МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра Прикладная математика и информатика
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)
Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка программного обеспечения чата для внутренней коммуникации сотрудников предприятия»

Обучающийся Д.В. Голуб _____
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

Тема бакалаврской работы - «Разработка программного обеспечения чата для внутренней коммуникации сотрудников предприятия». В рамках темы были рассмотрены основные тенденции современного рынка программного обеспечения и существующие разработки.

Структура работы представлена введением, тремя главами, девятью подглавами, заключением, списком использованной литературы и одним приложением.

В первой главе формируются требования и постановка задачи на разработку программного обеспечения чата для автоматизации внутренней коммуникации сотрудников предприятия.

Во второй главе проектируется программное обеспечение чата для автоматизации внутренней коммуникации сотрудников предприятия.

В третьей главе описан процесс реализации и тестирования программного обеспечения чата для автоматизации внутренней коммуникации сотрудников предприятия.

В заключении описываются результаты выполнения выпускной квалификационной работы.

В работе содержатся 40 рисунков, 3 таблицы, одно приложение, 20 источников. Всего в работе 88 страниц.

Оглавление

Введение

Коммуникация – это взаимодействие между субъктами. В качестве субъектов могут выступать люди, животные. Для совершения коммуникации, субъектам, как правило, нужно находиться рядом друг с другом. Также возможен вариант коммуникации на расстоянии, он существует с давних времен.

Раньше, для коммуникации на расстоянии, люди писали письма. Сегодня, данный подход уже утрачивает популярность. В век современных технологий, преобладает общение с использованием средств. Сначала разговоры по телефону, затем появился интернет и появились чаты.

Выбор темы «Разработка программного обеспечения чата для внутренней коммуникации сотрудников предприятия» бакалаврской работы обусловлен актуальностью данной проблемы. Сегодня существует множество решений, которые хоть и пользуются преимущественной удобностью, но обладают избыточным, лишним для делового общения функционалом. Также есть другие приложения, но которые требуют плату за свой функционал.

Объектом исследования бакалаврской работы является процесс взаимодействия сотрудников внутри предприятия.

Предмет - автоматизация процесса взаимодействия сотрудников внутри предприятия.

Цель выпускной квалификационной работы – разработка программного обеспечения чата для внутренней коммуникации сотруднокв предприятия для обеспечения удобного взаимодействия сотрудников внутри организации. Программный продукт будет ограничен локальной сетью, что позволит сохранить конфиденциальность передаваемой информации. Кроме того, функционал программы будет ограниченным и простым, без лишних возможностей, что не позволит неквалифицированным пользователям подхватить вредоносное программное обеспечение.

Для достижения данной цели необходимо решить следующие задачи:

- сформировать общую картину, чтобы определить, как будет выглядеть и работать будущий программный продукт;
- выбрать набор необходимых инструментов для разработки программного обеспечения;
- спроектировать базу данных для хранения информации о пользователях, чатах и сообщениях;
- написать приложение;
- провести тестирование готового продукта в стрессовых условиях и внести изменения в случае выявления необработанных ошибок.

В результате выполнения данной работы, планируется получить готовое решение для оптимизации бизнес-процесса внутренней коммуникации сотрудников предприятия. Коллеги смогут решать многие рабочие задачи, не покидая при этом свое рабочее место. Сами рабочие процессы станут более ускоренными и более контролируемыми. Повысится качество условий труда и выполняемых задач.

Данная работа состоит из введения, трех глав, заключения и списка используемой литературы и источников.

Первая глава посвящена формированию требований и постановке задачи на разработку ПО для автоматизации внутренней коммуникации сотрудников предприятия.

Вторая глава посвящена проектированию ПО для автоматизации внутренней коммуникации сотрудников предприятия.

В третьей главе описан процесс реализации и тестирования ПО для автоматизации внутренней коммуникации сотрудников предприятия.

В заключении описываются результаты выполнения выпускной квалификационной работы.

Бакалаврская работа состоит из 59 страниц текста, 40 рисунков, 3 таблиц, одного приложения и 20 источников.

Глава 1 Постановка задачи на разработку программного обеспечения чата внутренней коммуникации сотрудников предприятия

1.1 Описание задачи автоматизации внутренней коммуникации сотрудников предприятия

Компания «Дельта-софт» предоставляет комплексные услуги по внедрению и сопровождению программ на платформе 1С: Предприятие. Написанное программное обеспечение обеспечивает стабильную работу бизнеса.

Организация работает с 2007 года, за это время было осуществлено более 100 успешных внедрений программ.

В услуги внедрения входит:

- предварительное обследование: анализ бизнес-процессов предприятия;
- планирование: выбор оптимального решения от 1С и составление плана по наложению бизнес-процессов на предлагаемую программу;
- установка и внедрение: установка программы, доработка существующих или разработка новых модулей, обучение персонала;
- послевнедренческая поддержка: консультация пользователей по возникающим вопросам, установка свежих обновлений на внедренные программы.

