id, email FROM user

WHERE email = 'some@gmail.com' AND password = 'veryhardhashpassword';

Количество запросов: 1

Задействованные сущности: user

**Пользователь фильтрует задачи по статусу**

SELECT t.\*

FROM task t

JOIN group g ON t.group\_id = g.id

JOIN user u ON g.user\_id = u.id

WHERE u.id = '123e4567-e89b-12d3-a456-426655440000'

AND t.status = 'В процессе';

Количество запросов: 2

Количество JOIN: 2

Задействованные сущности: task, group, user

**Пользователь добавляет новую группу**

INSERT INTO Group (id, name, userId)

VALUES (

UUID(),

'Meal tasks',

'123e4567-e89b-12d3-a456-426655440000'

);

Количество запросов: 1

Задействованные сущности: group

**Пользователь добавляет новое задание в некоторой группе "some group"**

INSERT INTO Task (id, title, content, createdAt, change, deadline, status, priority, groupId)

VALUES (

UUID(),

'Поесть',

'Вкусно покушать',

NOW(),

NOW(),

NULL,

'В процессе',

'Важное',

(SELECT id FROM group WHERE Name = 'Meal task')

);

Количество запросов: 2

Задействованные сущности: group, task

**Пользователь создает новый тег**

INSERT INTO tag (id, name)

VALUES (UUID(), 'Итерация 1');

Количество запросов: 1

Задействованные сущности: tag

**Пользователь закрепляет некоторый тег к заданию**

INSERT INTO task\_tag (taskId, tagId)

VALUES (

(SELECT Id FROM task WHERE title = 'Поесть'),

(SELECT Id FROM tag WHERE name = 'Итерация 1')

);

Количество запросов: 3

Задействованные сущности: tag, task

**Пользователь прикрепляет ссылку к заданию**

INSERT INTO attachment (id, taskId, text, URL)

VALUES (

UUID(),

(SELECT id FROM task WHERE title = 'some task'),

'Яндекс браузер',

'https://ya.ru/?npr=1&utm\_referrer=https%3A%2F%2Fyandex.ru%2F'

);

Количество запросов: 2

Задействованные сущности: task, attachment

**Пользователь изменяет статус у задания**

* Изменяется значение у сущности задания

UPDATE task

SET status = 'Завершено', change = NOW()

WHERE title = 'Поесть';

Количество запросов: 1

Задействованные сущности: task

* Создается новая нода истории изменения и привязывается к ноде с типом

INSERT INTO changeHistory (id, taskId, modifiedAt, oldValue, newValue, fieldType)

VALUES (

UUID(),

(SELECT id FROM task WHERE title = 'Поесть'),

NOW(),

'В процессе',

'Завершено',

'Статус'

);

Количество запросов: 2

Задействованные сущности: task, changeHistory

**Пользователь хочет получить задачи, связанные с текущей**

SELECT t.\*

FROM task t

JOIN edge e ON (t.id = e.taskToId OR t.id = e.taskFromId)

WHERE (e.taskFromId = (SELECT id FROM task WHERE title = 'Поесть')

OR e.taskToId = (SELECT id FROM task WHERE title = 'Поесть'))

AND t.id != (SELECT id FROM task WHERE title = 'Поесть');

Количество запросов: 4

Задействованные сущности: task, edge

**Пользователь ищет путь от одной задачи до другой**

WITH RECURSIVE task\_path AS (

SELECT

t.id AS current\_task\_id,

t.title AS current\_name,

ARRAY[t.id] AS path\_ids

FROM task t

WHERE t.title = 'test\_task\_a'

UNION ALL

SELECT

t\_next.id AS current\_task\_id,

t\_next.title,

tp.path\_ids || t\_next.id

FROM task\_path tp

JOIN edge r ON r.id\_task\_from = tp.current\_task\_id

JOIN task t\_next ON t\_next.id = r.id\_task\_to

WHERE NOT t\_next.id = ANY(tp.path\_ids)

)

SELECT tp.path\_ids,

ARRAY\_AGG(t.title ORDER BY array\_position(tp.path\_ids, t.id)) AS path\_names

FROM task\_path tp

JOIN task t ON t.id = ANY(tp.path\_ids)

WHERE tp.current\_name = 'test\_task\_b'

GROUP BY tp.path\_ids;

Количество запросов: N

Задействованные сущности: task, edge

**Пользователь ищет топ-10 незавершенных задач**

SELECT \*

FROM Task t

JOIN User u ON t.userId = u.id

WHERE u.mail = 'test@mail.com'

AND t.status <> 'Завершена'

ORDER BY t.createdAt

LIMIT 10;

Количество запросов: 2

Задействованные сущности: task, user

**2.3 Сравнение моделей**

Таблица 2.3.1 — Удельный объём информации

| Параметр | Нереляционная модель (грязные) | Нереляционная модель (чистые) |
| --- | --- | --- |
| Грязные | V(Nu)=25782⋅Nu байт | V(Nu)=31639⋅Nu байт |
| Чистые | V(Nu)=24554⋅Nu байт | V(Nu)=23436⋅Nu байт |
| Избыточность | 1.05 | 1.35 |

Таблица 2.3.2 — Удельный объём информации

| № | Название запроса | Реляционная модель (Запросы) | Нереляционная, коллекции | Графовая модель (Запросы) | Реляционная, коллекции |
| --- | --- | --- | --- | --- | --- |
| 1 | Пользователь регистрируется | 2 | user | 2 | user |
| 2 | Пользователь логинится | 1 | user | 1 | user |
| 3 | Пользователь фильтрует задачи по статусу | 3 | task, group, user | 1 | task |
| 4 | Пользователь добавляет новую группу | 1 | group | 2 | group, user |
| 5 | Пользователь добавляет новое задание в группе | 2 | group, task | 2 | group, task |
| 6 | Пользователь создает новый тег | 1 | tag | 1 | tag |
| 7 | Пользователь закрепляет тег к заданию | 3 | tag, task | 3 | tag, task |
| 8 | Пользователь прикрепляет ссылку к заданию | 2 | task, attachment | 2 | task, attachment |
| 9 | Пользователь изменяет статус у задания | 1 | task | 1 | task |
| 10 | Создается новая нода истории изменения задания | 2 | task, changeHistory | 6 | task, history, fieldType |
| 11 | Пользователь хочет получить задачи, связанные с текущей | 4 | task, edges | 1 | task |
| 12 | Пользователь ищет путь от одной задачи до другой. | N | task, edges | N | task |
| 13 | Пользователь ищет топ-10 незавершенных задач. | 2 | task, user | 3 | task, user |

**Вывод**

По результатам исследования, можно сделать вывод, что нереляционная база данных Neo4j требует немного больше памяти для хранения данных по сравнению с реляционной базой данных. Однако этот недостаток компенсируется высокой скоростью доступа к связанным задачам. В реляционной модели для извлечения нужных данных потребуется больше времени из-за необходимости выполнения операций слияния таблиц (JOIN), что делает процесс более затратным по времени.

Избыточность в Neo4j превышает таковую в реляционной модели, что объясняется тем, что в графовой базе каждая связь между сущностями сопровождается дополнительными служебными данными в виде uuid, int и label.

# 3. РАЗРАБОТАННОЕ ПРИЛОЖЕНИЕ

## 3.1. Краткое описание

Приложение представляет собой современное веб-решение, построенное по классической архитектуре «клиент — сервер» и включает в себя следующие основные компоненты:

1. Frontend (клиентская часть) — реализован с использованием фреймворка React.js, который обеспечивает гибкое и удобное создание современных пользовательских интерфейсов. Для визуализации графов применялась библиотека react-flow, позволяющая наглядно отображать связи между задачами.
2. Backend (серверная часть) — построен на фреймворке Django 5, с помощью которого реализована вся основная серверная логика приложения. Для взаимодействия с графовой базой данных использовались Neo4j Python Driver и neomodel.
3. База данных — в качестве единственного хранилища данных использовалась графовая СУБД Neo4j, что позволило эффективно моделировать и анализировать взаимосвязи между задачами.

