



# **КУРСОВОЙ ПРОЕКТ**

**Тема: Разработка программного модуля  
«Система управления задачами и проектами».  
Специальность 09.02.07 Информационные системы и  
программирование**

**Выполнил студент(ка) группы 31ИС-21 \_\_\_\_\_ П.С Сивунов**

**Руководитель \_\_\_\_\_ В.Ю. Назаров**

**Москва 2023**



**УТВЕРЖДАЮ**  
**Зам. директора КМПО**  
**С.Ф. Гасанов**  
« \_\_\_\_\_ » \_\_\_\_\_ 2023 г.

## **ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

**по дисциплине: МДК.01.01 Разработка программных модулей**  
**Специальность 09.02.07 Информационные системы и**  
**программирование**  
**Студент(ка) группы 31ИС-21 Сивунов Пётр**  
**ТЕМА: Разработка программного модуля**  
**«Система управления задачами и проектами».**

**Дата выдачи задания « \_\_\_\_\_ » \_\_\_\_\_ 2023 г.**

**Срок сдачи работы « \_\_\_\_\_ » \_\_\_\_\_ 2023 г.**

**Москва 2023**

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«РОССИЙСКАЯ АКАДЕМИЯ НАРОДНОГО ХОЗЯЙСТВА И ГОСУДАРСТВЕННОЙ  
СЛУЖБЫ ПРИ ПРЕЗИДЕНТЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»  
КОЛЛЕДЖ МНОГОУРОВНЕВОГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

**Задание на  
курсовой проект**

Дисциплина: МДК.01.01 Разработка программных модулей

Тема: Разработка программного модуля

«Система управления задачами и проектами».

Специальность: 09.02.07 Информационные системы и программирование

Группа: 31ИС-21

ФИО студента Сивунов П.С.

ФИО руководителя Назаров В.Ю.

1. Проанализировать предметную область
2. Проанализировать готовые решения
3. Подготовить техническое задание
4. Обосновать выбор инструментов и средств разработки
5. Описать реализацию технического задания
6. Выполнить тестирование

Задание выдано «\_\_\_\_\_» \_\_\_\_\_ 2023 г.

Срок выполнения «\_\_\_\_\_» \_\_\_\_\_ 2023 г.

Сроки защиты \_\_\_\_\_

Преподаватель: \_\_\_\_\_

Задание получил: \_\_\_\_\_

## СОДЕРЖАНИЕ

	Стр.
ВВЕДЕНИЕ .....	5
1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ .....	7
1.1 Основные понятия и объекты.....	7
1.2 Обзор и анализ существующих программных решений .....	10
2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ .....	16
2.1 Требования к функциям, выполняемым системой .....	16
2.2 Описание среды разработки.....	19
2.3 Обоснование выбора инструментария по разработке .....	21
2.4 Модуль API .....	22
2.5 Структура приложения.....	22
3. РЕАЛИЗАЦИЯ ПРОЕКТА МОДУЛЯ СИСТЕМЫ.....	24
3.1 Описание кода модуля API.....	24
3.2 Описание десктопного приложения .....	25
3.3 Тестирование .....	32
ЗАКЛЮЧЕНИЕ .....	33
СПИСОК ИСТОЧНИКОВ .....	34
Приложение 1 .....	36
Приложение 2 .....	37
Приложение 3 .....	38
Приложение 4 .....	39

## ВВЕДЕНИЕ

Бизнес-процессы становятся все более сложными и конкурентоспособность компаний напрямую зависит от эффективного управления проектами, системы управления задачами и проектами становятся неотъемлемой частью организационной деятельности. С увеличением объема информации и разнообразия задач возникает потребность в интегрированных и интеллектуальных инструментах, способных эффективно координировать работу команд, управлять ресурсами и повышать общую производительность. В этом контексте возникает необходимость в создании "Системы управления задачами и проектами" — программного продукта, предназначенного для систематизации и автоматизации управленческих процессов.

Основной целью данного проекта является разработка модуля системы управления задачами и проектами. Модуль способен повысить эффективность работы команд, уменьшить временные и ресурсные затраты на управление проектами, а также обеспечить более прозрачную и обоснованную систему принятия управленческих решений.

Актуальность данного проекта обусловлена не только стремительным технологическим прогрессом, но и повышением требований к оперативности и точности управленческих решений. Сложность современных бизнес-процессов требует от организаций эффективных инструментов для планирования, контроля и анализа проектов. Управление задачами и проектами становится ключевым элементом успешного ведения бизнеса, влияя на конечные результаты и обеспечивая гибкость в условиях постоянных изменений рынка.

Для достижения поставленной цели – разработки модуля системы управления задачами и проектами – был выделен ряд ключевых задач:

- Анализ существующих подходов и систем управления проектами.
- Определение требований к системе.
- Разработка архитектуры и дизайна системы
- Реализация и тестирование

Объектом исследования являются современные бизнес-процессы и организационная деятельность компаний, осуществляющих управление проектами. В контексте данного проекта объектом также является создание программного продукта - модуля системы управления задачами и проектами.

Предметом исследования является процесс систематизации и автоматизации управленческих процессов. В рамках предмета исследования анализируются существующие подходы и системы управления проектами, определяются требования к создаваемому модулю, разрабатывается его архитектура и дизайн, а также проводится реализация и тестирование.

# 1. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Основные понятия и объекты

Система управления задачами и проектами (далее – Система) предназначена для автоматизации процессов управления задачами и проектами в организации.

Система должна обеспечивать следующие цели

1. Улучшение эффективности управления задачами и проектами
2. Повышение гибкости и адаптивности управления задачами и проектами
3. Сокращение затрат на управление задачами и проектами

Эффективность управления задачами и проектами можно повысить за счет следующих факторов:

- Повышение продуктивности менеджеров проектов. Система может автоматизировать такие задачи, как планирование, отслеживание хода выполнения и контроль задач и проектов. Это позволит освободить время менеджерам проектов для более стратегических задач.
- Повышение прозрачности и контроля за выполнением задач и проектов. Система обеспечивает прозрачность и контроль за выполнением задач и проектов. Это позволяет менеджерам проектов своевременно выявлять и устранять отклонения от плана.
- Сокращение времени и ресурсов, затрачиваемых на управление задачами и проектами. Система может помочь сократить время и ресурсы, затрачиваемые на управление задачами и проектами. Это связано с тем, что система автоматизирует рутинные задачи, повышает прозрачность и контроль, а также позволяет менеджерам проектов принимать более эффективные решения.

Гибкость и адаптивность управления задачами и проектами необходимы для своевременного реагирования на изменения в окружающей среде. Система обеспечивает гибкость и адаптивность управления задачами и проектами за счет следующих возможностей:

- Поддержка различных методологий управления проектами. Система должна поддерживать различные методологии управления проектами, что позволяет организациям выбирать наиболее подходящую методологию для своих задач.
- Возможность внесения изменений в план проекта. Система должна предоставлять возможность внесения изменений в план проекта в случае возникновения непредвиденных обстоятельств.
- Поддержка изменений в составе команды проекта. Система должна поддерживать изменения в составе команды проекта, что позволяет организациям быстро адаптироваться к изменениям в структуре организации.

Система может помочь сократить затраты на управление задачами и проектами за счет следующих факторов:

- Автоматизация рутинных задач. Система может автоматизировать такие задачи, как планирование, отслеживание хода выполнения и контроль задач и проектов. Это позволит сократить затраты на оплату труда менеджеров проектов.
- Использование шаблонов. Система позволяет использовать шаблоны для планирования и выполнения задач и проектов. Это позволит сократить затраты на подготовку документов.
- Интеграция с другими системами. Система может быть интегрирована с другими системами, используемыми в организации. Это позволит сократить затраты на ввод и обработку данных.

В предметной области Системы используются следующие основные понятия и объекты:

Задача — это единица работы, которая должна быть выполнена в рамках проекта или вне его. Задача может быть простой или сложной, а также может быть классифицирована по различным признакам, например, по типу, сложности, приоритету, сроку выполнения и т. д.



Проект – это совокупность взаимосвязанных задач, направленных на достижение определенной цели. Проект может быть различным по масштабу, сложности и длительности, а также может быть классифицирован по различным признакам, например, по типу, сложности, длительности, бюджету и т. Д.

Ресурс – это все, что может быть использовано для выполнения задач или проектов, например, люди, оборудование, материалы, деньги. Ресурс может быть материальным или нематериальным, а также может быть классифицирован по различным признакам, например, по типу, стоимости, доступности и т. Д.

Риск – это событие или условие, которое может оказать негативное влияние на достижение целей проекта. Риск может быть внутренним или внешним, а также может быть классифицирован по различным признакам, например, по вероятности наступления, тяжести последствий и т. Д.

Коммуникация – это обмен информацией между участниками проекта. Коммуникация может быть устной, письменной или электронной, а также может быть классифицирована по различным признакам, например, по типу, назначению, направлению и т. Д.

Качество – это степень соответствия продукта или услуги установленным требованиям. Качество может быть оценено с помощью различных показателей, например, надежности, функциональности, удобства использования и т. Д.

## 1.2 Обзор и анализ существующих программных решений

Существует множество программных продуктов, предназначенных для управления задачами и проектами, каждый из которых ориентирован на определенные потребности и особенности бизнес-процессов. Рассмотрим несколько ключевых программных решений, широко применяемых в предметной области:

## Jira

Очень популярная система, которая обычно используется для работы с кодом и как баг-трекер. Подходит для команд от 100 человек.

[illegible]

Рисунок 1 – Скриншот Jira

- Модули для разных команд.
- Работа с кодом в задачах. Благодаря интеграции с хостингом исходного кода Bitbucket программисты могут писать код и обсуждать его внутри задач.
- Модуль Confluence. Удобный способ вести документацию. Можно систематизировать информацию, например часто используемые формы и реестры, и контролировать версии.
- Создание задач прямо из чата техподдержки. Из-за этой функции систему часто используют как баг-трекер: проблемы легко зафиксировать и сразу начать решать.

- 16 видов отчётов. Диаграммы сгорания задач (чтобы видеть прогресс в достижении цели спринта) и скорости команды, отчёты по спринтам и по загруженности пользователей, контрольный график и так далее.
- Дорожные карты. Позволяют составить подробный план как внутри одной команды, так и для нескольких сразу.

Trello: Очень популярный планировщик и таск-трекер, который лучше всего подходит для творческих команд и небольших коллективов.

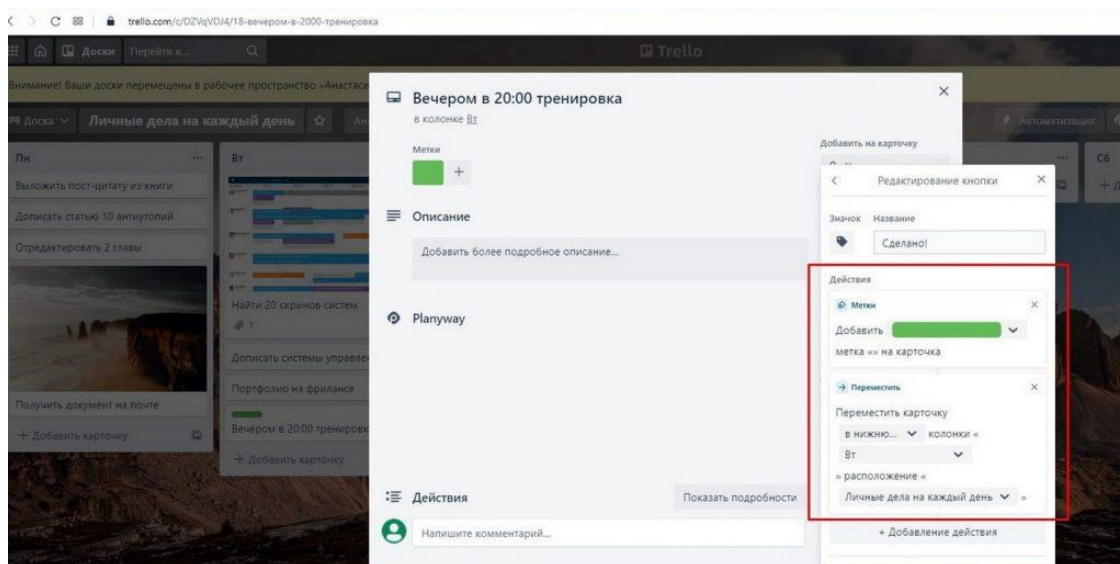


Рисунок 2 – Скриншот Trello

- Автоматизация. Можно настроить автоматические действия с карточками при определённых условиях.
- Шаблоны. Есть готовые варианты под разные отделы и цели: для личных дел, для команды маркетинга, для дизайнеров и т.п.
- Интеграция с почтой. У каждой карточки есть имейл-адрес, и оставлять комментарии можно прямо из электронного ящика без авторизации в системе.
- Хронология. На временной шкале отображаются все взаимосвязи между членами команды. Можно разбивать работу над проектом на этапы и контролировать каждый из них, чтобы равномерно распределять нагрузку в команде.