Организационная структура предприятия представлена на рисунке 1:

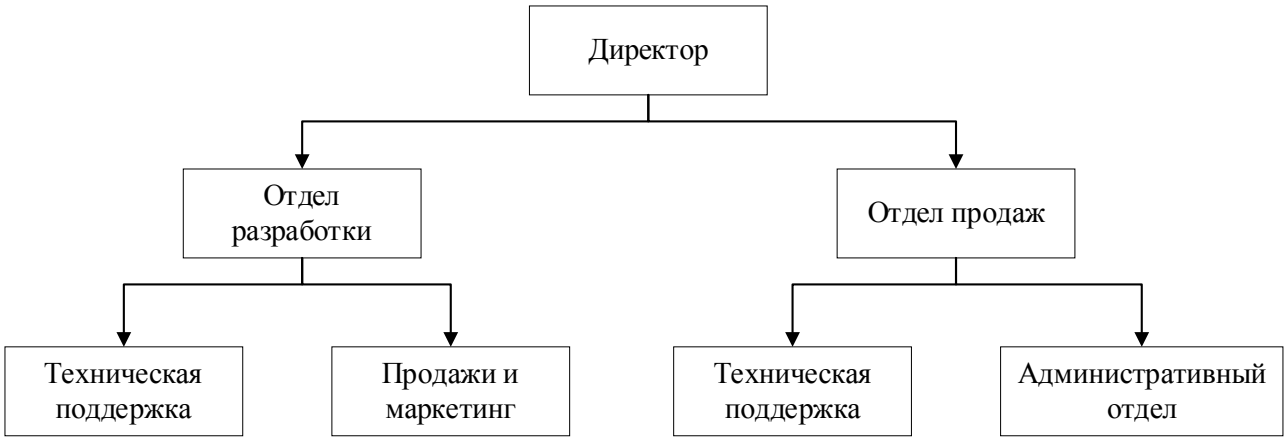

Рисунок 1 – Организационная структура ООО «Дельта-софт»

Руководство в лице Генерального директора занпимается управлением рабочими процессами и принимает стратегические решения.

Отдел разработки отвечает за разработку программного обеспечения, а также занимается ее последующим сопровождением.

Отдел продаж занимается продвижением и продажей разработанного программного обеспечения, а также работает с заказчиками и партнерами. Кроме того, этот отдел отвечает за анализ рынка и конкурентной среды.

Техническая поддержка, собственно, предоставляет техническую поддержку заказчикам и клиентам, что подразумевает собой решение проблем и вопросов пользователей по поводу разработанного программного продукта. Для реализации последнего, имеет обратную связь с отделом разработки для устранения внештатных ситуаций и ошибок.

Административный отдел отвечает за управление организационными процессами и ресурсами предприятия ООО «Дельта-Софт». Помимо этого, финансовое планирование и управление бухгалтерией входит в прямые обязанности этого отдела. Кроме перечисленного, административный отдел также занимается вопросами кадрового характера, касаемо найма, увольнения, обучения, развития и повышения квалификации сотрудников.

Решение, затрагиваемое в данной работе, должно оказать влияние на бизнес-процесс взаимодействия между сотрудниками разных отделов. Для

наглядной демонстрации необходимости автоматизации данного процесса была построена функциональная модель.

Важно понимать, что потребность в общении между сотрудниками внутри одного подразделения имеет большое значение для повышения производительности труда. Проблема в том, что сотрудники разных отделов и подразделений вынуждены покидать рабочие места для передачи какой-либо информации, что не всегда удобно и возможно. Практически в любом предприятии процесс передачи информации осуществляется регулярно.

Устранение отсутствия автоматизации – это очень важная и актуальная проблема, требующая скорейшего решения. Разрабатываемое программное обеспечения чата для внутренней коммуникации сотрудников предприятия позволит:

- снизить затраты времени: когда в компании большое количество сотрудников, то для коммуникации приходится тратить больше времени и усилий;
- повысить эффективность передачи информации: сотрудники могут не получить необходимую информацию вовремя из-за недостаточной организации;
- повысить прозрачность: без автоматизированного приложения чата, возможно, будет затруднительно отслеживать и контролировать обмен информацией между сотрудниками и подразделениями;
- улучшить связь между отделами: нехватка современных средств для коммуникации может привести к тому, что между различными отделами будет недостаточная связь, а также ухудшится координация работы между ними.

В конечном итоге, приложение поможет предприятию решить множество проблем и повысить проивзодительность рабочего процесса. Улучшится настроение сотрудников.

1.2 Обзор и анализ аналогов программного обеспечения чата для автоматизации внутренней коммуникации сотрудников предприятия

Для достижения лучшего результата следует провести анализ существующих решений чата. Прежде всего, это действие позволит определить основной функционал и рассмотреть преимущества, которые можно внедрить в свой программный продукт. Основные критерии приложений корпоративного чата:

- простота использования,
- фокус на основной функциональности,
- гибкость и настраиваемость,
- безопасность данных,
- совместимость с различными платформами.

По каждому критерию выделены актуальные существующие разработки:

а) простота использования:

1) Slack. Slack – корпоративный мессенджер. Был создан в 2013 году компанией Tiny Speck, которая впоследствии стала называться Slack Technologies [14]. У Slack есть важное преимущество, а именно создание каналов для разных команд, что упрощает рабочий процесс. Благодаря этому, Slack стал основным средством общения для многих компаний [17]. Однако, он не обладает русской локализацией. Помимо отсутствия русского языка, Slack является платным, если пользователю нужны групповые чаты и хранение истории общения дольше 90 дней.
2) Discord. Discord - мощная и многофункциональная онлайн-платформа, мессенджер, социальная сеть, площадка для форумов и ботов - в общем, всё и сразу. Он известен своей простотой, производительностью и особенно популярен среди любителей видеоигр. Интерфейс Discord интуитивно понятен, что делает его

доступным для неквалифицированных пользователей. Имеется возможность создания серверов и каналов для общения по разным темам, что делает его удобным для общения в группах и сообществах. Но, как и упоминалась ранее, Discord скорее предназначен для развлечений и обладает излишним для корпоративных целей функционалом.