## 3.2. Использованные технологии

Frontend:

* JavaScript
* Современные веб-технологии (HTML5, CSS3)
* [React.js](http://react.js)
* React-flow

Backend:

* Python
* Django 5
* REST API
* Neo4j Python Driver
* neomodel

Инфраструктура:

* Docker
* Docker Compose
* Neo4j (графовая база данных)
* Git (система контроля версий)

## 3.3. Снимки экрана приложения

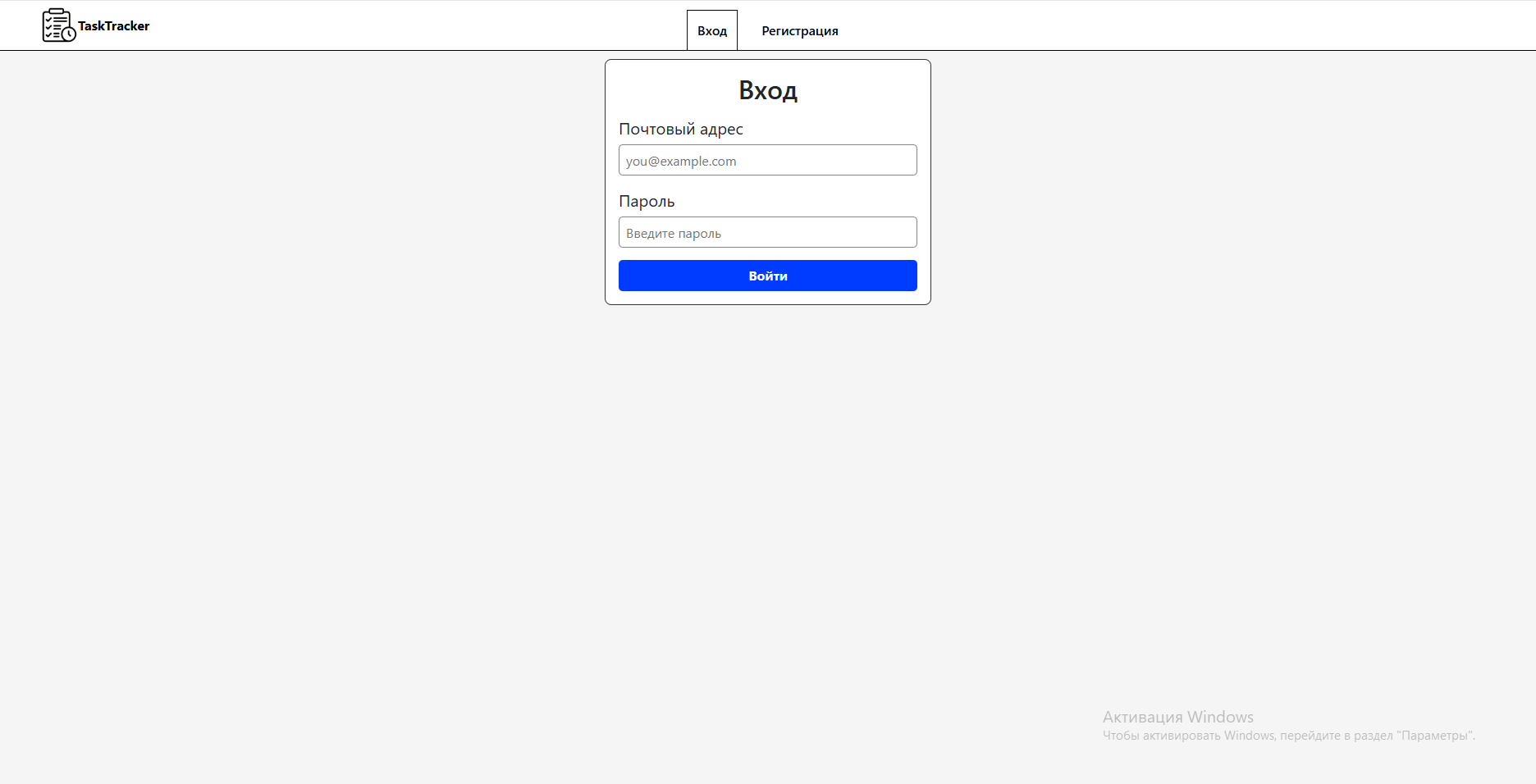


Рисунок 3.3.1 — Страница входа