- Визуализация данных в «Панели». Ключевые показатели представлены в форме диаграмм и дашбордов. Благодаря этому руководители могут быстро оценить, как продвигаются задачи и насколько загружена команда. «Битрикс24»

Корпоративный портал для крупных команд. Система заточена под отделы продаж и маркетинга, чаще используется как система управления взаимоотношениями с клиентами.

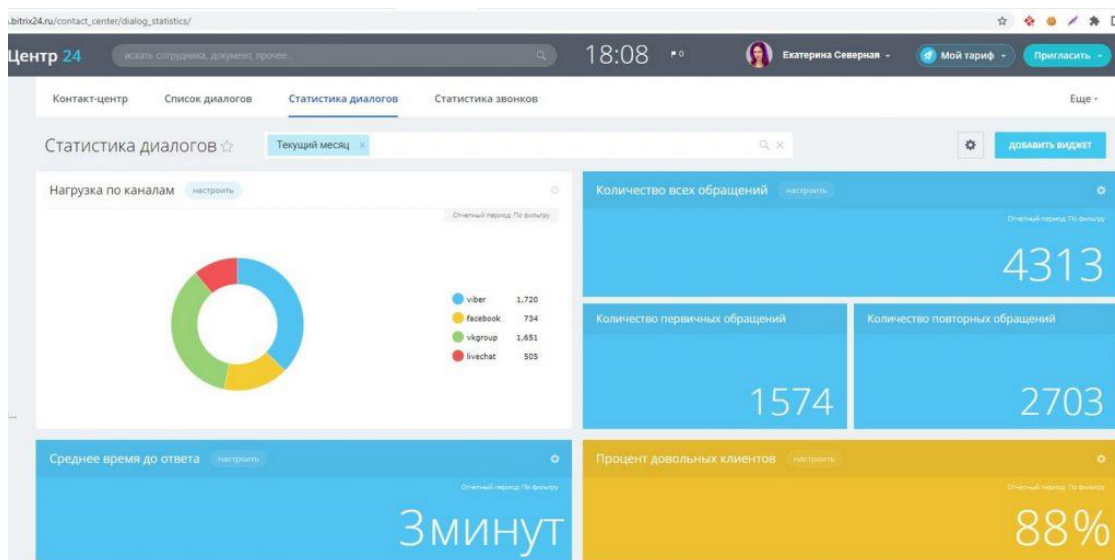


Рисунок 3 – Скриншот Битрикс24

- Здесь есть карточки, история, контроль сделок, телефония, записи разговоров, и т.д.
- Сквозная аналитика. Благодаря ей маркетологи прямо в «Битрикс24» могут рассчитывать ROMI, эффективность рекламы на разных площадках: от «Яндекс.Директа» и Google Ads до различных социальных сетей.
- Звонки и связь. Есть аудио и видеозвонки, групповые и личные чаты, отслеживание звонков, рассылки SMS-сообщений. Всё работает без сбоев, не хуже, чем в мессенджерах.
- Облачное хранилище. Файлы и документы легко редактировать прямо в системе.
- Группы. Их можно создавать по разным темам и направлениям работы, делать папки внутри, хранить в них файлы и вести общение.

- Контакт-центр. «Битрикс24» интегрирован с почтой, мессенджерами и социальными сетями. Тут есть телефония, онлайн-чат, а также статистика диалогов и звонков — с числом обращений и процентом довольных клиентов.

Asana: Система для планирования, операционного менеджмента и управления задачами. Лучше всего подойдёт небольшим командам: веб-студиям, агентствам и маркетинговым отделам.

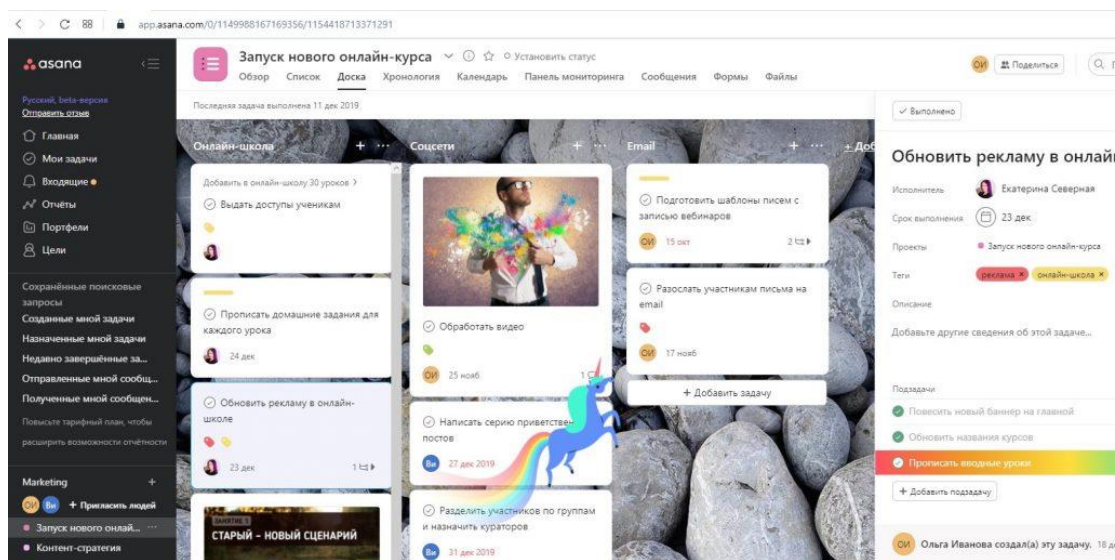


Рисунок 4 – Скриншот Asana

- Гибкая настройка задач. Можно пометить их любимыми тегами, выделять разными цветами, добавлять картинки на обложку карточки и делать список подзадач.
- Вехи. Они обозначают достижение промежуточного результата, по которому можно судить об успешности проекта в целом. Функция помогает визуально обозначать ключевые контрольные точки, чтобы лучше следить за происходящим.
- Работа на уровне проектов. Их можно копировать, экспортировать, делать из них шаблоны и синхронизировать задачи.
- Панель мониторинга. На ней в режиме реального времени можно видеть все важные данные по проектам.
- Цели. Их можно указать заранее и связать с рабочим процессом, чтобы потом отслеживать прогресс.

- **Файлы.** В отдельной вкладке можно посмотреть все документы, прикреплённые к проекту.

**Microsoft Project:** Microsoft Project представляет собой мощный инструмент для управления проектами, разработанный для организаций с разнообразными потребностями в управлении задачами и ресурсами.

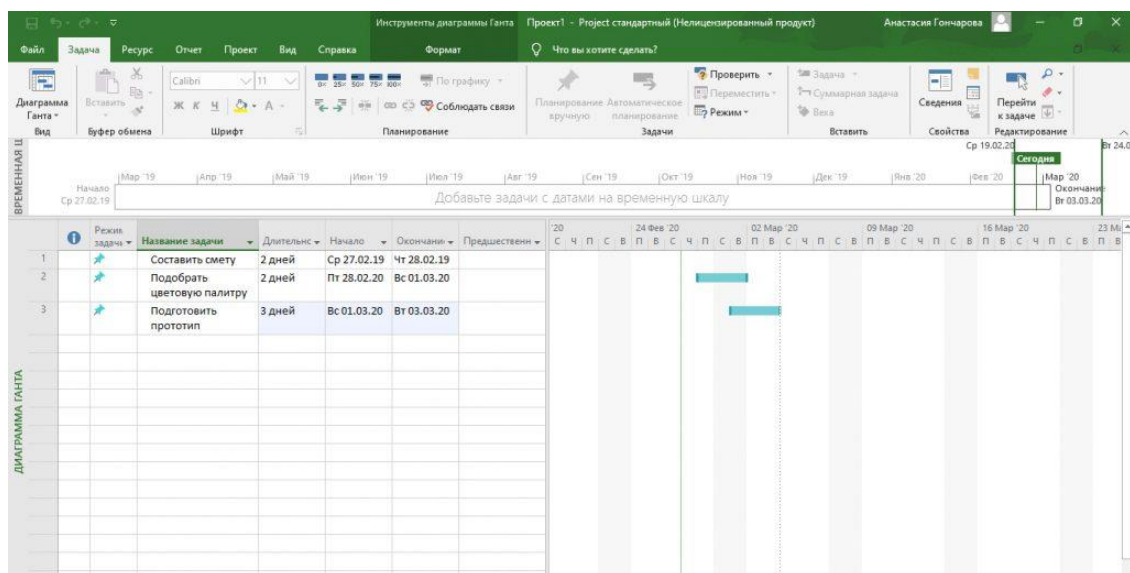


Рисунок 5 – Скриншот Microsoft Project

- **Гибкость планирования:** Пользователи могут определять задачи, устанавливать зависимости, назначать ресурсы и определять сроки выполнения.
- **Бюджетирование и анализ затрат:** Microsoft Project обеспечивает возможность составления бюджетов проектов, а также отслеживание и анализ затрат.
- **Отчетность**
- **Интеграция с другими приложениями Microsoft**
- **Поддержка различных методологий управления проектами**

Рынок программных продуктов для управления задачами и проектами предлагает разнообразные решения, ориентированные на различные потребности и особенности бизнес-процессов. Каждый из рассмотренных инструментов, таких как Jira, Trello, Битрикс24, Asana и Microsoft Project,

обладает своими уникальными функциональными возможностями, предназначенными для определенных типов команд и организаций.

Jira выделяется своей широкой функциональностью, особенно в области работы с кодом. Trello, напротив, ориентирован на простоту использования, визуализацию задач и автоматизацию процессов для творческих и небольших коллективов. Битрикс24 предлагает корпоративное решение с широким функционалом, включая систему управления клиентскими отношениями (CRM), облачное хранилище и контакт-центр. Asana предназначена для гибкого планирования задач и работы на уровне проектов, подходя небольшим командам и отделам. Microsoft Project, как мощный инструмент, предоставляет гибкие возможности планирования, бюджетирования и отчетности, а также интеграцию с другими приложениями Microsoft.

Эти преимущества делают их популярными и востребованными у пользователей.

## 2. РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

### 2.1 Требования к функциям, выполняемым системой

#### Функции компонента Авторизация

Окно авторизации должно открываться при запуске программы

Должна быть возможность ввести:

- Логин
- Пароль

Должна быть проверка на корректность введенных данных при нажатии на кнопку Войти.



Рисунок 6 – Набросок окна авторизации

#### Функции главного окна

Главное окно для администратора проекта должно обеспечить автоматизацию следующих функций платформы:

- Вывод задач по дате в текущем проекте
- Смена текущего проекта
- Просмотр описания задачи
- Добавление задачи себе, другим участникам проекта
- Редактирование задачи
- Удаление задачи



Главное окно для обычного пользователя должно обеспечить автоматизацию следующих функций платформы:

- Вывод задач по дате в текущем проекте
- Смена текущего проекта
- Просмотр описания задачи
- Изменение статуса задачи

#### Профиль

Личный профиль пользователя должен быть доступен всем пользователям платформы после авторизации.

Личный профиль пользователя должен содержать блок общих сведений, заполняемых при регистрации пользователя:

- ФИО пользователя;
- Логин
- Пароль

Пользователь должен иметь возможность редактирования общих сведений, а также сведений, необходимых для авторизации пользователя.