б) фокус на основной функциональности:

1) WhatsApp. WhatsApp - это бесплатное приложение для простого, безопасного и надёжного обмена сообщениями и совершения звонков [20]. Помимо простого обмена сообщениями, имеет возможность обмена голосовыми и видео сообщениями. Он для многих пользователей является основным средством для общения. Однако, WhatsApp привязывается к телефону и авторизоваться можно только с ним.
2) Telegram. В основе Telegram лежит кроссплатформенная система обмена мгновенными сообщениями. Она включает в себя функции голосовых, видео и текстовых сообщений [18]. Вместе с основным функционалом для мессенджера, Telegram предлагает ряд дополнительных возможностей: создание каналов, создание и использование ботов, добавление и обмен стикерами и т.д.. Но, Telegram предоставляет слишком много свободы действий. Много каналов с непотребным контентом. Мошенничество и фишинг часто практикуются в Telegram.

в) гибкость и настраиваемость:

1) Microsoft Teams. Microsoft Teams – корпоративный вариант приложения для общения от Microsoft. Предоставляет возможность удаленного общения с коллегами [15]. Из минусов стоит отметить, что данное решение входит в набор Microsoft 365 и стоит денег, тем более сегодня пользоваться сервисами Microsoft и приобрести их продукцию на территории РФ невозможно.

2) Google Chat. Google Chat – коммуникационный сервис от Google. Он удобен для продуктивной совместной работы, позволяет пользователям создавать чат-группы, настраивать уведомления и интегрировать сервисы Google Workspace. Это удобное средство общения для пользователей, активно пользующимися сервисами Google [13]. Однако, есть минус - оно обладает не самым понятным и приятным интерфейсом

г) безопасность данных:

1) Wire. Wire называется «самым безопасным мессенджером». Он уделяет большое внимание безопасности и конфиденциальности данных [16]. Самые требовательные организации в мире полагаются на Wire с его постоянно включенным шифрованием для чатов, голосовых и видеоконференций. Главные минусы – стоимость и отсутствие русского языка. Несмотря на open source, бесплатно использовать Wire получится только 30 дней и то, добавить можно только 1 контакт
2) Threema. Threema – вариант безопасного мессенджера от швейцарских разработчиков. Благодаря последовательному сквозному шифрованию никто, кроме предполагаемого получателя, не может прочитать передаваемые сообщения. Даже сама Threema в качестве оператора сервиса не может этого сделать. Threema была создана с нуля с учетом защиты данных и конфиденциальности (Privacy by Design), поэтому при использовании сервиса генерируется как можно меньше метаданных [19]. Однако, доступ к личной переписке все же есть, а значит и анонимности не существует. Также, отсутствует русский язык.

д) адаптивность:

1) Zoom. Видеовстречи, командный чат, VoIP-телефон, вебинары, доска объявлений, контакт-центр и мероприятия – это все про Zoom [8]. Он часто используется в образовательных учреждениях, однако,

несмотря на такое обилие преимуществ - он платный, имеет проблемы с защитой, а также лимит сессии всего 40 минут.

2) Skype. Миллионы частных лиц и предприятий используют Skype. Он позволяет проводить видеоконференции и чат, а еще совершать звонки на мобильные телефоны прямо из приложения. Skype доступен на всех современных платформах, но есть и существенные недостатки, как плохое качество связи, оптимизация и нестабильная работа приложения.

Анализ существующих разработок показал, что есть множество приложений, способных предоставить пользователю возможность чата. У каждого отдельного программного продукта есть свои преимущества, на которые стоит обратить внимание при разработке собственного решения. Но также имеются и свои недостатки, которые тоже не стоит оставлять без внимания.

Существует немаловажный фактор доступности этих программных решений на территории России. На сегодняшний день – это проблема особенно актуальна. Многие крупные корпорации приостановили работу в РФ, что сильно сужает диапазон доступного программного обеспечения для использования.

1.3 Разработка требований к программному обеспечению чата для автоматизации внутренней коммуникации сотрудников предприятия

Бизнес-цель – это конечный результат, который ожидается получить в результате разработки программного обеспечения. Они бывают долгосрочными и краткосрочными. Бизнес-цели являются важной состовляющей бизнес-стратегии и их следует выявить для достижения успеха на выходе.

Были выявлены следующие бизнес-цели:

- улучшение коммуникации сотрудников,
- более эффективное управление рабочими процессами,
- экономия времени и ресурсов,
- повышение ментального настроения сотрудников.

С целью эффективного взаимодействия внутри компании, реализация программного обеспечения для чата строится на ряде ключевых требований, в том числе:

Функциональные требования:

Прежде всего, безусловной является реализация системы аутентификации пользователей и механизмов авторизации, которая позволяет настроить градации доступа к определённым функциям в зависимости от статуса учётной записи. Чтобы обеспечить беспрерывный обмен информацией и удобную координацию действий команд, предусмотрено наличие как групповых чатов для коллективного ведения проектов и проведения межподразделенческих собраний, так и приватных бесед для индивидуального общения.

Следующий неотъемлемый аспект – это модуль управления проектами, встроенный непосредственно в приложение, который дает возможность отслеживания заданий и проектов, а также осуществляет коммуникацию в рамках конкретных задач.

Охрана конфиденциальной информации, обменом которой обеспечивает программа, также превалирует. Меры по обеспечению безопасности данных и их защиты от несанкционированных вмешательств и утечек становятся краеугольным камнем защиты интеллектуальной собственности фирмы.

Наконец, разрабатываемая система чата обязана быть адаптирована к многоплатформенным рабочим средам, включая, но не ограничиваясь, современными разновидностями операционных систем таких как Windows, Linux, Mac OS X, а также мобильных iOS и Android, обеспечивая универсальность и широкую доступность ресурса для всех сотрудников организации.