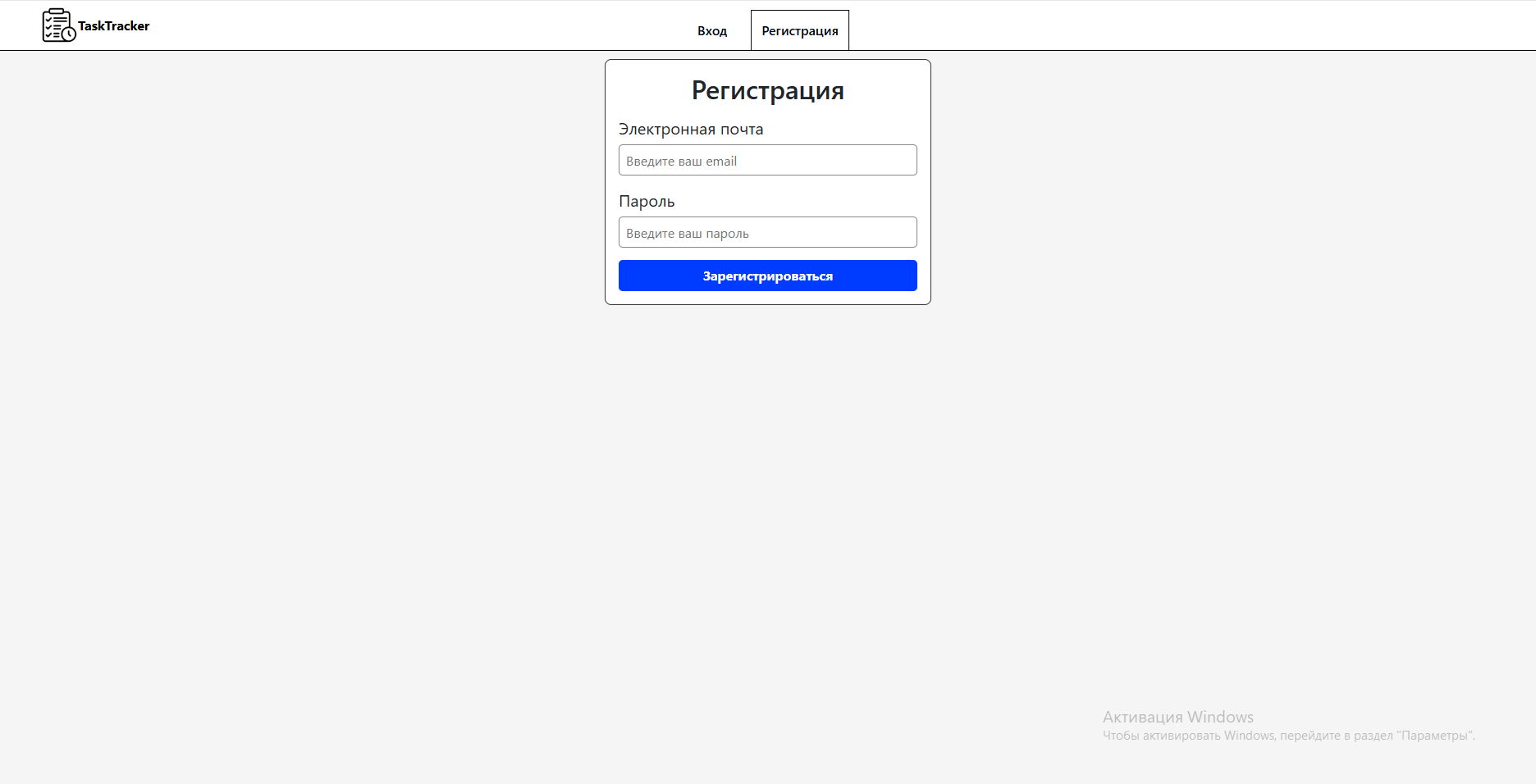


Рисунок 3.3.2 — Страница регистрации

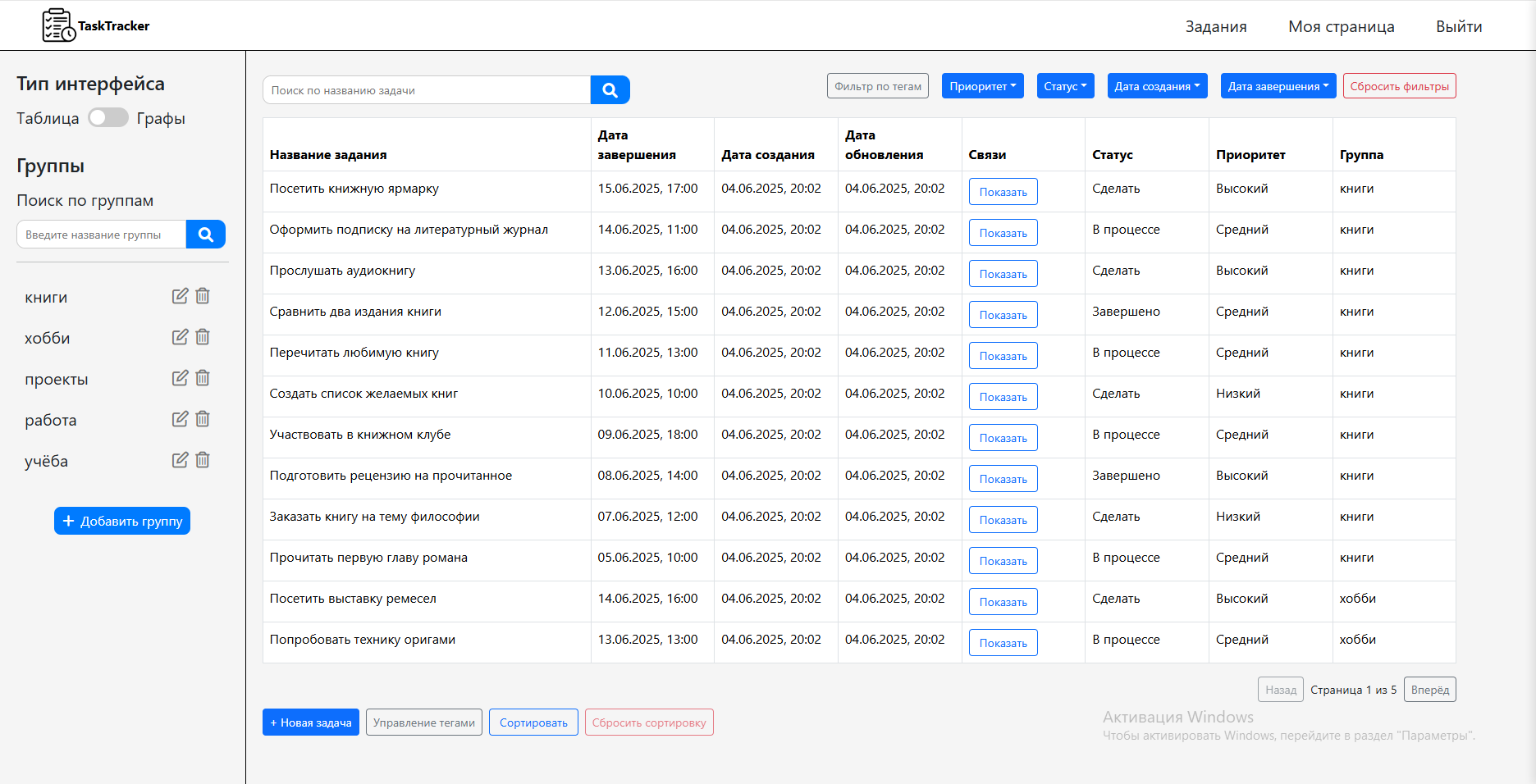


Рисунок 3.3.3 — Страница всех тасок

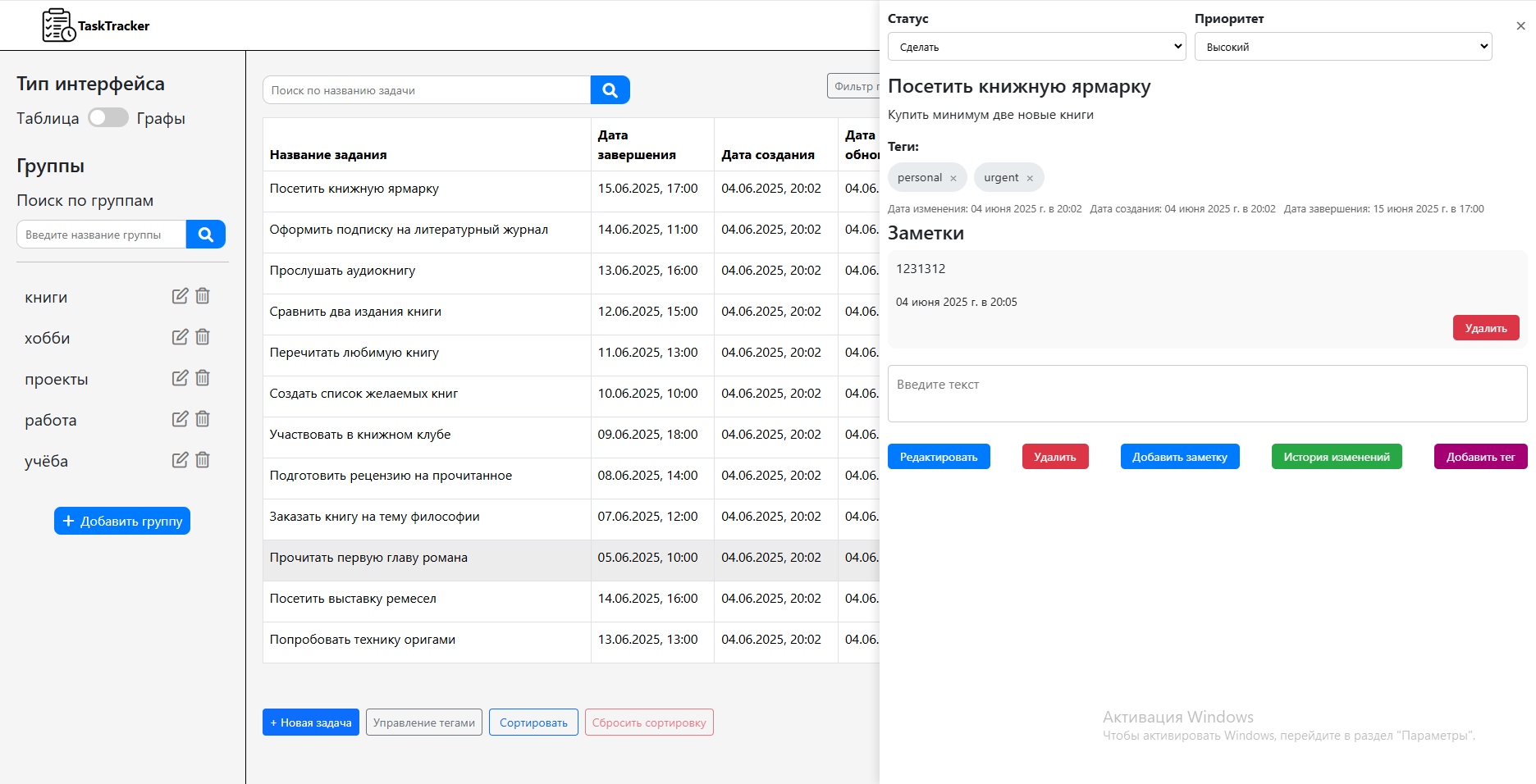