#### Уведомление о задаче

В Платформе должны быть реализованы следующие функции в части уведомления пользователя о поступившей задаче:

- Получение заявки исполнителем, после создания задачи администратором
- Возможность принять заявку

#### Проекты

В платформе должны быть реализованы следующие функции в части информирования пользователя о его проектах:

- Просмотр текущих проектов

#### Чат

В платформе должны быть реализованы следующие функции в части коммуникации между пользователями – участниками проекта. Должно быть доступно окно Чат. Оно должно выполнять функции:

- Отображение истории сообщений

## - Отправка и получение сообщений

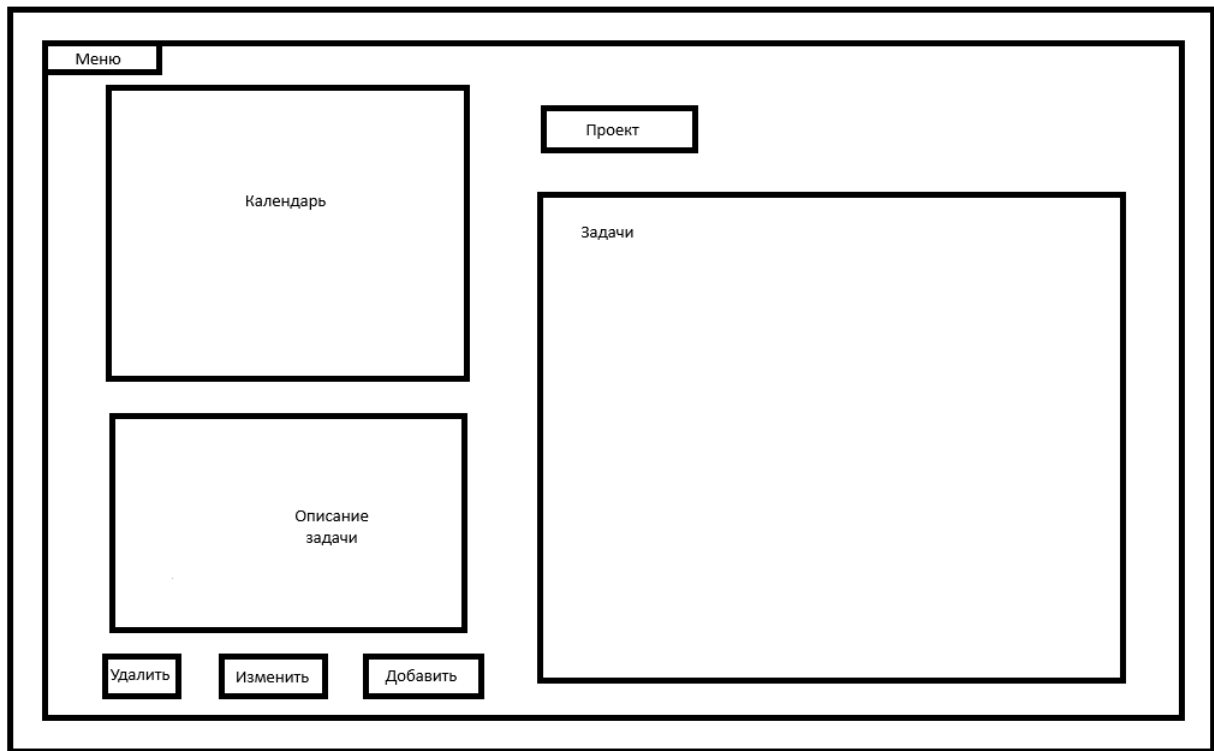


Рисунок 7 – набросок главного окна

### Функции компонента Регистрация

Окно регистрации должно быть доступно администратору, для того чтобы зарегистрировать человека в программе на определенный проект. После чего работник входит в систему с помощью учетных данных, переданных ему администратором.

Должна быть возможность ввести:

- Логин
- Пароль
- ФИО
- Проект, на который назначается человек

Должна быть проверка на корректность введенных данных при нажатии на кнопку зарегистрировать.

Регистрация

Имя

Логин

Пароль

Проект

Зарегистрировать

Рисунок 8 – Набросок окна регистрации

Приложение должно состоять из 2 модулей:

- API – модуль обменивающийся данными с БД на локальном сервере
- Desktopное приложение с пользовательским интерфейсом. Обмен данных с БД через API с помощью http-запросов.

## 2.2 Описание среды разработки

Для разработки программного модуля “Система управления задачами и проектами” были выбраны следующие программные инструменты:

### 1. Среда разработки: PyCharm 2023.2.4

- Описание: PyCharm – это интегрированная среда разработки (IDE) для языка программирования Python. Она предоставляет разработчикам удобный интерфейс, обширные инструменты отладки, автоматическое завершение кода, анализ кода в реальном времени и

другие функции, содействующие эффективному и комфортному процессу разработки.

## 2. Язык программирования: Python

- Описание: Python – высокоуровневый язык программирования, известный своей простотой и читаемостью кода. Он предоставляет разработчикам мощный инструментарий для разнообразных задач и широко используется в веб-разработке, научных исследованиях, анализе данных и создании прикладных программ.

## 3. Платформа: PyQt6

- Описание: PyQt6 представляет собой набор Python-оберток для библиотеки Qt, предназначенной для создания графических пользовательских интерфейсов. PyQt6 обеспечивает разработчиков высокопроизводительными и гибкими инструментами для создания современных и эстетичных интерфейсов приложений. Поддержка PyQt6 обеспечивает широкий спектр функциональности для проекта.

## 4. FastAPI

- Описание: FastAPI представляет собой современный фреймворк для создания веб-приложений на языке Python с акцентом на быстрое действие и автоматическую генерацию документации API. Он обеспечивает простоту использования, поддержку стандарта OpenAPI и встроенные средства валидации запросов и ответов, что делает его отличным выбором для создания веб-сервисов и API.

## 5. Другие инструменты и библиотеки:

- Requests: Библиотека Requests предоставляет простой и удобный способ отправки HTTP-запросов и работы с ответами. Она широко используется для взаимодействия с веб-сервисами, API и другими удаленными ресурсами, предоставляя удобный интерфейс для выполнения различных HTTP-запросов.

Выбранные программные инструменты предоставляют широкий набор функциональности и поддерживают разработку приложений с использованием необходимых технологий и библиотек.

### 2.3 Обоснование выбора инструментария по разработке

Выбранный инструментарий обладает несколькими преимуществами:

- PyCharm предоставляет удобный и интуитивно понятный интерфейс, что упрощает процесс разработки и улучшает продуктивность. Обладает широким функционалом, включая интеграцию с системами контроля версий, предоставляет продвинутое средства отладки и профилирования кода.
- Язык программирования Python обладает простым и лаконичным синтаксисом, поставляется с обширной стандартной библиотекой, предоставляя множество готовых модулей и инструментов для различных задач.
- PyQt6 обеспечивает гибкий инструментарий для создания кроссплатформенных графических интерфейсов, что позволяет модулю работать на различных операционных системах.
- Фреймворк FastAPI обеспечивает высокую производительность благодаря использованию современных асинхронных технологий, что важно для обработки большого количества запросов.
- Библиотека Requests предоставляет простой и интуитивно понятный интерфейс для отправки HTTP-запросов, что упрощает взаимодействие с внешними ресурсами.

Этот выбранный инструментарий лучше всего соответствует требованиям курсовой работы, так как обладает необходимыми функциональными возможностями и предоставляет удобный способ разработки модуля “Система управления задачами и проектами”.

## 2.4 Модуль API

API разделен на модуль с классами (back), и модуль описывающий сервер и http-запросы (FastApi).

Библиотека SQLAlchemy позволяет использовать объектно-ориентированный подход к проектированию БД, что упрощает понимание и реализацию схемы БД. Это означает, что каждая таблица в БД представляется как класс в Python. Поля таблицы представляются атрибутами класса, а связи между таблицами представляются отношениями между классами.

Рассмотрим диаграмму БД (см приложение 1)

Согласно диаграмме, в коде описываются классы, определяющие сущности БД.

Для обмена данными между приложением и сервером используется API. API может быть развернуто как на локальном сервере

Рассмотрим диаграмму классов. (см приложение 2)

Модуль FastApi представляет класс, состоящий из методов – реализующих http запросы к серверу.

В классе DataRequests хранятся запросы, посылаемые приложением к серверу, и ссылки на классы. Запрос содержит ссылку на класс – таблицу в БД.

## 2.5 Структура приложения

Функционал, который предоставляет интерфейс, ограничивается в зависимости от уровня прав пользователя.

Предусмотрено 2 уровня доступа:

Admin: Администратор. Может создавать проекты, регистрировать на них работников, создавать, редактировать, удалять задачи для работников и себя. Видит статусы выполнения задач работниками. Имеет доступ в чат с каждым работником.

User: Сотрудник. Сотрудник, который видит заявки на выполнение задач от Администратора, может принимать их. Имеет доступ к чату с другими сотрудниками и администратором.

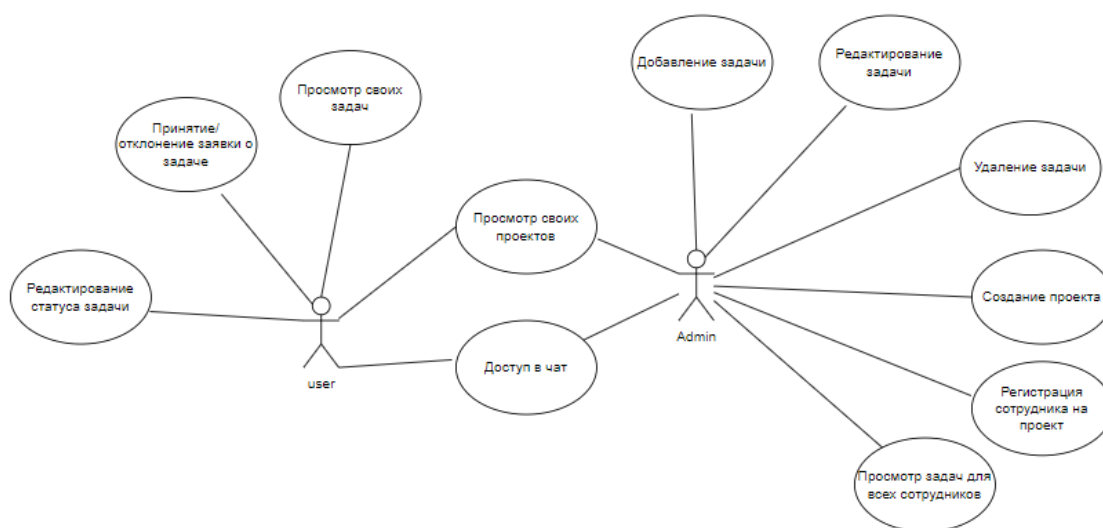


Рисунок 9 – Диаграмма вариантов использования

В этой главе были рассмотрены ключевые аспекты, начиная с формулировки требований к функционалу системы, затем переходя к описанию среды разработки. Особое внимание уделено обоснованию выбора инструментария, что является фундаментом успешной разработки. Модуль API и структура приложения также подробно рассмотрены в контексте общей концепции системы. Вместе эти элементы обеспечивают необходимую функциональность.

### 3. РЕАЛИЗАЦИЯ ПРОЕКТА МОДУЛЯ СИСТЕМЫ

#### 3.1 Описание кода модуля API

API – фактически локальный сервер базы данных. В нем хранится БД.

Код состоит из 2 частей: файла классов и главного файла.

Файл классов отвечает за подключение к БД, создание таблиц, создание сессии.

```
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker, declarative_base

__all__ = ['Tasks', 'User', 'session', 'Chat', 'Project', 'Request',
           'User_project']
engine = create_engine('sqlite:///db.db')
Base = declarative_base()

class Tasks(Base)
class User(Base)
class Chat(Base)
class Project(Base)
class Request(Base)
class User_project(Base)

Base.metadata.create_all(engine)
Sessionlocal = sessionmaker(bind=engine)
session = Sessionlocal()
```

Здесь происходит создание объекта engine для взаимодействия с базой данных SQLite. Определение моделей таблиц. Создание таблиц в базе данных с помощью метаданных моделей. Создание сессии для взаимодействия с данными в таблицах.

Главный файл инициализирует приложение FastApi

```
from fastapi import FastAPI, Body, HTTPException
from sqlalchemy.exc import NoResultFound

from back import *
from sqlalchemy import and_, or_
from datetime import datetime

app = FastAPI()

class DataRequests

session.close()
```

DataRequests - описывает http-запросы к серверу. По окончании сессия закрывается. Запуск локального сервера осуществляется командой терминала `uvicorn main:app --reload`.