Подобный подход разрабатываемого функционирования значительно повышает эффективность внутренних процессов и открывает новые горизонты для коммуникативной деятельности предприятия.

Нефункциональные требования:

Системное приложение предполагает стойкость в контексте многопользовательской среды, обеспечивая быстродействие даже при увеличившемся до сотен числе агентов-пользователей.

Предвидится, что инструмент будет обладать гибкостью для расширения, что позволит справиться с нарастающим потоком информации и растущей аудиторией.

Ради исключения сбоев необходимо программное решение, которое инициирует обнаружение и последующее устранение несоответствий, тем самым заложив основу для повышения устойчивости к ошибкам.

Интерфейсные требования:

Чат-интерфейс, предназначенный для использования персоналом с различным уровнем технического знания, обязан отличаться лаконичностью и интуитивной ясностью. Обязательным условием его конструкции является выбор шрифтов и цветовой палитры, не вызывающих дискомфорт у пользователей. Текст и фон приложения должны быть оформлены таким образом, чтобы гарантировать легкость чтения без напряжения зрения, обеспечивая при этом визуальное оптимальность среды общения для лиц не владеющих специализированными техническими знаниями.

Для достижения задачи по разработке приложения, автоматизирующего внутренние информационные потоки предприятия, становится критической задачей определение объема необходимых работ. Это становится возможным после установления бизнес-целей и спецификий, предъявляемых к информационным технологиям на стадии планирования проекта. В итоге, возникает возможность реализации программного решения, ориентированного на фасилитацию внутренних коммуникационных процессов между сотрудниками компании.

Вывод по первой главе:

В области создания программного инструмента, задуманного для межличностной переписки сотрудников компании, была проведена предварительная подготовка, включающая детектирование и анализ предшествующих программных решений. Построенные на фундаменте полученных данных, спецификации и критерии стали отправным пунктом для последующей разработки уникального программного продукта, чьей миссией является оптимизация делового общения внутри корпоративного пространства.

Глава 2 Проектирование программного обеспечения чата для внутренней коммуникации сотрудников предприятия

2.1 Выбор технологии логического моделирования

При применении логического моделирования учитывается взаимодействие приложения и их состояний. Симуляция поведения системы включает детали о роли пользователей, характеристики сущностей, а также способы обработки данных. Используя язык моделирования UML, конструируются различные диаграммы - это может быть изображение последовательности операций, вариативность использования, структура классов и компонентно-модульная архитектура. При разработке этих диаграмм применяются CASE-технологии для повышения эффективности и точности модели. Базы данных, содержащие информацию о ключевых атрибутах системы, также подлежат детальному рассмотрению в процессе создания модели. Моделирование в области логических структур использует математический аппарат и алгебру логики для формального представления процессов.

Унифицированный язык моделирования (UML) – это семейство грфических нотаций, в основе которого лежит единая метамодель. Он помогает в описании и проектировании программных систем, в особенности систем, построенных с использованием объектно ориентированных (ОО) технологий [11, с. 27].

Выбор подходящего программного обеспечения для логического моделирования оказывается критическим шагом в визуальном представлении данных. После тщательного сравнительного анализа различных доступных вариантов, было определено, что для наших целей максимально подходит Visio, разработанное компанией Microsoft – инструмент, обладающий всеми необходимыми функциями и шаблонами для эффективного создания

логических моделей. Этот выбор продиктован обилием удобных функций в Visio: презентабельные шаблоны, обширный каталог инструментария, интуитивно понятный интерфейс на русском языке, возможности сохранения созданных диаграмм на локальный носитель и простое их экспортирование через копирование в графическом виде для последующего включения в текстовые документы формата doc.

2.2 Логическое моделирование программного обеспечения чата для автоматизации внутренней коммуникации сотрудников предприятия

Проектирование инструмента коммуникации внутри компании требует организации взаимодействия заинтересованных сторон этой системы. В этом контексте, имеет место быть интеграция возможностей управления проектами и задачами в чат-платформу. Развитие такой платформы дает возможность наладить более продуктивную корреспонденцию между сотрудниками.

Чтобы понять динамику взаимодействия пользователей с программным обеспечением для обмена сообщениями, необходимо составить диаграмму вариантов использования. Данная схема должна визуализировать основных пользователей чата. В представленной диаграмме будут освещены их функции, полномочия и назначенные цели, раскрывая тем самым иерархию в контексте программируемой среды общения.

На диаграмме вариантов использования были выделены основные функции главных участников процесса:

- пользователь;
- системный администратор;
- программист.

Диграмма прецендентов изображена на рисунке 2:

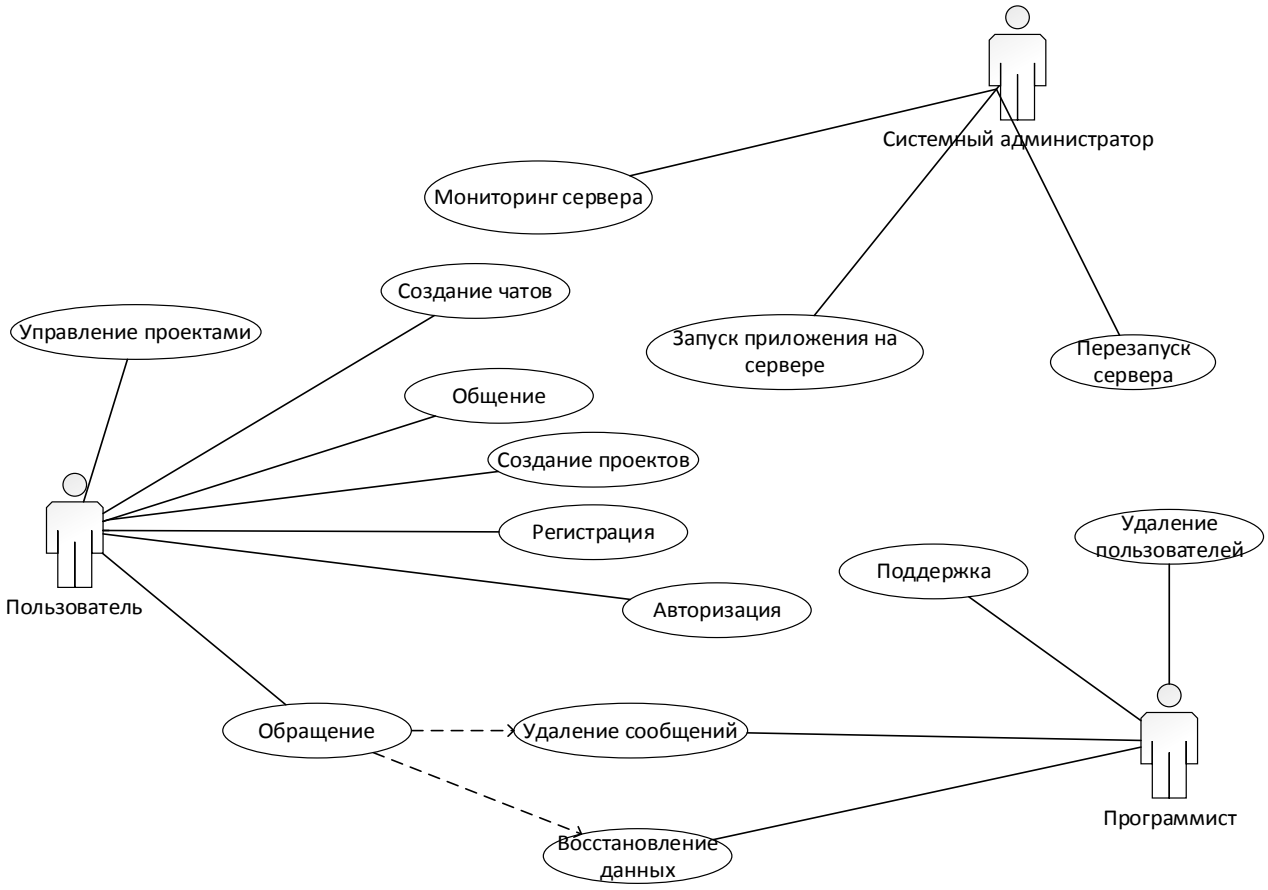

Рисунок 2 – Диаграмма вариантов использования

У каждого из приведенных лиц есть свои рабочие задачи и возможности. Необходимо распределить роли и обязанности по работе с программой для комфортного процесса работы с ней, а также поддержания ее стабильной функциональности.

Для каждого из акторов, принимающих непосредственное участие в процессе работы с разработанным программным продуктом, нужно определить роли и задачи, которые будут выполняться в процессе взаимодействия с приложением чата. Эти задачи были проинициализированы подробно и описаны в таблице 1:

Таблица 1 – Роли и выполняемые обязанности участников работы с программой

Участник	Задача	Описание
1	2	3
Пользователь	Авторизация	Авторизация в приложении
	Регистрация	Регистрация в приложении
	Создание чатов	Создание пользовательских чатов
	Общение	Общение с другими пользователями в созданных чатах
	Создание проектов	Создание проектов для управления своими задачами
	Управление проектами	Изменение состояний проекта, назначение участников проекта
Программист	Поддержка	Сопровождение и обновление программы
	Удаление пользователей	Удаление пользователя в случае необходимости
	Удаление сообщений	Удаление сообщений по просьбе пользователя или необходимости
	Восстановление данных	Восстановление доступа к аккаунту по просьбе пользователя
Системный администратор	Запуск приложения на сервере	Запуск приложения на локальном сервере предприятия
	Мониторинг сервера	Остлеживание производительности сервера приложения и потребляемых ресурсов
	Перезапуск сервера	Перезапуск сервера приложения в случае необходимости, сбоев или перегрузки

Взаимодействие через чат с участниками проекта обеспечивает эффективность рабочего процесса. С целью иллюстрации функционирования разработанного приложения для общения и координации деятельности сотрудников, была создана диаграмма последовательности.

Диаграмма последовательности описывает жизненный цикл объекта от начала работы пользователя с приложением, до завершения взаимодействия и выхода из программы. Такие диаграммы часто используются разработчиками, ведь в них можно подробно и, в то же время, наглядно описать взаимодействие пользователя, а также порядок действий во время работы с программым обеспечением.

Диаграмма последовательности, представляющая работу приложения чата для внутренней коммуникации сотрудников предприятия изображена на рисунке 3.

После успешного процесса идентификации, пользовательский интерфейс направляет субъекта на главную страницу веб-ресурса. Здесь он позволяет, в дальнейшем, пользователю осуществлять переходы к разнообразным функциональным секторам, в частности, к модулю корпоративного общения либо сфере администрирования проектных активностей. В рамках сегмента управления проектами, он рассчитывает на представление ряда проектов, каждый из которых обладает индивидуально определённой стадией реализации.

Указанный пользователь наделён полномочиями модификации состояний данных проектов. К тому же, в пределах каждого проекта заложен специфичный чат, предмет доступа к которому ограничен кругом его участников. В контексте страницы обмена сообщениями, обозначается возможность уединённого взаимодействия пользователя с другими индивидами платформы.