Рисунок 3.3.4 — Детали таски

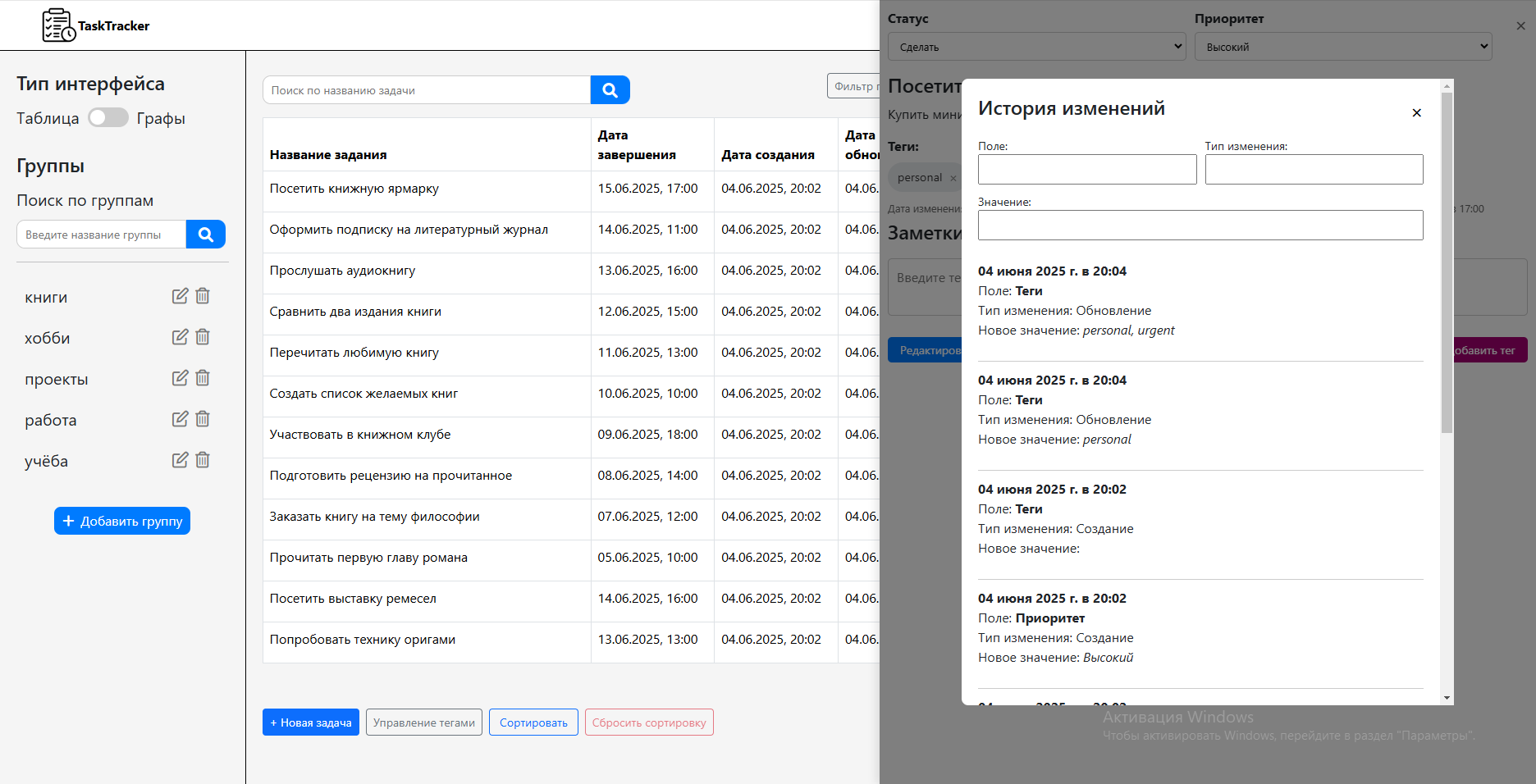


Рисунок 3.3.5 — История изменений таски

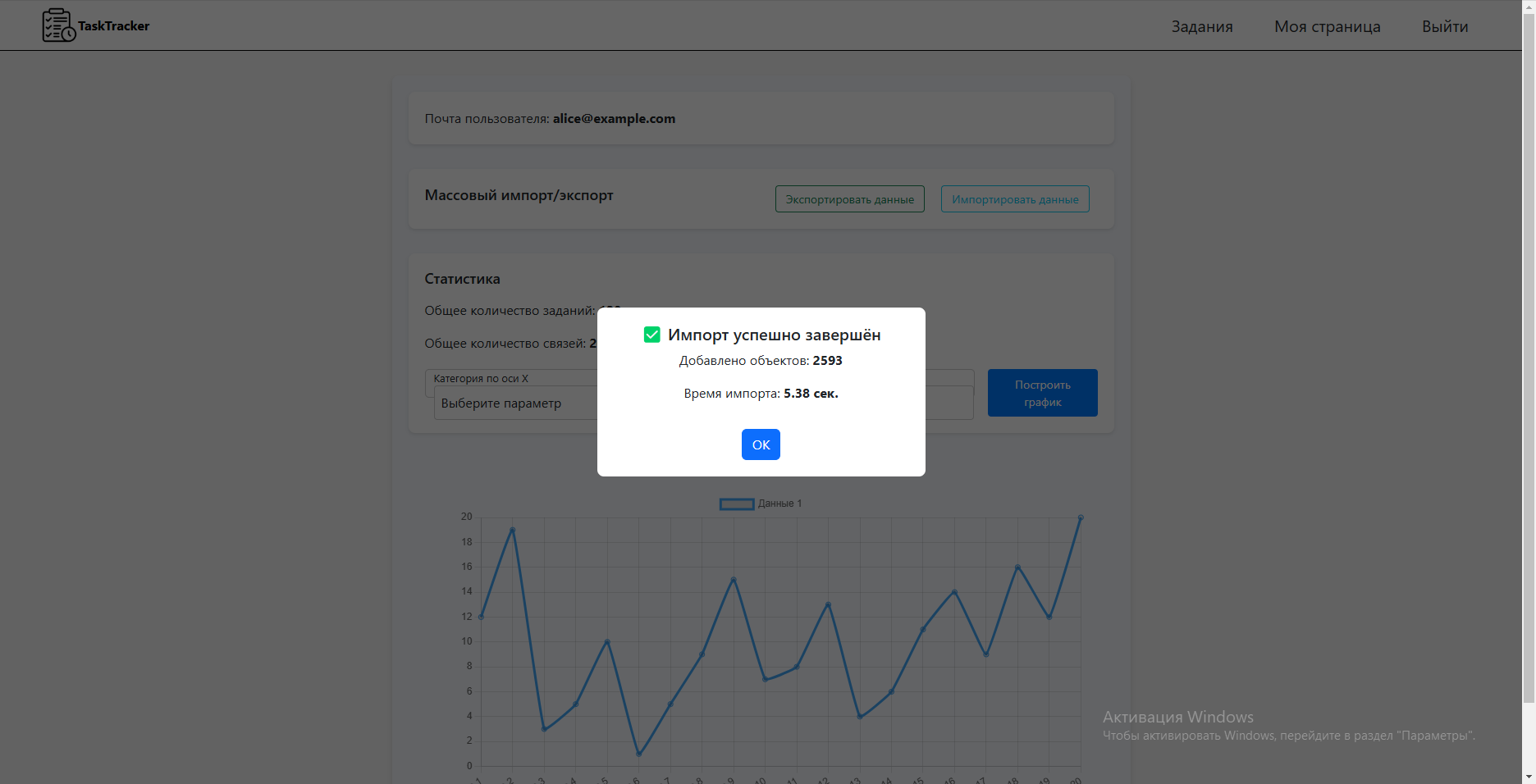


Рисунок 3.3.6 — Успешный импорт данных