## 3.2 Описание десктопного приложения

### 3.2.1 Запуск

Программа начинает выполнение с файла main.py:

```
def main():
    import sys
    from app import app
    from auth import Login

    auth_window = Login()
    auth_window.show()

    def close_all_windows():
        for widget in app.topLevelWidgets():
            widget.close()

    app.aboutToQuit.connect(close_all_windows)

    sys.exit(app.exec())
```

Здесь импортируется app – инициализированный экземпляр приложения из файла app.py

```
import sys
from PyQt6 import QtWidgets
app = QtWidgets.QApplication(sys.argv)
```

Затем импортируется окно авторизации, и запускается приложение. Также прописана пользовательская функция close\_all\_windows, отвечающая за закрытие всех окон при закрытии главного окна приложения

### 3.2.2 Окно авторизации

Представляет собой окно с 2 полями ввода логина и пароля соответственно и кнопкой войти.

При инициализации создается дополнительный поток, отвечающий за проверку соединения с сервером.

Спроектированный пользовательский интерфейс окна авторизации выглядит следующим образом:

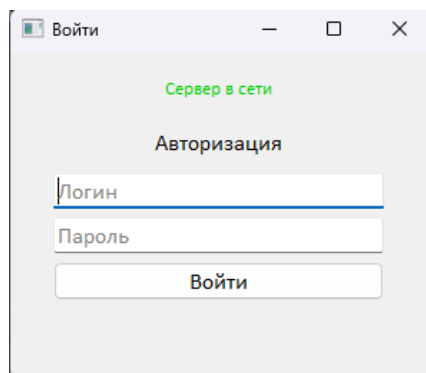


Рисунок 10 – Окно авторизации

Данные считываются с полей ввода, и в зависимости от ответа сервера происходит авторизация, или сообщение об ошибке. Функция авторизации:

```
def login(self):
    """Авторизация пользователя"""
    if self.st_label.text() != "Сервер в сети":
        return
    if self.loginEdit.text() != '' and self.passEdit.text() != '':
        response = requests.get(
f"{BASE_URL}/auth?login={str(self.loginEdit.text())}&password={str(self.passEdit
.text())}")
        if response.status_code == 404:
            self.error_label.setText('Неправильное имя пользователя\или
пароль')
        elif response.status_code == 200:
            user_id = int(response.json()['id'])
            permission = response.json()['root']

            from todo import MainWindow

            self.mw = MainWindow(user_id, permission)
            self.mw.show()

            self.close()
        else:
            self.error_label.setText('Что-то пошло не так')
    else:
        self.error_label.setText('Поля не могут быть пустыми')
```

### 3.2.3 Главное окно

Главное окно содержит календарь, таблицу вывода дел, поля вывода описаний дел.

Спроектированный пользовательский интерфейс главного окна выглядит следующим образом:

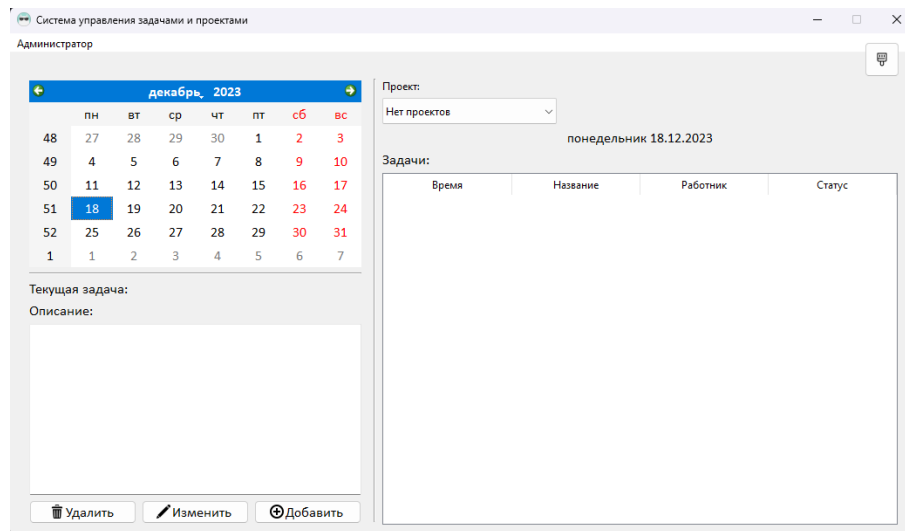


Рисунок 11 – Главное окно

При инициализации создаются объекты дочерних окон: профиль, чат, проекты, регистрация, добавления дела, изменение дела. Выполняется запрос для отображения списка дел на текущую дату. Часть функции отображения списка дел:

```
for row in range(len(self.lst_do)):
    for col in range(self.listWidget.columnCount()):
        time = self.lst_do[row]['time'].split(':')
        if self.lst_do[row]["status"] != 2 and self.permission == 'user' and
self.lst_do[row]["user_id"] == self.user_id:
            item = None
            if col == 0:
                item = QTableWidgetItem(f'{time[0]}:{time[1]}')
            elif col == 1:
                item =
QTableWidgetItem(f'{self.lst_do[row]["title"]}')
            elif col == 2:
                item = QTableWidgetItem(
                    f'"Сделано" if self.lst_do[row]["status"] == True else "На
выполнении"')
                self.listWidget.setItem(row, col, item)
            elif self.permission == 'admin':
                item = None
                user =
requests.get(f"{BASE_URL}/list_users?user_id={self.lst_do[row]['user_id']}").jso
n()
                if col == 0:
                    item = QTableWidgetItem(f'{time[0]}:{time[1]}')
                elif col == 1:
                    item =
QTableWidgetItem(f'{self.lst_do[row]["title"]}')
                elif col == 2:
                    item = QTableWidgetItem(f'{user['FIO']}')
                elif col == 3:
                    item = QTableWidgetItem(
                        f'"Сделано" if self.lst_do[row]["status"] == True else "На
выполнении"')
                    self.listWidget.setItem(row, col, item)
```

Здесь в зависимости от прав человека формируется и заполняется таблица дел.

### 3.2.4 Дочерние окна

#### 3.2.4.1 Чат.

Спроектированный пользовательский интерфейс окна чата выглядит следующим образом:

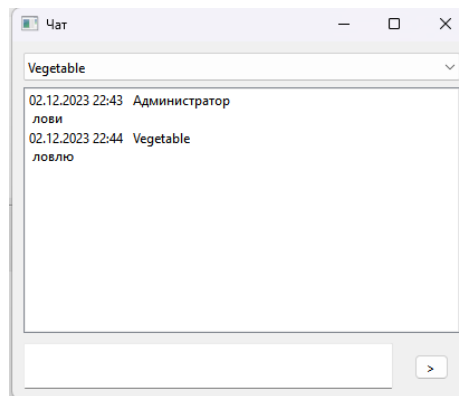


Рисунок 12 – Окно чата

При появлении запрашивает и формирует выпадающий список участников проекта. При выборе человека отображает историю переписки с ним. Функция отображения истории чата:

```
def render_ch(self):
    self.listWidget.clear()
    r_id = 0
    for i in self.query:
        if self.combo.currentText() == i['FIO']:
            r_id = i['id']
    query =
requests.get(f'{BASE_URL}/history_chat?sender_id={self.user_id}&reciever_id={r_id}')
    self.sort_hist = sorted(query.json(), key=lambda x: (x['date'], x['time']))

    for i in self.sort_hist:
        time = i['time'].split(':')
        date = datetime.strptime(i['date'], '%Y-%m-%d').date().strftime("%d.%m.%Y")
        self.listWidget.addItem(f'{date} {time[0]}:{time[1]} {self.get_sender(i['sender_id'])}\n {i['message']}
```

#### 3.2.4.2 Профиль.

Спроектированный пользовательский интерфейс окна профиля выглядит следующим образом:

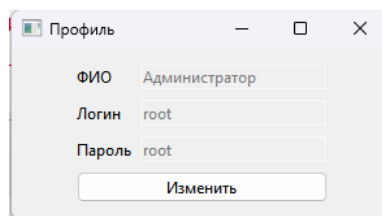


Рисунок 13 – Окно профиля

При инициализации вызывается функция, которая запрашивает и отображает данные пользователя:

```
def render(self):
    self.query = requests.get(f"{BASE_URL}/profile/{self.user_id}").json()
    fio, login, password = str(self.query['FIO']), str(self.query['login']),
str(self.query['password'])
    self.lineEdit_2.setText(fio)
    self.lineEdit.setText(login)
    self.lineEdit_3.setText(password)
```

### 3.2.4.3 Добавление дела

Спроектированный пользовательский интерфейс окна добавления дела выглядит следующим образом:

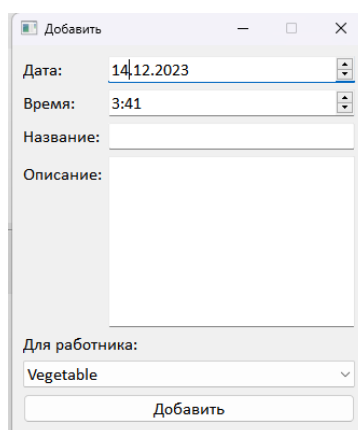


Рисунок 14 – Окно добавления дела

При отображении окно формирует выпадающий список пользователей, которым можно назначит дело. Часть функции добавление дела:

```
data = {
    'date': str(self.dateEdit.date().toPyDate()),
    'time':
str(self.timeEdit.time().currentTime().toPyTime().strftime('%H:%M:%S')),
    'title': self.titleLineEdit.text(),
    'description': str(self.descEdit.toPlainText()),
    'status': 0 if self.user_id == u_id else 2,
    'user_id': u_id,
```

```

        'project_id': self.proj_id,
    }
    query = requests.post(f'{BASE_URL}/add', json=data)
    if self.user_id != u_id:
        data2 = {
            'sender_id': self.user_id,
            'task_id': query.json()['id'],
        }
    requests.post(f'{BASE_URL}/add2', json=data2)

```

Здесь формируются 2 словаря данных, один для информации о деле, второй для записи о заявке о деле. После чего отправляется запрос с данными на сервер.

#### 3.2.4.4 Добавление пользователя

Спроектированный пользовательский интерфейс окна регистрации пользователя выглядит следующим образом:

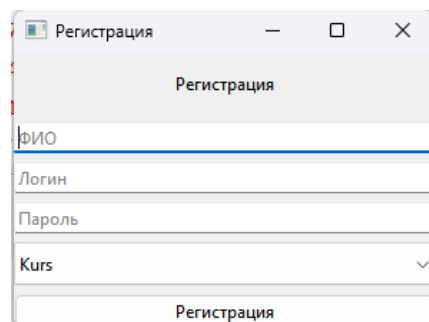


Рисунок 15 – Окно регистрации пользователя

При инициализации окна заполняется список проектов на который можно зарегистрировать пользователя. Часть функции регистрации:

```

data = {
    'FIO': self.fio_label.text(),
    'login': str(self.login_edit.text()),
    'password': self.password_edit.text(),
    'root': "user",
}
q = requests.post(f'{BASE_URL}/add_user', json=data).json()
pr_id = 0
for i in self.pr:
    if self.combo.currentText() == i['name']:
        pr_id = i['id']
data2 = {
    'user_id': q['id'],
    'project_id': pr_id
}
requests.post(f'{BASE_URL}/user_to_project', json=data2)

```

Здесь также формируются 2 словаря данных, один для информации о человеке, второй для записи о человека на выбранный проект. После чего отправляется запрос с данными на сервер.

#### 3.2.4.5 Окно работы с проектами

Позволяет администратору просматривать, добавлять, удалять проекты.

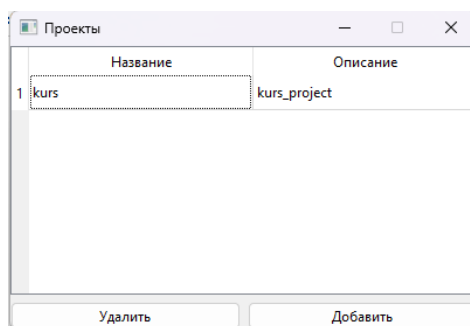


Рисунок 16 – Окно работы с проектами

#### 3.2.4.6 Окно уведомлений

Позволяет работнику просматривать уведомления о задачах, и принимать их к исполнению.