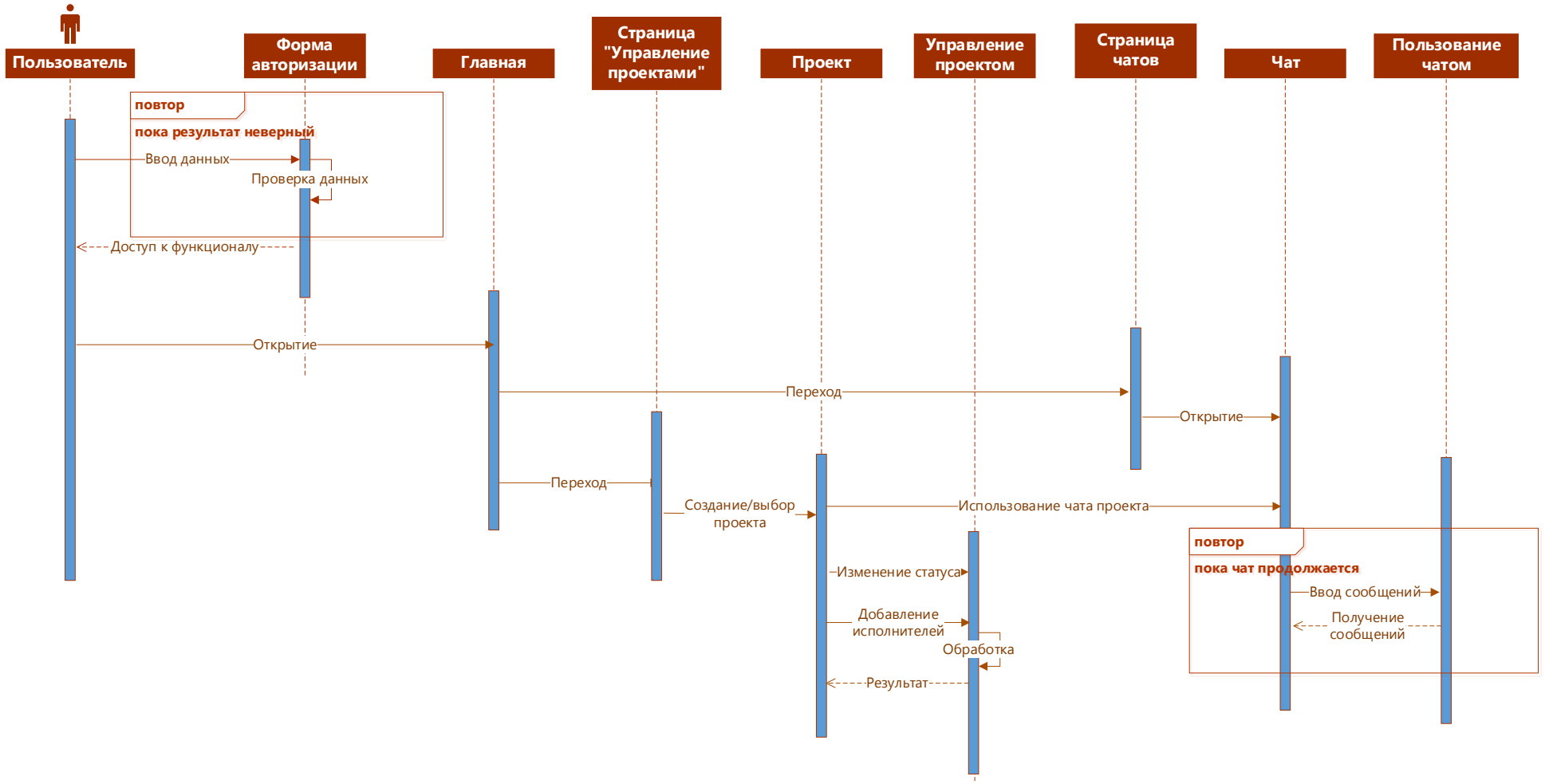

Рисунок 3 – Диаграмма последовательности

Концептуальная основа базы данных для синхронизации коммуникаций в чате обязательно должна отражать четыре важные категории информации: данные о пользователях, подробности по проектам, метаданные чатов и детали сообщений, включенные в них. Спектр содержания базы данных требует тщательно проработанного дизайна, что подразумевает разработку диаграммы сущностей, которая будет исполнена в строгом соответствии с заданными параметрами. Подтверждение данной структуры, служащей графическим представлением архитектуры данных, изображено на рисунке 4. Приведенный функционал чата позволит упростить общение между сотрудниками внутри предприятия, а руководителям обеспечит удобное управление проектами. На программиста(ов) ложится задача поддержки и обновления приложения. Системные администраторы ответственны за работу сервера, как сервера сокетов, так и сервера веб-приложения. Программное обеспечение чата должно выйти достаточно простым, понятным и удобным в использовании.

Основной функционал программы системы включает в себя:

- регистрация и авторизация пользователей: авторизированные пользователи могут получить доступ к функционалу приложения, а пользователи без учетной записи могут создать ее;
- отправка текстовых сообщений: внутри чатов имеется возможность обмена текстовыми сообщениями;
- создание чатов: любой пользователь может выбрать собеседника и создать новый чат с ним;
- просмотр и архивация истории сообщений: все чаты и история переписок хранится в БД и пользователи могут видеть свои чаты;
- создание и управление проектами: пользователи смогут создавать проекты для облегчения рабочего процесса, смогут обновлять статус проекта, добавлять новых участников;
- проектные чаты: у каждого проекта есть свой чат, общаться в котором и решать задачи могут только участники выделенного проекта.