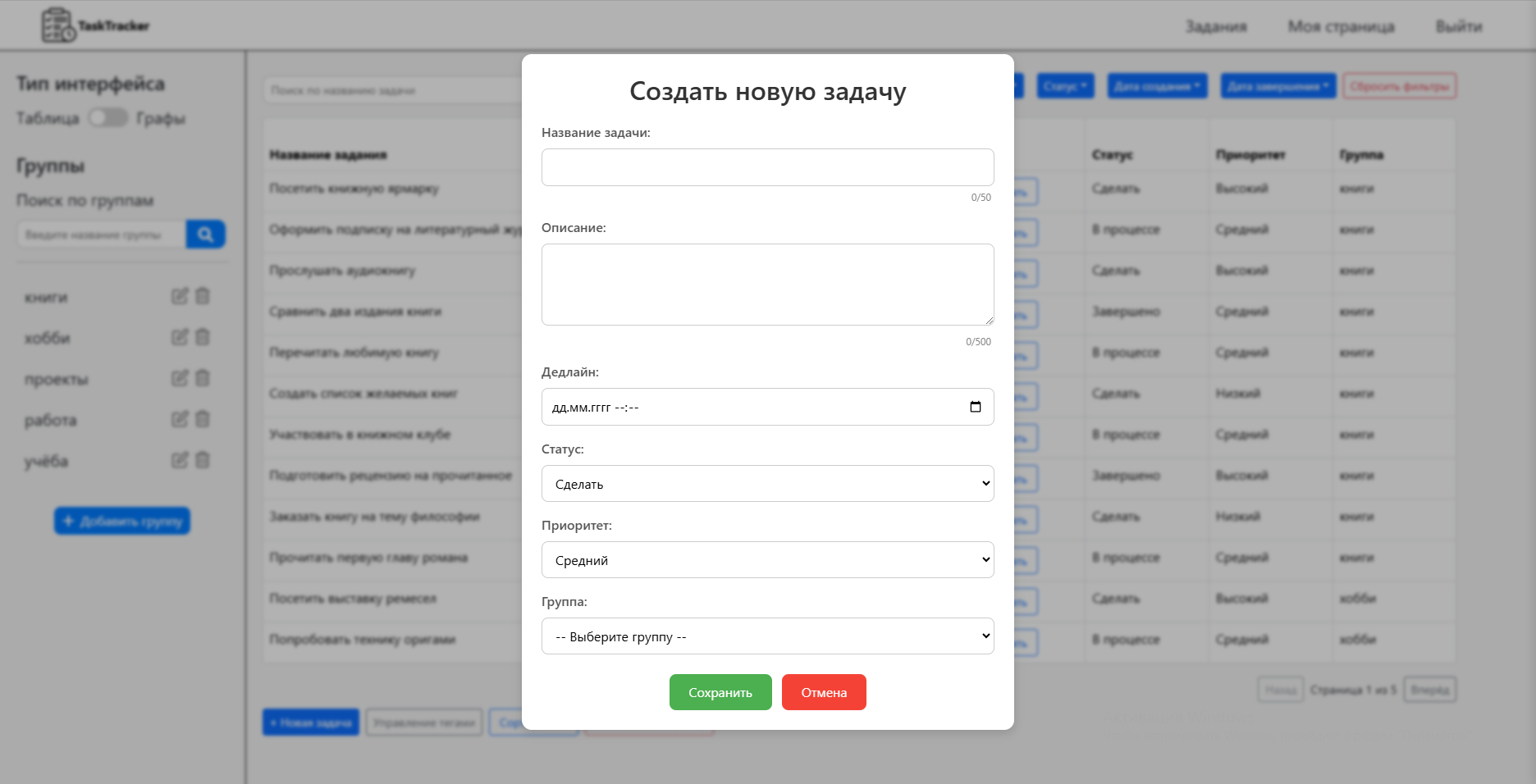


Рисунок 3.3.7 — Создание таски

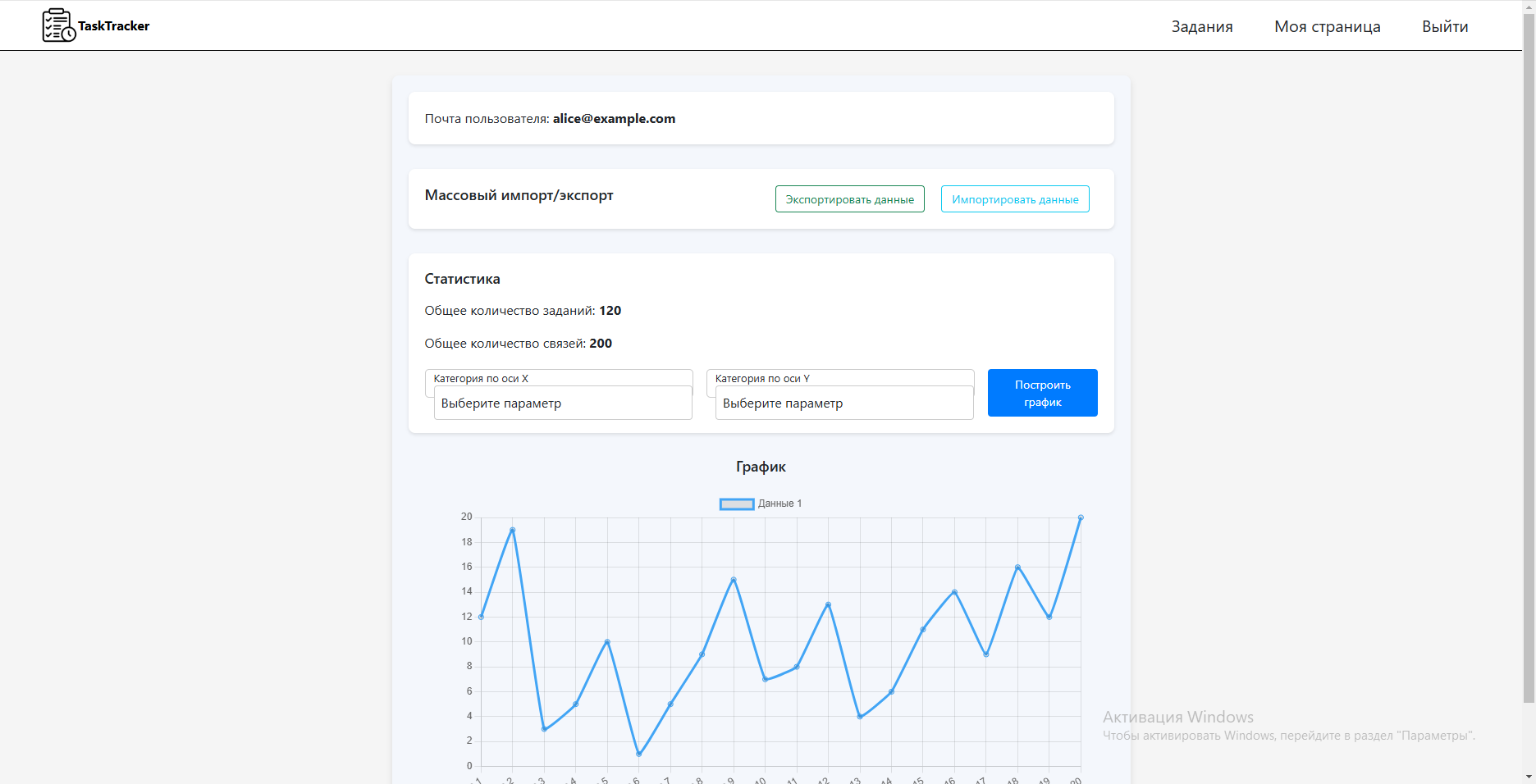


Рисунок 3.3.8 — Страница пользователя

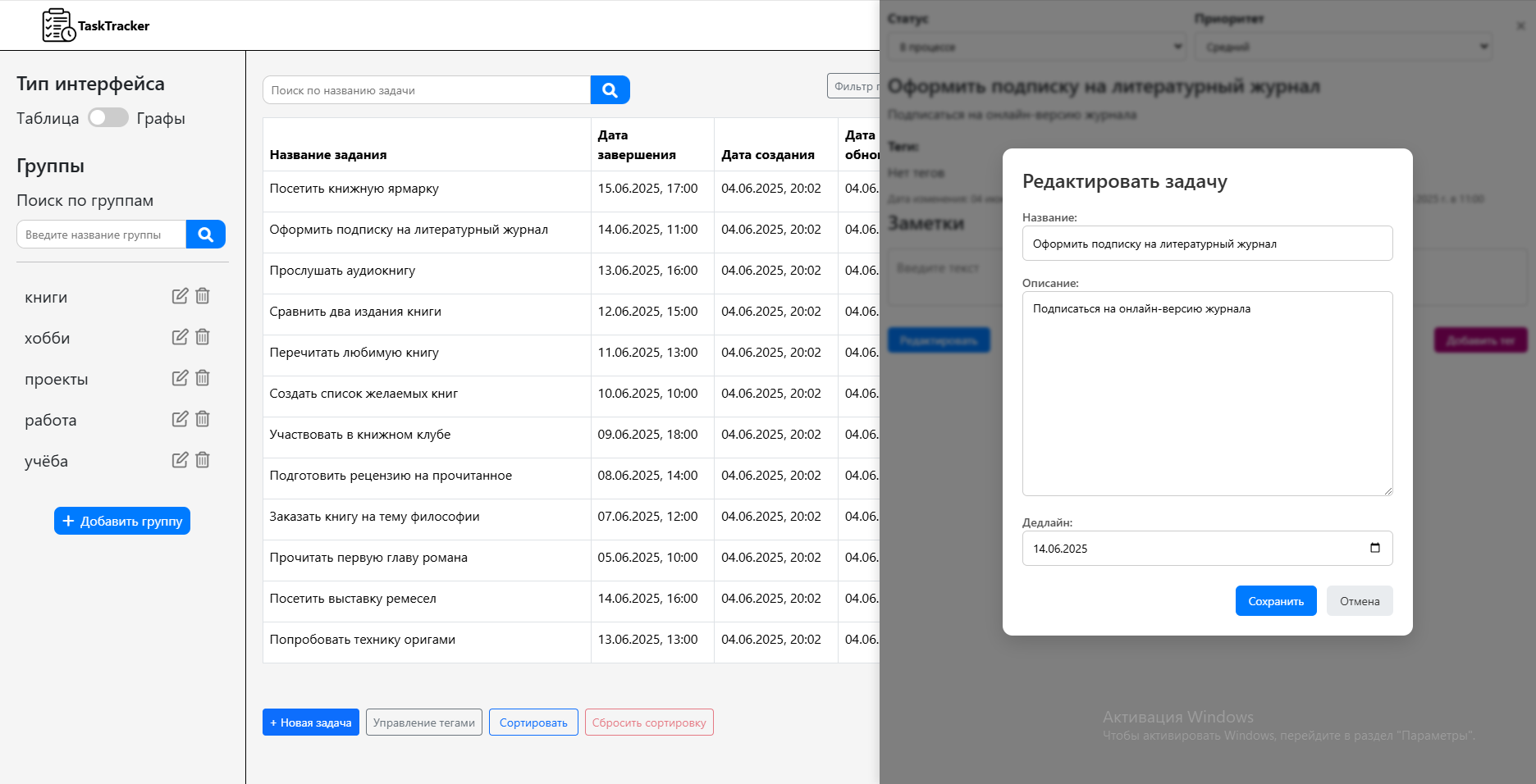


Рисунок 3.3.9 — Редактирование таски