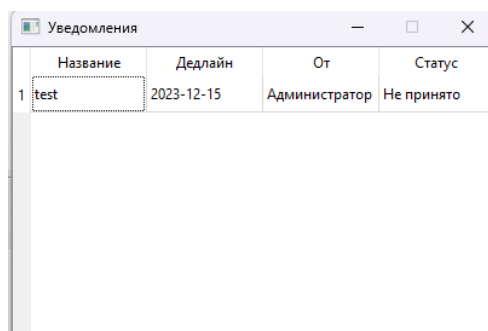


Рисунок 17 – Окно уведомлений сотрудника

### 3.3 Тестирование

Было выполнено тестирование согласно плану тестирования (см. приложение 3).

Также было выполнено:

- Интеграционное тестирование: фокусировалось на проверке взаимодействия между различными модулями и компонентами приложения. Это включало в себя тестирование API, базы данных и интерфейсов взаимодействия.
- Системное тестирование: проводилось на уровне всего приложения. Основной задачей было удостовериться в правильной работе всех функций, соответствии интерфейса требованиям и общей стабильности системы.
- Ручное тестирование: проводилось с целью оценки пользовательского опыта, проверки соответствия интерфейса дизайну и выявления неочевидных проблем.

На протяжении тестирования были выявлены и успешно устранены различные дефекты. Производительность приложения оказалась на высоком уровне, а все функциональные требования были полностью выполнены. Тестирование десктопного приложения "Система управления задачами и проектами" сыграло важную роль в обеспечении его надежной и стабильной работы.



## ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были проанализированы существующие подходы и системы управления проектами, определены требования к Системе, была разработана и протестирована архитектура и дизайн Системы.

Разработанная Система является мощным инструментом, который может значительно повысить эффективность организационной деятельности.

Система обеспечивает следующие преимущества:

- Повышение эффективности работы команд: система позволяет сотрудникам легко координировать свои действия, отслеживать прогресс и вносить изменения в планы по мере необходимости.
- Уменьшение временных и ресурсных затрат: система автоматизирует многие задачи, связанные с управлением проектами, что позволяет сотрудникам сосредоточиться на более важных аспектах своей работы.
- Улучшение прозрачности и обоснованности принятия решений: система предоставляет руководителям подробную информацию о ходе выполнения проектов.

Система была протестирована с использованием различных тестов и показала, что она работает корректно и соответствует требованиям.

Можно доработать систему, добавив следующие функции:

- Возможность отслеживать время, затрачиваемое на задачи и проекты. Эта функция позволит руководителям оценивать эффективность использования рабочего времени сотрудников.
- Возможность создавать отчеты о выполнении задач и проектов. Эти отчеты будут полезны для руководителей, которые хотят отслеживать прогресс проектов и оценивать их результаты.
- Возможность интеграции с другими системами. Эта функция позволит объединить систему управления задачами и проектами с другими системами.

## СПИСОК ИСТОЧНИКОВ

### Нормативно-правовые источники

1. ГОСТ 7.32–2017 Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления: Стандартинформ, 2017

2. ГОСТ 7.1—2003 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления

3. ГОСТ 7.9—95 Система стандартов по информации, библиотечному и издательскому делу. Реферат и аннотация. Общие требования

4. ГОСТ 7.11—2004 (ИСО 832:1994) Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов и словосочетаний на иностранных европейских языках

5. ГОСТ 7.12—93 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Сокращение слов на русском языке. Общие требования и правила

6. ГОСТ 7.80—2000 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Заголовок. Общие требования и правила составления

7. ГОСТ 7.82—2001 Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание электронных ресурсов. Общие требования и правила составления

### Учебники, учебные пособия, статьи.

8. Чернышев, С. А. Основы программирования на Python: учебное пособие для среднего профессионального образования / С. А. Чернышев. — 2-е изд., перераб. и доп. — Москва: Издательство Юрайт, 2023.

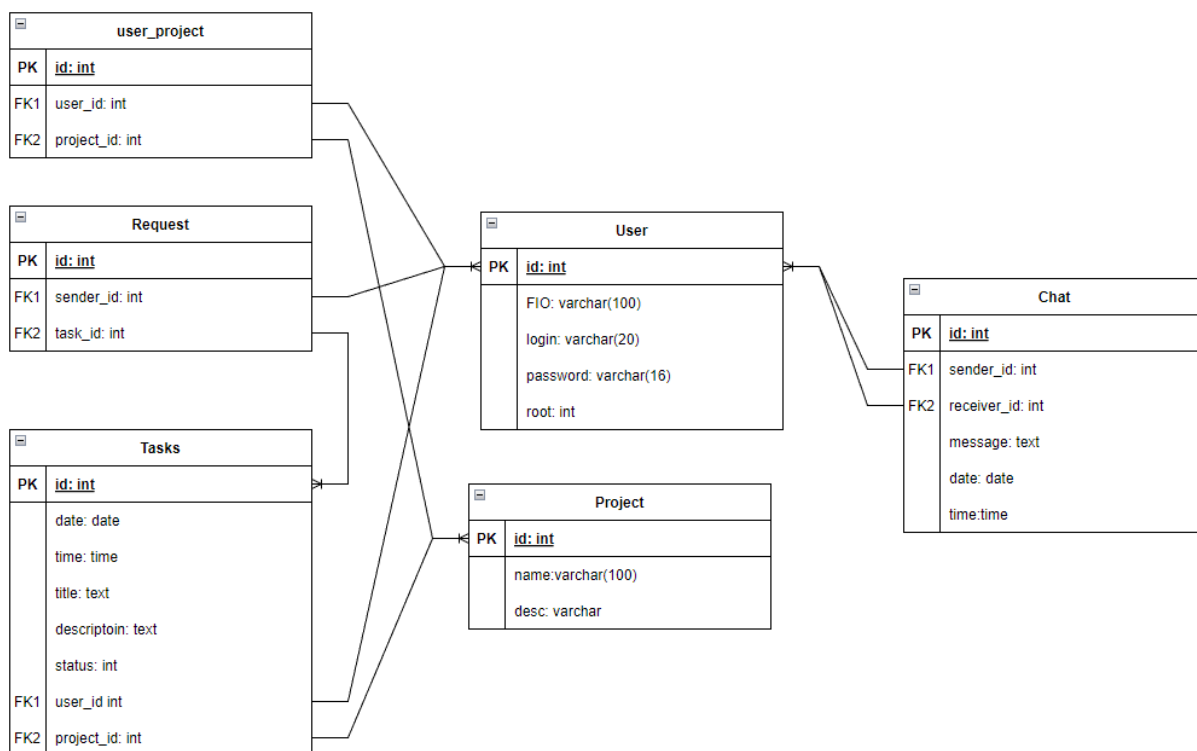
9. Программирование на SQL Маркин, А. В. Программирование на SQL: учебное пособие для среднего профессионального образования / А. В. Маркин. — Москва: Издательство Юрайт

### Интернет-источники

10. Добрый, добрый Python - обучающий курс от Сергея Балакирева // <https://stepik.org/course/100707/syllabus>
11. Добрый, добрый Python ООП - обучающий курс от Сергея Балакирева // <https://stepik.org/course/116336/syllabus>
12. Qt for Python // <https://doc.qt.io/qtforpython-6/> (дата обращения: 16.12.2023).
13. Работа с SQLite в Python (для чайников) / Хабр // <https://habr.com/ru/articles/754400/> (дата обращения: 16.12.2023).
14. Python и FastAPI | Подключение к базе данных и создание таблиц // <https://metanit.com/python/fastapi/2.1.php> (дата обращения: 16.12.2023).
15. Знакомство с FastAPI / Хабр // <https://habr.com/ru/articles/488468/>
16. FastAPI // <https://fastapi.tiangolo.com/> (дата обращения: 16.12.2023).
17. Как сделать из Python-скрипта исполняемый файл / Хабр // <https://habr.com/ru/companies/slurm/articles/746622/> (дата обращения: 16.12.2023).
18. Работа с датой и временем в Python // <https://python-scripts.com/datetime-time-python> (дата обращения: 16.12.2023).
19. Jira | Программное обеспечение для отслеживания задач // <https://www.atlassian.com/ru/software/jira>
20. Manage Your Team's Projects From Anywhere | Trello // <https://trello.com>
21. Битрикс24 + AI. Бесплатный онлайн-сервис для бизнеса! // <https://www.bitrix24.ru>
22. Asana. Работайте над большими идеями без лишней суеты. // <https://asana.com/ru>
23. ПО для управления проектами | Microsoft Project // <https://www.microsoft.com/ru-ru/microsoft-365/project/project-management-software>

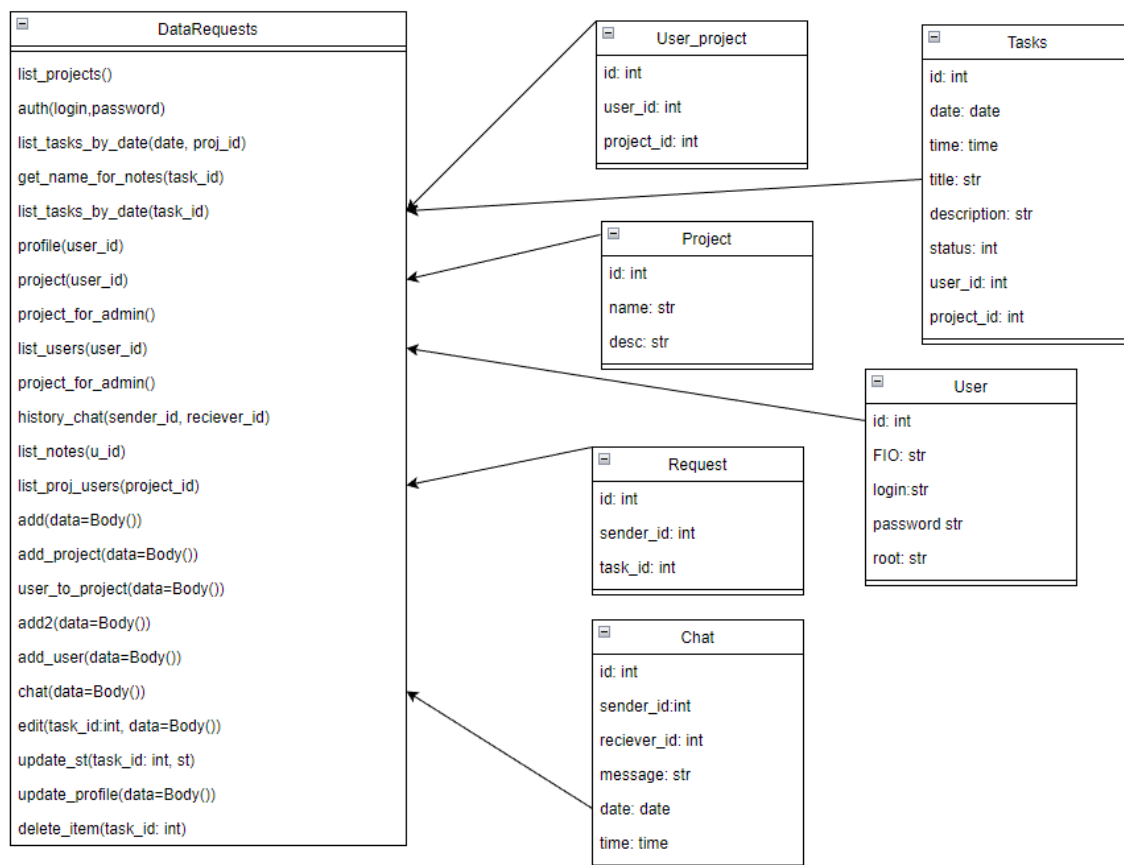
# Приложение 1

## Диаграмма БД



## Приложение 2

### Диаграмма классов



# Приложение 3

## План тестирования

№	Наименование функциональности	Наименование поля	Тестовый набор	Результат (должно получиться)	Результат тестирования
1	Редактирование, добавление дела	Название	Ничего не введено	Согласен с разработчиком	Сообщение: «Поля не могут быть пустыми»
		Описание	Ничего не введено	Согласен с разработчиком	Сообщение: «Поля не могут быть пустыми»
		Для работника	Выбран пункт «Не найдено»	Согласен с разработчиком	Сообщение: «Ты не можешь добавить задачу этому пользователю. Проверь, есть ли у тебя люди в проекте»
2	Авторизация	Логин и пароль	Ввод любых не подходящих значений	Согласен с разработчиком	Ошибка: «Неверный логин или пароль»
		Логин и пароль	Правильные данные авторизации	Согласен с разработчиком	Пользователь авторизован
3	Фильтр дел по дате	Задачи	Выбрана другая дата.	Согласен с разработчиком	Отображается список дел на дату. Дел нет – список пустой
		Описание	Выбрано дела	Согласен с разработчиком	Отобразилось описание дела.