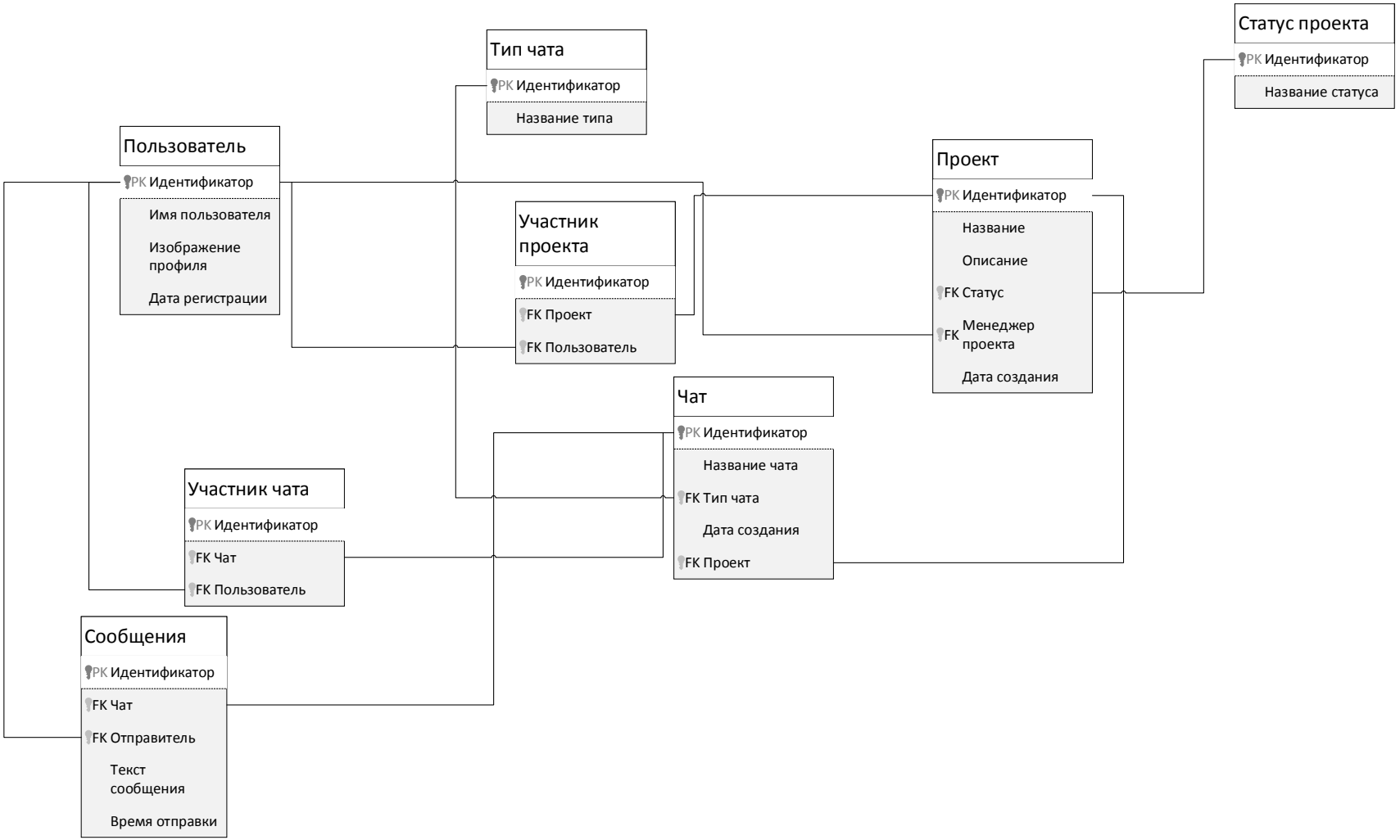

Рисунок 4 – Диаграмма сущностей

Программное обеспечение имеет как привычный для чатов функционал, так и возможность управления проектами, что выделяет его на фоне существующих разработок. В конечном итоге, пользователи получат качественный программный продукт.

2.3 Моделирование данных программного обеспечения

С целью хранения информации, связанной с программным продуктом, определено применение СУБД PostgreSQL. Проявление структуры этой базы данных воплощено в ERD, которую визуализировал pgAdmin, оперируя с дефинированными структурами таблиц. ERD диаграмма иллюстрирована на рисунке 5.

В рамках реализации базы данных, состоящей из многоуровневых отношений, было разработано несколько структур данных. Реляционная таблица «Пользователи» удостоверяет сущности с уникальными идентификаторами (PK) и сопряжёнными атрибутами: именем (username), изображением профиля (profilepic) и датой регистрации (registrationdate). Аналогично сформированы данные в таблице «Чаты», содержащей атрибуты уникального ключа (PK), наименования (title), а также связанных внешних ключей (FK) к типам чатов (type) и проектам (projects).

Проектирование предусматривает таблицу «Типы чатов», где каждый тип идентифицируется уникальным кодом и соответствующим описанием (title). Информационная связь между участниками и чатами отражена в таблице «Участники чата» с ключами, ссылками на данные пользователей (userid) и чаты (chatid). В отношении «Сообщения» фиксированы атрибуты для уникального кода сообщения (id), ссылок на чат (chatid) и отправителя (senderid), вместе с текстом сообщения (message) и временной отметкой (timestamp).