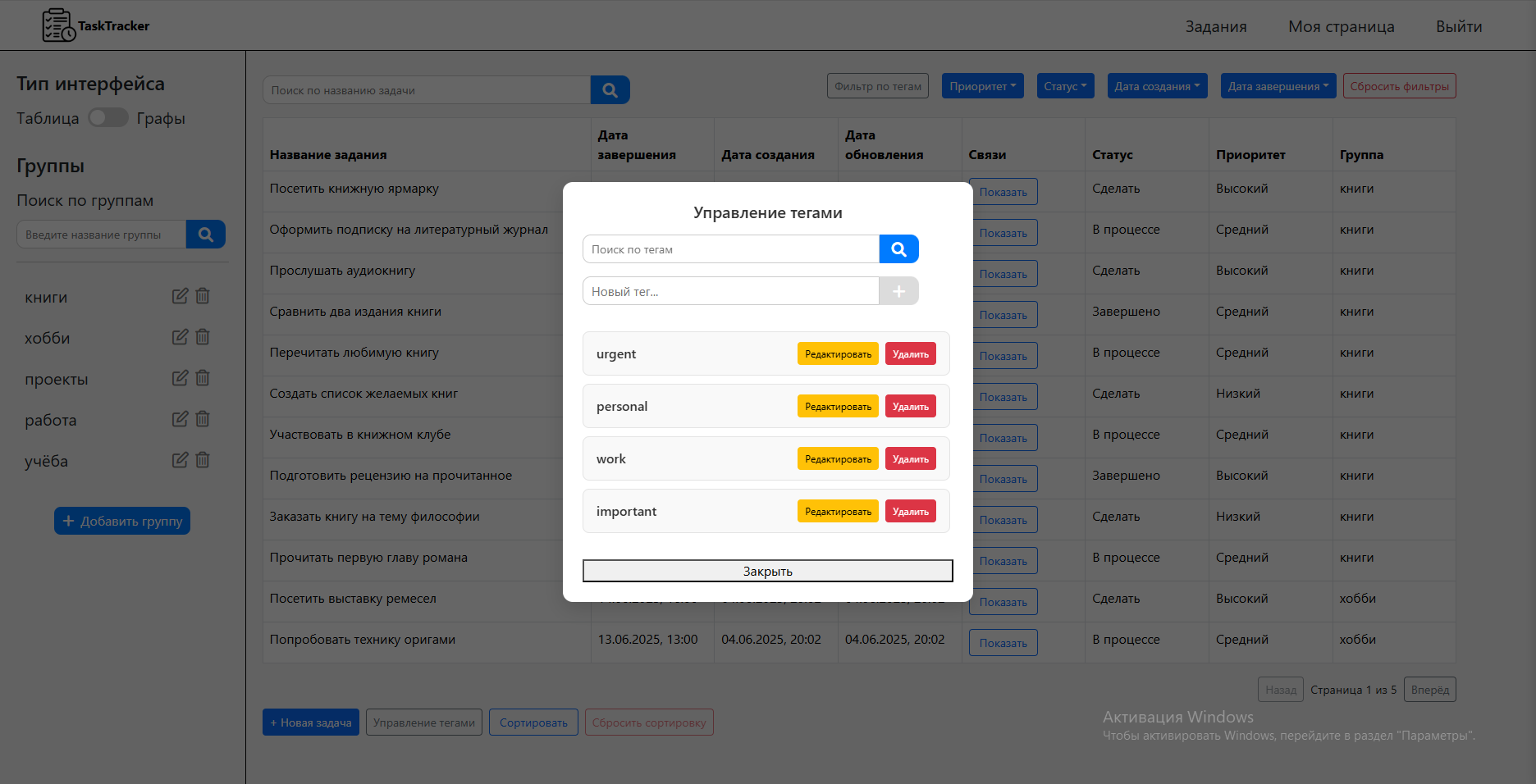


Рисунок 3.3.10 — Управление тегами

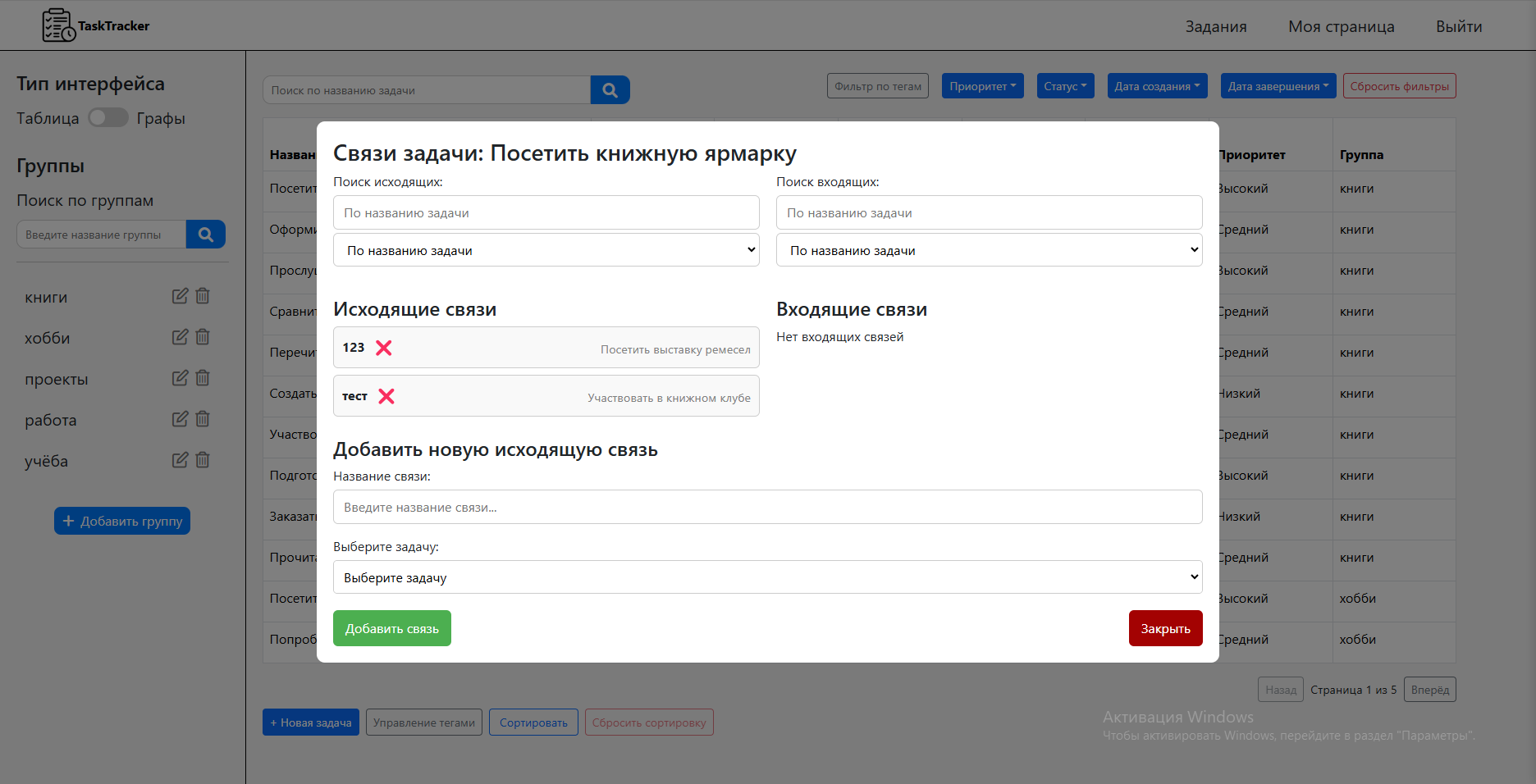
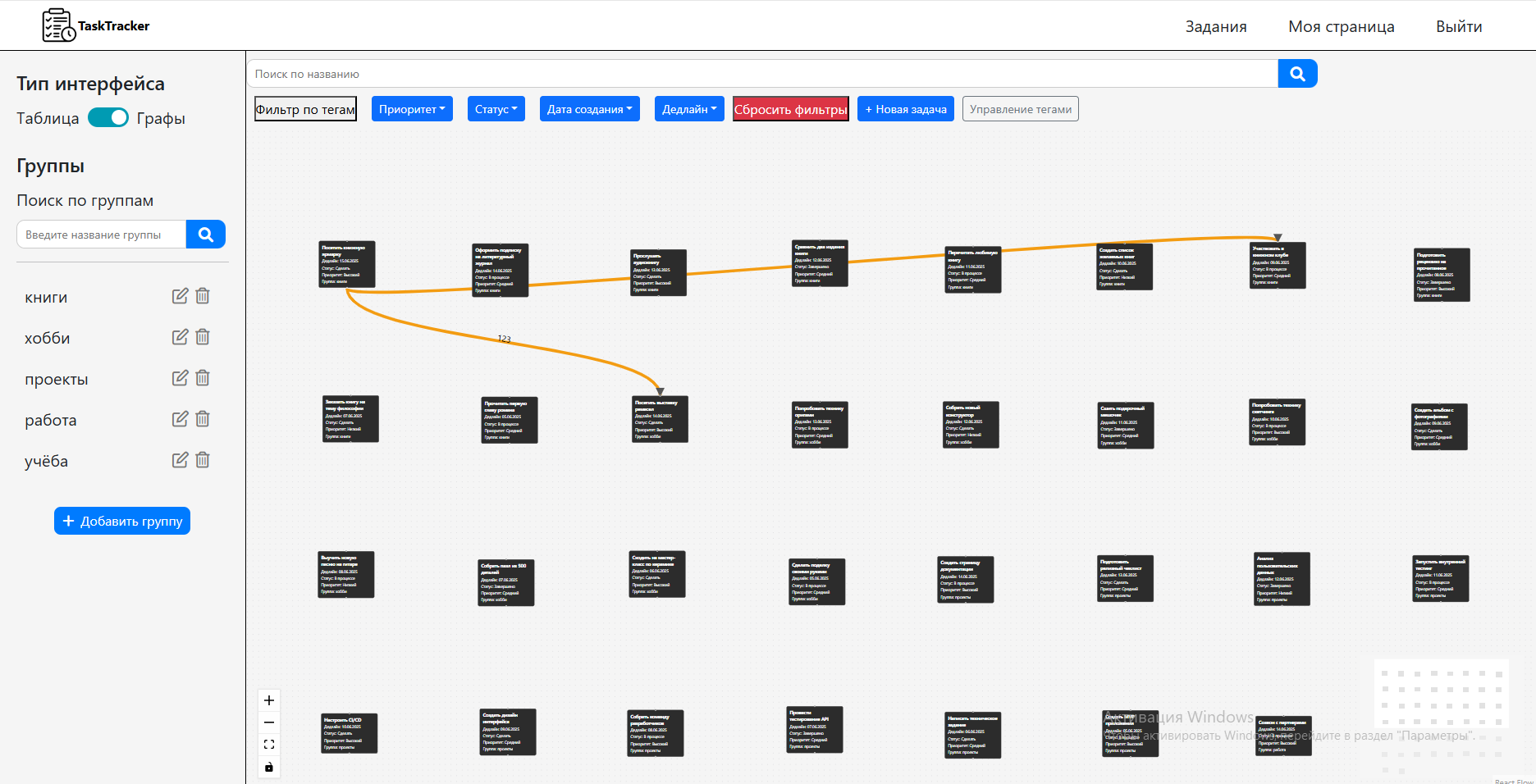
Рисунок 3.3.11 — Создание связей между тасками