## Листинг 1. Класс DataRequests

```
from fastapi import FastAPI, Body, HTTPException
from sqlalchemy.exc import NoResultFound

from back import *
from sqlalchemy import and_, or_
from datetime import datetime

app = FastAPI()

class DataRequests:
    @staticmethod
    @app.get('/connection')
    async def connection():
        return True

    @staticmethod
    @app.get('/list_projects')
    async def list_projects():
        return session.query(Project).all()

    @staticmethod
    @app.get('/auth')
    async def auth(login, password):
        try:
            return session.query(User).filter(and_(User.login ==
str(login), User.password == str(password))).one()
        except NoResultFound as e:
            raise HTTPException(status_code=404, detail=str(e))

    @staticmethod
    @app.get('/list_tasks_by_date')
    async def list_tasks_by_date(date, proj_id):
        try:
            return session.query(Tasks).filter(
                and_(Tasks.date == date, Tasks.project_id ==
proj_id)).all()
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.get('/get_name_for_notes')
    async def get_name_for_notes(task_id):
        try:
            q = session.query(Request).filter(Request.task_id ==
task_id).first()
```

```

        u = session.query(User).filter(User.id ==
q.sender_id).one()
        return u.FIO
    except Exception as e:
        print('Ошибка:', e)

    @staticmethod
    @app.get('/list_tasks_by_id/{task_id}')
    async def list_tasks_by_id(task_id):
        try:
            return session.query(Tasks).where(Tasks.id ==
task_id).one()
        except Exception as e:
            return e

    @staticmethod
    @app.get('/profile/{user_id}')
    async def profile(user_id):
        try:
            return session.query(User).where(User.id ==
user_id).one()
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.get('/project/{user_id}')
    async def project(user_id):
        try:
            return session.query(Project).where(
                and_(User_project.project_id == Project.id,
User_project.user_id == user_id)).all()
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.get('/project_for_admin')
    async def project_for_admin():
        try:
            return session.query(Project).all()
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.get('/list_users')
    async def list_users(user_id):
        return session.query(User).filter(User.id ==
user_id).one()

    @staticmethod
    @app.get('/history_chat')
    async def history_chat(sender_id, reciever_id):
        return session.query(Chat).where(and_(or_(Chat.reciever_id

```



```

== sender_id, Chat.reciever_id == reciever_id),
                                or_(Chat.sender_id
== sender_id, Chat.sender_id == reciever_id))).all()

    @staticmethod
    @app.get('/list_notes')
    async def list_notes(u_id):
        query = session.query(Tasks).where(Tasks.user_id ==
u_id).all()
        return query

    @staticmethod
    @app.get('/list_proj_users/{project_id}')
    async def list_proj_users(project_id):
        return session.query(User).where(
            and_(User_project.user_id == User.id,
User_project.project_id == project_id)).all()

    @staticmethod
    @app.post('/add')
    async def add(data=Body()):
        try:
            task = Tasks(date=datetime.strptime(data["date"], '%Y-
%m-%d').date(),
                                time=datetime.strptime(data["time"],
'%H:%M:%S').time(),
                                title=data["title"],
description=data["description"], status=data["status"],
                                user_id=data["user_id"],
                                project_id=data['project_id'])
            session.add(task)
            session.commit()
            session.refresh(task)
            return task
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.post('/add_project/{user_ud}')
    async def add_project(user_ud, data=Body()):
        try:
            pr = Project(
                name=data["name"], desc=data["desc"])
            session.add(pr)
            session.commit()
            session.refresh(pr)

            pr_user = User_project(user_id = user_ud, project_id =
pr.id)

            session.add(pr_user)
            session.commit()
            session.refresh(pr_user)

```

```

        return pr

    except Exception as e:
        print('Ошибка:', e)

    @staticmethod
    @app.post('/user_to_project')
    async def user_to_project(data=Body()):
        try:
            u_p = User_project(user_id=data["user_id"],
project_id=data["project_id"])
            session.add(u_p)
            session.commit()
            session.refresh(u_p)
            return u_p
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.post('/add2')
    async def add2(data=Body()):
        try:
            req = Request(sender_id=data["sender_id"],
task_id=data["task_id"])
            session.add(req)
            session.commit()
            session.refresh(req)
            return req
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.post('/add_user')
    async def add_user(data=Body()):
        try:
            user = User(FIO=data["FIO"], login=data["login"],
password=data["password"], root=data["root"])
            session.add(user)
            session.commit()
            session.refresh(user)
            return user
        except Exception as e:
            print('Ошибка:', e)

    @staticmethod
    @app.post('/chat')
    async def chat(data=Body()):
        try:
            ch = Chat(sender_id=data['sender_id'],
reciever_id=data['reciever_id'],
message=data["message"],
date=datetime.strptime(data["date"], '%Y-%m-%d').date(),

```

```

        time=datetime.strptime(data["time"],
'%H:%M:%S').time())
        session.add(ch)
        session.commit()
        session.refresh(ch)
        return ch
    except Exception as e:
        print('Ошибка:', e)

    @staticmethod
    @app.put('/edit/{task_id}')
    async def edit(task_id: int, data=Body()):
        # Получаем задачу по идентификатору
        task = session.query(Tasks).filter_by(id=task_id).one()

        # Проверяем, найдена ли задача
        if task is None:
            raise HTTPException(status_code=404, detail="Задача не
найдена")
        # Обновляем поля задачи
        if "date" in data:
            task.date = datetime.strptime(data["date"], '%Y-%m-
%d').date()
        if "time" in data:
            task.time = datetime.strptime(data["time"],
'%H:%M:%S').time()
        if "title" in data:
            task.title = data["title"]
        if "description" in data:
            task.description = data["description"]
        if "status" in data:
            task.status = data["status"]
        if "user_id" in data:
            task.user_id = data["user_id"]

        # Фиксируем изменения в базе данных
        session.commit()
        session.refresh(task)

        return task

    @staticmethod
    @app.put('/update_st')
    async def update_st(task_id: int, st):
        task = session.query(Tasks).filter_by(id=task_id).one()

        # Проверяем, найдена ли задача
        if task is None:
            raise HTTPException(status_code=404, detail="Задача не
найдена")
        # Обновляем поля задачи
        task.status = st

```

```

        session.commit()
        session.refresh(task)

    return task

    @staticmethod
    @app.put('/update_profile')
    async def update_profile(data=Body()):
        pr = session.query(User).filter_by(id=data['id']).one()

        # Проверяем, найдена ли задача
        if pr is None:
            raise HTTPException(status_code=404, detail="Задача не
найдена")

        # Обновляем поля задачи
        if "FIO" in data:
            pr.FIO = data['FIO']
        if "login" in data:
            pr.login = data['login']
        if "password" in data:
            pr.password = data['password']

        session.commit()
        session.refresh(pr)

    return pr

    @staticmethod
    @app.delete("/tasks/{task_id}")
    async def delete_item(task_id: int):
        item = session.query(Tasks).filter(Tasks.id ==
task_id).one()
        req = session.query(Request).filter(Request.task_id ==
task_id).all()

        session.delete(item)
        for i in req:
            session.delete(i)
        session.commit()

    @staticmethod
    @app.delete("/project/{name}")
    async def delete_project_id(name: str):
        item = session.query(Project).filter(Project.name ==
str(name)).one()
        u_p =
session.query(User_project).filter(User_project.project_id ==
item.id).all()
        t_p = session.query(Tasks).filter(Tasks.project_id ==
item.id).all()
        session.delete(item)

```

```

        for i in t_p:
            session.delete(i)
        for i in u_p:
            u = session.query(User).filter(User.id ==
i.user_id).one()
            ch = session.query(Chat).where(or_(Chat.reciever_id ==
u.id, Chat.reciever_id == u.id)).all()
            for i in ch:
                session.delete(i)
            session.delete(i)
            if u.root != 'admin':
                session.delete(u)

session.commit()

```

## Листинг 2. Код сущностей SQLAlchemy

```
from sqlalchemy import *
from sqlalchemy.orm import sessionmaker, declarative_base

__all__ = ['Tasks', 'User', 'session', 'Chat', 'Project',
           'Request', 'User_project']
engine = create_engine('sqlite:///db.db')
Base = declarative_base()

class Tasks(Base):
    __tablename__ = 'task'

    id = Column(Integer, primary_key=True)
    date = Column(Date, nullable=False)
    time = Column(Time, nullable=False)
    title = Column(Text, nullable=False)
    description = Column(Text, nullable=False)
    status = Column(Integer)
    user_id = Column(Integer, ForeignKey('user.id'))
    project_id = Column(Integer, ForeignKey('project.id'))

class User(Base):
    __tablename__ = 'user'

    id = Column(Integer, primary_key=True)
    FIO = Column(String(100), nullable=False)
    login = Column(String(20), nullable=False)
    password = Column(String(16), nullable=False)
    root = Column(Text, nullable=False, default='user')

class Chat(Base):
    __tablename__ = 'chat'

    id = Column(Integer, primary_key=True)
    sender_id = Column(Integer, ForeignKey('user.id'))
    reciever_id = Column(Integer, ForeignKey('user.id'))
    message = Column(String, nullable=False)
    date = Column(Date, nullable=False)
    time = Column(Time, nullable=False)

class Project(Base):
    __tablename__ = 'project'

    id = Column(Integer, primary_key=True)
    name = Column(String(100), nullable=False)
    desc = Column(String, nullable=False)

class Request(Base):
    __tablename__ = 'request'

    id = Column(Integer, primary_key=True)
    sender_id = Column(Integer, ForeignKey('user.id'))
    task_id = Column(Integer, ForeignKey('task.id'))

class User_project(Base):
```

```

__tablename__ = 'user_project'
id = Column(Integer, primary_key=True)
user_id = Column(Integer, ForeignKey('user.id'))
project_id = Column(Integer, ForeignKey('project.id'))

```

```

Base.metadata.create_all(engine)
Sessionlocal = sessionmaker(bind=engine)
session = Sessionlocal()

```

### Листинг 3. Код файла main.py

```

def main():
    import sys
    from app import app
    from auth import Login

    auth_window = Login()
    auth_window.show()

    def close_all_windows():
        for widget in app.topLevelWidgets():
            widget.close()

    app.aboutToQuit.connect(close_all_windows)

    sys.exit(app.exec())

if __name__ == "__main__":
    main()

```

### Листинг 4. Код файла app.py

```

import sys
from PyQt6 import QtWidgets
app = QtWidgets.QApplication(sys.argv)

```

### Листинг 5. Код файла connection.py

```

import configparser
config = configparser.ConfigParser()
config.read("connect.ini")
BASE_URL = f"{str(config["Connection"]["BASE_URL"]).strip('"')}"

```

### Листинг 6. Код окна авторизации

```

from PyQt6 import QtGui, QtWidgets, QtCore
from PyQt6.QtWidgets import QWidget, QLineEdit
import httpx
import asyncio
import requests

from connection import BASE_URL

async def ping_serv(url):

```

```

try:
    async with httpx.AsyncClient() as client:
        response = await client.get(url, timeout=10)
        if response.status_code == 200:
            return 'Сервер в сети'
        else:
            return f"Код ответа:{response.status_code}"
except httpx.TimeoutException:
    return "Превышено время ожидания"
except httpx.ConnectError:
    return f"Сервер не в сети"

class Worker(QtCore.QObject):
    finished = QtCore.pyqtSignal(str)

    def __init__(self):
        super().__init__()
        self.loop = asyncio.new_event_loop()
        asyncio.set_event_loop(self.loop)

    async def run_async(self):
        while True:
            result = await ping_serv(f"{BASE_URL}/connection")
            self.finished.emit(result)
            await asyncio.sleep(2)

    def run(self):
        self.loop.run_until_complete(self.run_async())

class Login(QWidget):

    def update_st_label(self, result):
        self.st_label.setText(result)
        self.st_label.setStyleSheet(
            "QLabel{color: rgb(0, 207, 0);}" if result == 'Сервер
в сети' else self.st_label.setStyleSheet(
            "QLabel{color: rgb(253,44,2);}")

    def keyPressEvent(self, event: QtGui.QKeyEvent):
        if event.key() == QtCore.Qt.Key.Key_Enter or event.key()
== QtCore.Qt.Key.Key_Return:
            self.loginButton.click()
        else:
            super().keyPressEvent(event)

    def login(self):
        """Авторизация пользователя"""
        if self.st_label.text() != "Сервер в сети":
            return
        if self.loginEdit.text() != '' and self.passEdit.text() !=