Рисунок 3.3.12 — Графовое представление тасок

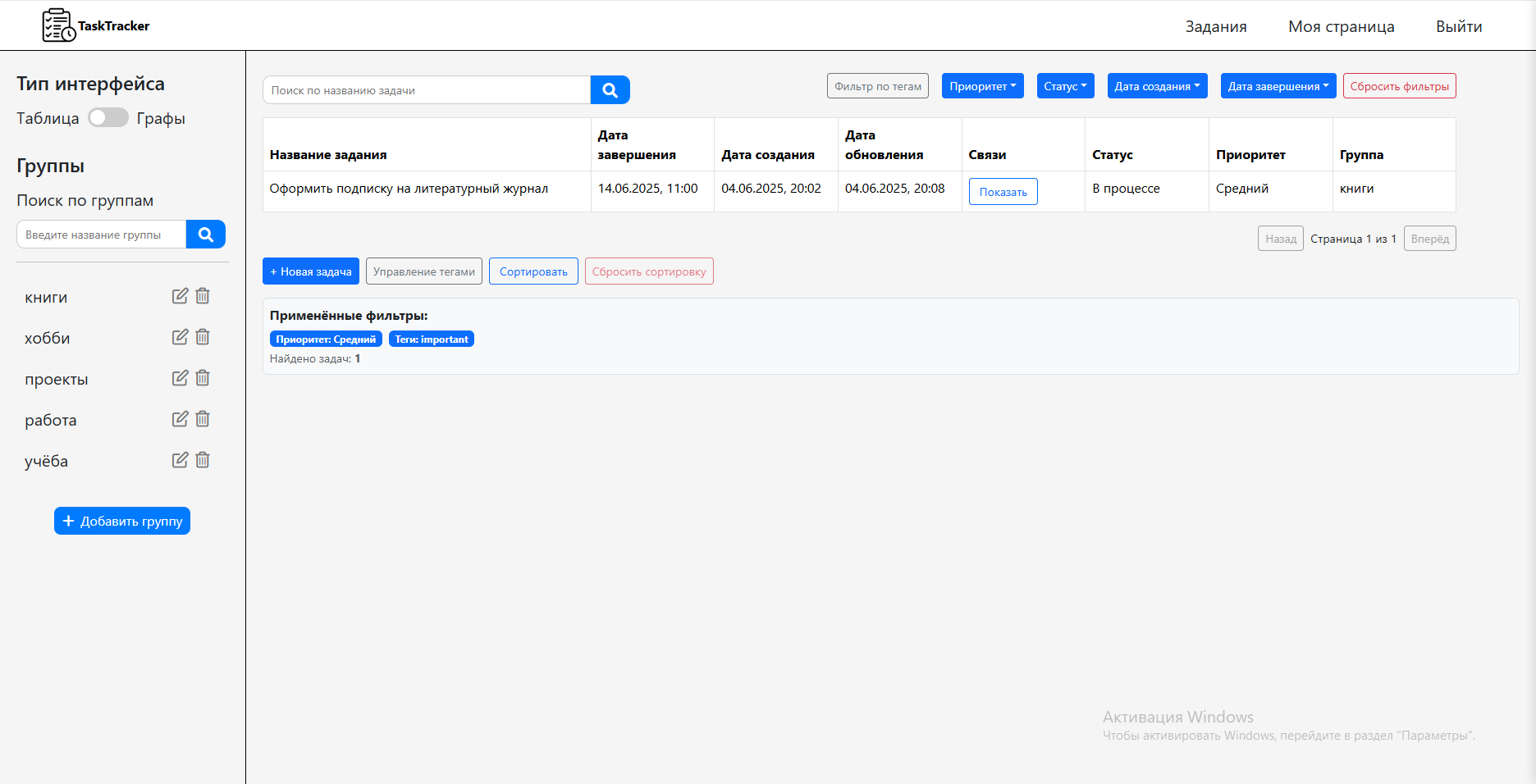


Рисунок 3.3.13 — Фильтрация тасок

# ВЫВОДЫ

## 4.1. Достигнутые результаты

В рамках проекта было реализовано полнофункциональное веб-приложение для управления и визуализации пользовательских задач. Основные достижения включают:

* Разработка полноценного API для работы с задачами, хранящимися в графовой базе данных Neo4j.
* Реализация удобной системы поиска и фильтрации данных практически для всех сущностей проекта.
* Визуализация задач в виде графа с отображением связей между ними, что упрощает восприятие структуры проекта.
* Обеспечение высокой производительности при работе с большим объёмом данных за счёт эффективного использования графовой модели хранения.

## 4.2. Недостатки и пути для улучшения полученного решения

* Возможность прикрепления ссылок и файлов к задачам в виде отдельных сущностей.
* Поддержка кастомизации пользовательской статистики для анализа активности и прогресса.
* Улучшенная система кэширования для повышения производительности при больших объёмах данных.
* Система уведомлений и обратной связи, включая оповещения о приближающихся дедлайнах.
* Сохранение положения узлов на графе для индивидуальной настройки визуализации.
* Возможность совместного управления задачами несколькими пользователями в рамках группы.

## 4.3. Будущее развитие решения

* Подключение OAuth
* Интеграция с календарем и телеграмом
* Добавление многоязычности
* Внедрение AI помощника
* Реализацию роли администраторов и разграничения прав доступа

# ЗАКЛЮЧЕНИЕ

В ходе разработки было создано полнофункциональное веб-приложение для формирования, отслеживания и управления пользовательскими задачами. Система поддерживает создание, редактирование и кастомизацию задач (добавление тегов, комментариев), а также предоставляет гибкую фильтрацию по различным критериям: наличие тегов, дата создания, дата завершения и принадлежность к группам.

Ключевой особенностью проекта стала визуализация задач в виде графа с отображением их связей. Использование графовой базы данных Neo4j позволило эффективно обрабатывать сложные зависимости между сущностями и обеспечивать высокую производительность.

Интуитивно понятный пользовательский интерфейс, разработанный с использованием React.js, обеспечивает лёгкий вход в работу с системой без необходимости изучения документации. Современные архитектурные решения делают проект гибким, масштабируемым и пригодным для дальнейшего сопровождения и развития другими разработчиками.

# ПРИЛОЖЕНИЯ

1. **Документация по сборке и развертыванию приложения**

1. Подготовка к установке

* Клонирование репозитория:
  + git clone <https://github.com/moevm/nosql1h25-tasktrack.git>
  + cd nosql1h25-tasktrack
* Настройка переменных окружения: отредактируйте файл .env в соответствии с вашими требованиями или скопируйте уже существующий файл .env-example в .env.

3. Сборка и запуск

* Сборка контейнеров:
  + docker-compose build --no-cache -d
* Запуск приложения:
  + docker-compose up
* Иной вариант: одной командой:
  + docker-compose up -d -–build

4. Доступ к сервисам

* Все приложение работает по адресу: <http://localhost:9000>
* Для доступа к ручкам серверной части: <http://localhost:9000/api/>

5. Остановка приложения

* Остановка контейнеров:
  + docker-compose down
* Остановка и удаление контейнеров (включая данные БД)
  + docker-compose down -v

1. **Инструкция для пользователя**

1. Начало работы

* Откройте приложение в браузере по адресу <http://localhost:9000>
* На главной странице пользователь может зарегистрировать, после чего автоматически произойдёт вход в систему.
* Или же пользователь может залогиниться по почте и паролю.

2. Основные функции

1. Просмотр тасок

* Используйте поисковую строку для быстрого поиска по названию таски
* Применяйте фильтры для уточнения результатов
* Просматривайте детальную информацию о выбранном объекте
* Используйте группы для логического разделения тасок
* Пользуйтесь сортировками для упорядоченности по определенным признакам

1. Детали тасок

* Создавайте новые таски
* Редактируйте старые таски
* Устанавливайте теги для тасок
* Создавайте зависимости между тасками
* Оставляйте комментарии для тасок
* Просматривайте историю изменений тасок
* Удаляйте таски

1. Работа с графом связей

* Переключайтесь на графовое представление
* Изучайте связи визуально, а не в виде списка
* Фильтрация, создание и остальные детали тасок так же доступны в этом режиме
* Для создания связи в режиме графа, используйте зажатый левый контрл, затем следуйте подсказкам в правом нижнем углу
* Для удаления связи зажмите левый альт и следуйте подсказкам (Иногда он залипает, понажимайте его пару раз, пока не будет нормально высвечена подсказка).

# ЛИТЕРАТУРА

Официальная документация:

1. React.js – <https://react.dev/learn>
2. React-flow – <https://reactflow.dev/api-reference>
3. Django – <https://docs.djangoproject.com/en/5.2/>
4. neomodel – <https://neomodel.readthedocs.io/en/latest/>
5. Neo4j – <https://neo4j.com/docs/>
6. Docker – <https://docs.docker.com/>

Ссылка на репозиторий с проектом – [https://github.com/moevm/nosql1h25-tasktrack](https://github.com/moevm/nosql1h25-tasktrack#)