```



```

':
    response = requests.get(

f"{BASE_URL}/auth?login={str(self.loginEdit.text())}&password={str
(self.passEdit.text())}"
    if response.status_code == 404:
        self.error_label.setText('Неправильное имя
пользователя\или пароль')
    elif response.status_code == 200:
        user_id = int(response.json()['id'])
        permission = response.json()['root']

        from todo import MainWindow

        self.mw = MainWindow(user_id, permission)
        self.mw.show()

        self.close()
    else:
        self.error_label.setText('Что-то пошло не так')
else:
    self.error_label.setText('Поля не могут быть пустыми')

```

## Листинг 7. Код главного окна

```
import locale

locale.setlocale(locale.LC_ALL, 'ru_RU.utf8')
from PyQt6.QtGui import QAction, QPalette
from PyQt6.QtWidgets import QMainWindow, QMenuBar, QMessageBox,
QApplication
from PyQt6 import QtCore, QtGui, QtWidgets
from add import Add
from edit import Edit
from datetime import datetime
from chat import Chat
from profile import Profile
from project import Project
from notifications import Note
from auth import Login
import requests

from connection import BASE_URL

class MainWindow(QMainWindow):
    """Главное окно"""

    def set_theme(self):
        from app import app
        if app.styleSheet() == "":
            from pathlib import Path
            import qdarktheme
            dark_palette = qdarktheme.load_palette()
            app.setPalette(dark_palette)
            app.setStyleSheet(Path('style.css').read_text())

    self.addButton.setIcon(QtGui.QIcon('resources/ico/add_white.svg'))

    self.editButton.setIcon(QtGui.QIcon('resources/ico/edit_white.svg'
))

    self.rmButton.setIcon(QtGui.QIcon('resources/ico/delete_white.svg'
))

    self.button_style.setIcon(QtGui.QIcon('resources/ico/theme_white.s
vg'))

    self.profile.setIcon(QtGui.QIcon('resources/ico/profile_white.svg'
))

    self.notifications_action.setIcon(QtGui.QIcon('resources/ico/notes
_white.svg'))

    self.projects_action.setIcon(QtGui.QIcon('resources/ico/notes_whit
```

```

e.svg'))

self.chat_action.setIcon(QtGui.QIcon('resources/ico/chat_white.svg'))

self.exit_action.setIcon(QtGui.QIcon('resources/ico/exit_white.svg'))

self.reg_action.setIcon(QtGui.QIcon('resources/ico/add_white.svg'))
)
        self.menubar.setStyleSheet("color: rgb(255, 255,
255);")
    else:
        app.setStyleSheet("")
        empty_palette = QPalette()
        app.setPalette(empty_palette)

self.addButton.setIcon(QtGui.QIcon('resources/ico/add.svg'))

self.editButton.setIcon(QtGui.QIcon('resources/ico/edit.svg'))

self.rmButton.setIcon(QtGui.QIcon('resources/ico/delete.svg'))

self.button_style.setIcon(QtGui.QIcon('resources/ico/theme.svg'))

self.profile.setIcon(QtGui.QIcon('resources/ico/profile.svg'))

self.notifications_action.setIcon(QtGui.QIcon('resources/ico/notes
.svg'))

self.projects_action.setIcon(QtGui.QIcon('resources/ico/notes.svg'))
))

self.chat_action.setIcon(QtGui.QIcon('resources/ico/chat.svg'))

self.exit_action.setIcon(QtGui.QIcon('resources/ico/exit.svg'))

self.reg_action.setIcon(QtGui.QIcon('resources/ico/add.svg'))
        self.menubar.setStyleSheet("color: rgb(0,0,0);")

    def set_menu(self):
        """Меню пользователя"""

self.Me.setTitle(str(requests.get(f"{BASE_URL}/profile/{self.user_
id}").json()['FIO']))

    def for_permission(self):
        """Проверка разрешений пользователя"""
        if self.permission == 'user':
            self.Me.removeAction(self.reg_action)

```

```

        self.addButton.close()
        self.rmButton.close()
    else:
        self.Me.removeAction(self.notifications_action)

    def dateview(self):
        """Отображение выбранной даты"""
        self.calendarWidget.setLocale(
            QtCore.QLocale(QtCore.QLocale.Language.Russian,
QtCore.QLocale.Country.Russia))
        day =
datetime.strptime(str(self.calendarWidget.selectedDate().toPyDate(
)), '%Y-%m-%d').date().strftime("%A")
        date =
datetime.strptime(str(self.calendarWidget.selectedDate().toPyDate(
)), '%Y-%m-%d').date().strftime(
            "%d.%m.%Y")
        self.current_date_label.setText(f"{day} {date}")

    def renderList(self):
        """Запрос и парсинг списка дел по дате из БД"""
        self.current_matter_label.setText('Текущая задача: ')
        self.textDescription.clear()
        self.listWidget.clear()
        self.lst_do = requests.get(

f"{BASE_URL}/list_tasks_by_date?date={str(self.calendarWidget.sele
ctedDate().toPyDate())}&proj_id={self.selected_proj}").json()
        self.listWidget.setRowCount(len(self.lst_do))
        if self.permission == 'admin':
            self.listWidget.setColumnCount(4)
            self.listWidget.setHorizontalHeaderLabels(['Время',
'Название', 'Работник', 'Статус'])
        else:
            self.listWidget.setColumnCount(3)
            self.listWidget.setHorizontalHeaderLabels(['Время',
'Название', 'Статус'])
        for row in range(len(self.lst_do)):
            for col in range(self.listWidget.columnCount()):
                time = self.lst_do[row]['time'].split(':')
                if self.lst_do[row]["status"] != 2 and
self.permission == 'user' and self.lst_do[row][
                    "user_id"] == self.user_id:
                    item = None
                    if col == 0:
                        item =
QtWidgets.QTableWidgetItem(f'{{time[0]}}:{{time[1]}}')
                    elif col == 1:
                        item =
QtWidgets.QTableWidgetItem(f'{{self.lst_do[row]["title"]}}')
                    elif col == 2:
                        item = QtWidgets.QTableWidgetItem(

```

```

        f'{"Сделано" if
self.lst_do[row]["status"] == True else "На выполнении"}')
        self.listWidget.setItem(row, col, item)
        elif self.permission == 'admin':
            item = None
            user =
requests.get(f"{BASE_URL}/list_users?user_id={self.lst_do[row]['us
er_id']}").json()
            if col == 0:
                item =
QtWidgets.QTableWidgetItem(f'{{time[0]}:{{time[1]}}')
            elif col == 1:
                item =
QtWidgets.QTableWidgetItem(f'{{self.lst_do[row]["title"]}}')
            elif col == 2:
                item =
QtWidgets.QTableWidgetItem(f'{{user['FIO']}}')
            elif col == 3:
                item = QtWidgets.QTableWidgetItem(
                    f'{"Сделано" if
self.lst_do[row]["status"] == True else "На выполнении"}')
                self.listWidget.setItem(row, col, item)
                self.textDescription.clear()
                self.current_matter_label.setText('Текущая задача: ')

def getId(self, title):
    """Получение id по названию"""
    for i in range(len(self.lst_do)):
        if self.lst_do[i]['title'] == title:
            return self.lst_do[i]['id']

def getDescription(self):
    """Отображение описание выбранного дела"""
    try:
        self.current_matter_label.setText('')
        self.textDescription.setText('')

        for i in range(len(self.lst_do)):
            title =
self.listWidget.item(self.listWidget.currentRow(), 1).text()
            if self.lst_do[i]['id'] == self.getId(title):

self.textDescription.setText(self.lst_do[i]['description'])
            title =
str(self.listWidget.item(self.listWidget.currentRow(), 1).text())
            self.current_matter_label.setText(f'Текущая задача:
{title}') if \
                self.listWidget.item(self.listWidget.currentRow(),
1).text() != 'Название' \
                else self.current_matter_label.setText('Текущая
задача: ')
    except:

```

```

        self.current_matter_label.setText('Текущая задача: ')

    def add(self):
        """Окно добавления"""
        self.msg_add.show(self.calendarWidget, self.user_id,
self.selected_proj, self.permission)

    def edit(self):
        """Окно редактирования """
        try:
            if self.listWidget.item(self.listWidget.currentRow(),
1).text() != 'Название':
                task = None
                for i in range(len(self.lst_do)):
                    title =
self.listWidget.item(self.listWidget.currentRow(), 1).text()
                    if self.lst_do[i]['id'] == self.getId(title):
                        task =
requests.get(f"{BASE_URL}/list_tasks_by_id/{self.lst_do[i]['id']}")
                        .json()
                        self.msg_edit.show(self.user_id, task,
self.selected_proj, self.permission)
        except:
            msg = QMessageBox(self)
            msg.setText('Выбери что то')
            msg.setWindowTitle("Ошибка")
            msg.exec()

    def rm_task(self):
        """ Удаление дела """
        for i in range(len(self.lst_do)):
            title =
self.listWidget.item(self.listWidget.currentRow(), 1).text()
            if self.lst_do[i]['id'] == self.getId(title):
                requests.delete(f"{BASE_URL}/tasks/{self.getId(title)}").json()
                self.renderList()

    def proj_user(self):
        """Проекты пользователя"""
        self.project_combobox.clear()
        query =
requests.get(f"{BASE_URL}/project_for_admin").json()
        if self.permission == 'admin' else requests.get(
            f"{BASE_URL}/project/{self.user_id}").json()
        if len(query) > 0:
            for i in query:
                self.project_combobox.addItem(i['name'])
                self.selected_proj = query[0]['id']
        else:
            self.project_combobox.addItem('Нет проектов')

```

```

def change_project(self):
    """Смена проекта"""
    query =
requests.get(f"{BASE_URL}/project_for_admin").json() if
self.permission == 'admin' else requests.get(
    f"{BASE_URL}/project/{self.user_id}").json()
    for i in query:
        if self.project_combobox.currentText() == i['name']:
            self.selected_proj = i['id']
            break
    self.renderList()

def chat_sh(self):
    """Показ чата"""
    self.ch_win.show(self.user_id, self.selected_proj)

def prof_sh(self):
    """Показ профиля"""
    self.prof.show(self.user_id)

def proj_sh(self):
    """Показ проектов"""
    self.proj.show(self.user_id, self.permission)

def notes_sh(self):
    """Показ уведомлений"""
    self.notes.show(self.user_id, self.selected_proj)

def exit(self):
    """Выход пользователя"""
    self.close()
    self.login.show()

def closeEvent(self, event):
    for widget in QApplication.topLevelWidgets():
        if widget != self:
            widget.close()

    super().closeEvent(event)

def reg(self):
    """Показ регистрации"""
    from reg import Reg
    self.reg_win = Reg()
    self.reg_win.show()

```

## Листинг 8. Код окна добавления задачи

```

from PyQt6 import QtCore, QtGui, QtWidgets

import requests
from PyQt6.QtWidgets import QMessageBox

```

```

from connection import BASE_URL

class Add:
    def execute(self):
        if self.titleLineEdit.text() != '' and
self.descEdit.toPlainText() != '':
            u_id = 0
            for i in self.query:
                if self.combobox.currentText() == i['FIO']:
                    u_id = i['id']
            if self.combobox.currentText() == 'Не найдено':
                msg = QMessageBox(self._winAdd)
                msg.setText('Ты не можешь добавить задачу этому
пользователю.\n'
                           'Проверь, есть ли у тебя люди в
проекте')
                msg.setWindowTitle("Ошибка")
                msg.exec()
            else:
                data = {
                    'date': str(self.dateEdit.date().toPyDate()),
                    'time':
str(self.timeEdit.time().currentTime().toPyTime().strftime('%H:%M:
%S')),
                    'title': self.titleLineEdit.text(),
                    'description':
str(self.descEdit.toPlainText()),
                    'status': 0 if self.user_id == u_id else 2,
                    'user_id': u_id,
                    'project_id': self.proj_id,
                }

                query = requests.post(f'{BASE_URL}/add',
json=data)

                if self.user_id != u_id:
                    data2 = {
                        'sender_id': self.user_id,
                        'task_id': query.json()['id'],
                    }
                    requests.post(f'{BASE_URL}/add2', json=data2)

                self.titleLineEdit.clear()
                self.descEdit.clear()
                self._winAdd.close()
            else:
                msg = QMessageBox(self._winAdd)
                msg.setText('Поля не могут быть пустыми')
                msg.setWindowTitle("Ошибка")
                msg.exec()

    def render_users(self):

```



```

        self.query =
requests.get(f"{BASE_URL}/list_proj_users/{self.proj_id}").json()
        self.combobox.clear()
        if len(self.query) > 0:
            for i in self.query:
                if i['id'] != self.user_id:
                    self.combobox.addItem(i['FIO'])
        else:
            self.combobox.addItem('Не найдено')

    def show(self, date: QtWidgets.QCalendarWidget, user_id,
proj_id, permission) -> None:
        self.user_id = user_id
        self.proj_id = proj_id
        self.dateEdit.setDate(date.selectedDate())
        self.timeEdit.setTime(self.timeEdit.time().currentTime())
        self.render_users()
        self._winAdd.show()

```

## Листинг 9. Код окна редактирования

```

import requests.models

from add import *

class Edit(Add):
    def execute(self):
        if self.permission == "admin":
            if self.titleLineEdit.text() != '':
                u_id = 0
                for i in self.query:
                    if self.combobox.currentText() == i['FIO']:
                        u_id = i['id']
                data = {
                    'date': str(self.dateEdit.date().toPyDate()),
                    'time': str(self.timeEdit.time().toPyTime()),
                    'title': self.titleLineEdit.text(),
                    'description':
str(self.descEdit.toPlainText()),
                    'status': 1 if self.checkBox.isChecked() else
0,
                    'user_id': u_id
                }
                if self.user_id != u_id:
                    data2 = {
                        'sender_id': self.user_id,
                        'task_id': self.task["id"],
                    }
                    data['status'] = 2
                    requests.post(f'{BASE_URL}/add2', json=data2)
                requests.put(f'{BASE_URL}/edit/{self.task["id"]}',
json=data)

```

```

        self._winAdd.close()
        self.titleLineEdit.clear()
        self.descEdit.clear()
    else:
        data = {
            'date': str(self.dateEdit.date().toPyDate()),
            'time': str(self.timeEdit.time().toPyTime()),
            'title': self.titleLineEdit.text(),
            'description': str(self.descEdit.toPlainText()),
            'status': 1 if self.checkBox.isChecked() else 0,
            'user_id': self.user_id
        }
        requests.put(f'{BASE_URL}/edit/{self.task["id"]}',
json=data)
        self._winAdd.close()

    def show(self, user_id, task, proj_id, permission) -> None:
        self.task = task
        self.user_id = user_id
        self.proj_id = proj_id
        self.permission = permission
        if self.permission == 'user':
            self.dateEdit.setDisabled(True)
            self.timeEdit.setDisabled(True)
            self.titleLineEdit.setDisabled(True)
            self.descEdit.setDisabled(True)
            self.combobox.close()
            self.for_label.close()

            date = task['date'].split('-')
            self.dateEdit.setDate(QDate(int(date[0]),
int(date[1]), int(date[2])))

        self.timeEdit.setTime(QTime.fromString(task['time']))
        self.titleLineEdit.setText(task['title'])
        self.titleFIRST = task['date']
        self.descEdit.setText(task['description'])
        self.render_users()
        self._winAdd.show()

```

## Листинг 10. Код окна уведомлений для работника

```

import requests
from PyQt6 import QtCore, QtWidgets
from project import Project
from connection import BASE_URL

class Note(Project):

```

```

    def show(self, user_id, proj_id=0):
        self.user_id = user_id
        self.proj_id = proj_id
        self.query =
requests.get(f"{BASE_URL}/list_proj_users/{self.proj_id}").json()

        self.fr =
requests.get(f"{BASE_URL}/get_name_for_notes/").json()
        self.render()

        self._proj_win.show()

    def accept(self):
        if self.listWidget.item(self.listWidget.currentRow(),
0).text() != 'Название':
            for i in self.notes:
                if
self.listWidget.item(self.listWidget.currentRow(), 0).text() ==
i['title']:

requests.put(f"{BASE_URL}/update_st/?task_id={i['id']}&st={0}").js
on()

                self.render()

```

#### Листинг 11. Код окна проектов

```

import requests
from PyQt6 import QtCore, QtWidgets, QtGui
from connection import BASE_URL

class Project:
    def for_permission(self):
        if self.permission == 'user':
            self.pushButton.close()
            self.pushButton_2.close()

    def render(self):
        query =
requests.get(f"{BASE_URL}/project_for_admin").json() if
self.permission == 'admin' else requests.get(
        f"{BASE_URL}/project/{self.user_id}").json()
        self.listWidget.clear()
        self.listWidget.setColumnCount(2)
        self.listWidget.setHorizontalHeaderLabels(['Название',
'Описание'])
        self.listWidget.setRowCount(len(query))
        for row in range(len(query)):
            for col in range(self.listWidget.columnCount()):
                item = None
                if col == 0:
                    item =
QtWidgets.QTableWidgetItem(f'{query[row]['name']}')

```

```

        elif col == 1:
            item =
QtWidgets.QTableWidgetItem(f'{query[row]['desc']}')
            self.listWidget.setItem(row, col, item)

    def show(self, user_id, permission):
        self.permission = permission
        self.user_id = user_id
        self.render()
        self._proj_win.show()

    def add(self):
        self.add_pr.show_(self.user_id)

    def remove_pr(self):
        title = self.listWidget.item(self.listWidget.currentRow(),
0).text()
        requests.delete(f'{BASE_URL}/project/{title}').json()
        self.render()

from add import Add

class Add_proj(Add):
    def __init__(self):
        Add.__init__(self)
        self.user_id = None

        self.dateEdit.close()
        self.timeEdit.close()
        self.date_label.close()
        self.time_label.close()
        self.for_label.close()
        self.combobox.close()

        _translate = QtCore.QCoreApplication.translate

        self._winAdd.setWindowTitle(_translate("Form", "Добавить
проект"))
        self.pushButton.clicked.connect(self.add_pr)

    def add_pr(self):
        """ Удаление дела """
        data = {
            'name': self.titleLineEdit.text(),
            'desc': self.descEdit.toPlainText(),
        }
        requests.post(f'{BASE_URL}/add_project/{self.user_id}',
json=data).json()
        self.titleLineEdit.clear()
        self.descEdit.clear()

```

```

        self._winAdd.close()

    def show_(self, user_id):
        self.user_id = user_id
        self._winAdd.show()

```

## Листинг 12. Код окна чата

```

from datetime import datetime
import requests
from PyQt6 import QtCore, QtWidgets, QtGui
from connection import BASE_URL

class Chat:

    def show(self, u_id, p_id):
        self.user_id = u_id
        self.proj_id = p_id
        self.query =
requests.get(f"{BASE_URL}/list_proj_users/{self.proj_id}").json()
        self.combo.clear()
        if len(self.query) > 0:
            for i in self.query:
                if i['id'] != self.user_id:
                    self.combo.addItem(i['FIO'])
        else:
            self.combo.addItem('Не найдено')

        self.render_ch()
        self._winAdd.show()

    def send(self):
        r_id = 0
        for i in self.query:
            if self.combo.currentText() == i['FIO']:
                r_id = i['id']

        data = {
            'date': str(QtCore.QDate.currentDate().toPyDate()),
            'time':
str(QtCore.QTime.currentTime().toPyTime().strftime('%H:%M:%S')),
            'sender_id': self.user_id,
            'reciever_id': r_id,
            'message': str(self.textEdit.toPlainText())
        }
        query = requests.post(f'{BASE_URL}/chat', json=data)
        self.textEdit.clear()
        self.render_ch()

    def get_sender(self, id):
        for i in self.query:
            if i['id'] == id:
                return i['FIO']

```

```

def render_ch(self):
    self.listWidget.clear()
    r_id = 0
    for i in self.query:
        if self.combo.currentText() == i['FIO']:
            r_id = i['id']
    query =
requests.get(f'{BASE_URL}/history_chat?sender_id={self.user_id}&re
ciever_id={r_id}')

    self.sort_hist = sorted(query.json(), key=lambda x:
(x['date'], x['time']))

    for i in self.sort_hist:
        time = i['time'].split(':')
        date = datetime.strptime(i['date'], '%Y-%m-
%d').date().strftime("%d.%m.%Y")
        self.listWidget.addItem(f'{date} {time[0]}:{time[1]}
{self.get_sender(i['sender_id'])}\n {i['message']}')

```

### Листинг 13. Код окна регистрации

```

from PyQt6 import QtCore, QtGui, QtWidgets
from PyQt6.QtWidgets import QWidget
import requests

from connection import BASE_URL

class Reg(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowIcon(QtGui.QIcon('resources/ico/cat.ico'))
        self.pr = []
        self.mw = None
        self.resize(295, 192)
        self.setFixedSize(295, 192)
        self.verticalLayout = QtWidgets.QVBoxLayout(self)
        self.verticalLayout.setContentsMargins(0, 0, 0, 0)
        self.reg_label = QtWidgets.QLabel(parent=self)
        self.verticalLayout.addWidget(self.reg_label, 0,
QtCore.Qt.AlignmentFlag.AlignHCenter)
        self.fio_label = QtWidgets.QLineEdit(parent=self)
        self.verticalLayout.addWidget(self.fio_label)
        self.login_edit = QtWidgets.QLineEdit(parent=self)
        self.verticalLayout.addWidget(self.login_edit)
        self.password_edit = QtWidgets.QLineEdit(parent=self)
        self.verticalLayout.addWidget(self.password_edit)
        self.combo = QtWidgets.QComboBox(parent=self)
        self.verticalLayout.addWidget(self.combo)

        self.reg_button = QtWidgets.QPushButton(parent=self)

```

```

self.verticalLayout.addWidget(self.reg_button)
self.combo.setFixedHeight(30)

_translate = QtCore.QCoreApplication.translate
self.setWindowTitle(_translate("Form", "Регистрация"))
self.reg_label.setText(_translate("Form", "Регистрация"))
self.fio_label.setPlaceholderText(_translate("Form",
"ФИО"))
self.login_edit.setPlaceholderText(_translate("Form",
"Логин"))
self.password_edit.setPlaceholderText(_translate("Form",
"Пароль"))
self.reg_button.setText(_translate("Form", "Регистрация"))

self.reg_button.clicked.connect(self.register)

self.render_pr()

def render_pr(self):
    self.pr = requests.get(f'{BASE_URL}/list_projects').json()
    for i in self.pr:
        self.combo.addItem(str(i['name']))

def register(self) -> bool:
    try:
        data = {
            'FIO': self.fio_label.text(),
            'login': str(self.login_edit.text()),
            'password': self.password_edit.text(),
            'root': "user",
        }
        q = requests.post(f'{BASE_URL}/add_user',
json=data).json()

        pr_id = 0
        for i in self.pr:
            if self.combo.currentText() == i['name']:
                pr_id = i['id']

        data2 = {
            'user_id': q['id'],
            'project_id': pr_id
        }
        requests.post(f'{BASE_URL}/user_to_project',
json=data2)

        self.close()
        return True
    except:
        print('error')
        return False

```

## Листинг 14. Код окна профиля

```
import requests
from PyQt6 import QtCore, QtWidgets, QtGui
from connection import BASE_URL

class Profile:

    def edit(self):
        self.lineEdit_3.setDisabled(False)
        self.lineEdit.setDisabled(False)
        self.lineEdit_2.setDisabled(False)
        self.ok_button.show()
        self.change_button.close()

    def ok(self):
        data = {
            'FIO': self.lineEdit_2.text(),
            'login': self.lineEdit.text(),
            'password': self.lineEdit_3.text(),
            'id': self.query['id']
        }
        requests.put(f'{BASE_URL}/update_profile', json=data)
        self.lineEdit_3.setDisabled(True)
        self.lineEdit.setDisabled(True)
        self.lineEdit_2.setDisabled(True)
        self._prof_win.close()
        self.change_button.show()
        self.ok_button.close()

    def render(self):
        self.query =
requests.get(f'{BASE_URL}/profile/{self.user_id}').json()
        fio, login, password = str(self.query['FIO']),
str(self.query['login']), str(self.query['password'])
        self.lineEdit_2.setText(fio)
        self.lineEdit.setText(login)
        self.lineEdit_3.setText(password)

    def show(self, user_id):
        self.user_id = user_id
        self.render()
        self._prof_win.show()
```