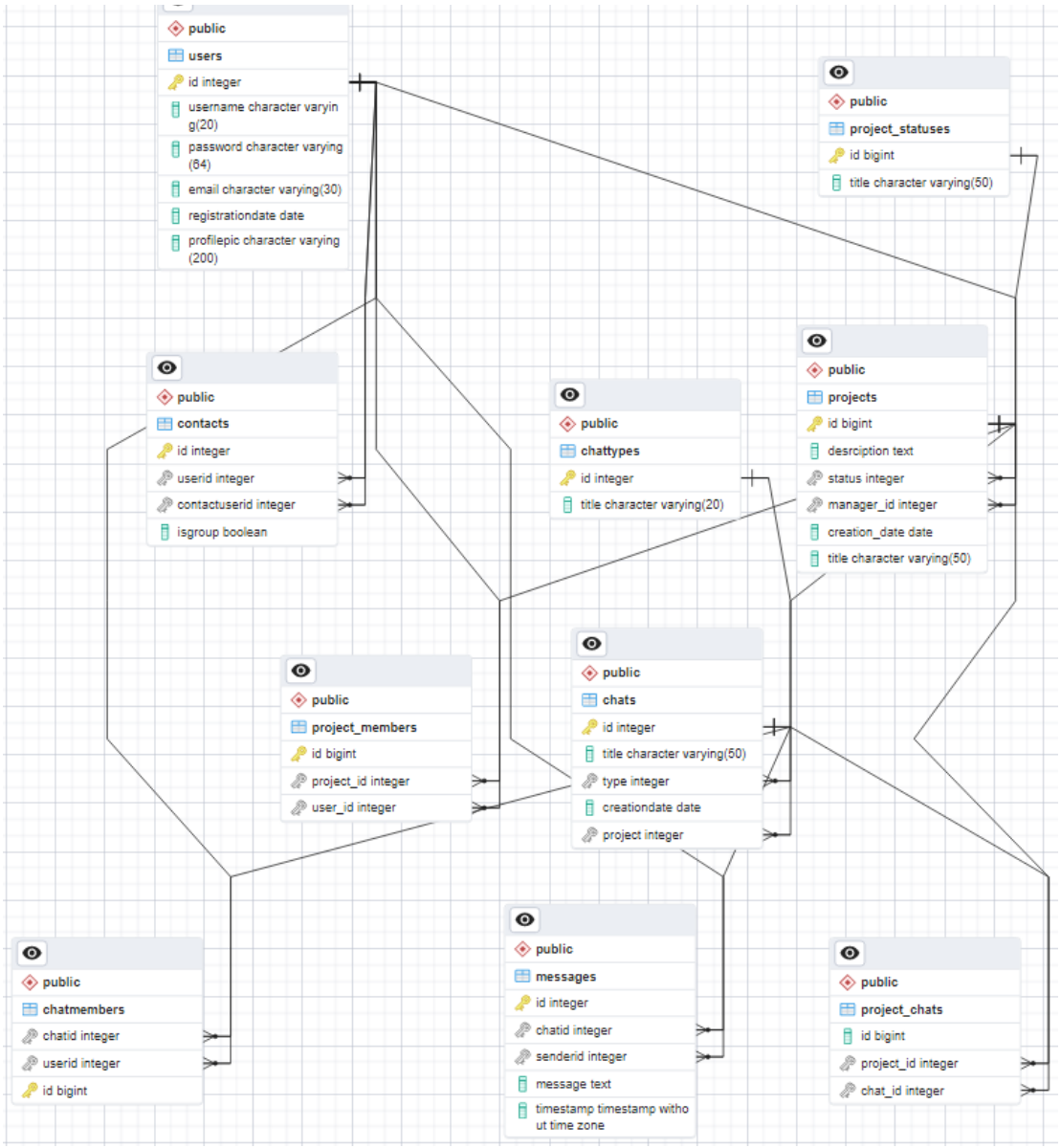

Рисунок 5 – ERD модель базы данных приложения

Касательно проектов, база включает таблицу «Проекты» с полями для уникального кода проекта (id), подробного описания (description), статуса в виде внешней ссылки (status), идентификатора руководителя (manager_id), даты начала проекта (creation_date) и названия (title). Статусы проекта каталогизированы в отдельной таблице, включающей уникальный идентификатор (id) и заголовок статуса (title). Наконец, взаимоотношения участников и проектов учтены в таблице «Участники проекта», где каждый

участник связан с проектом и пользователем через соответствующие идентификаторы (project_id, user_id).

Такова отображённая структура проекта базы данных, предназначенная для эффективного взаимодействия разнообразных данных о пользователях, их участии в проектах и коммуникациях.

Эффективное управление информационными потоками в приложении для обмена сообщениями, а также гарантирование ее сохранности и целостности, являются ключевыми аспектами деятельности, которые осуществляются с помощью хорошо организованной структуры базы данных. Стабильность в функционировании указанного программного продукта, не говоря уже о его продолжительной эксплуатации, возможна благодаря такой организации данных.

2.4 Функциональная схема работы программы

Для более подробного описания всех возможностей, каждой страницы и каждой отдельно взятой функции программного обеспечения чата, были спроектированы функциональные блок-схемы работы программы.

Блок-схема - графическое изображение работы программы. Блок-схемы часто применяются для:

- описания алгоритмов и процессов;
- изучения работы программы;
- планирования разработки программного обеспечения;
- объяснения сложных процессов с помощью простых логичных диаграмм.

Функциональные блок-схемы позволяют изобразить последовательность действий графически, что обеспечит наглядное понимание работы механизма или программного продукта для пользователей.

Для создания блок-схем используются:

- прямоугольники;

– овалы;

– ромбы;

– стрелки.

Функциональная схема главного меню изображена на рисунке 6:

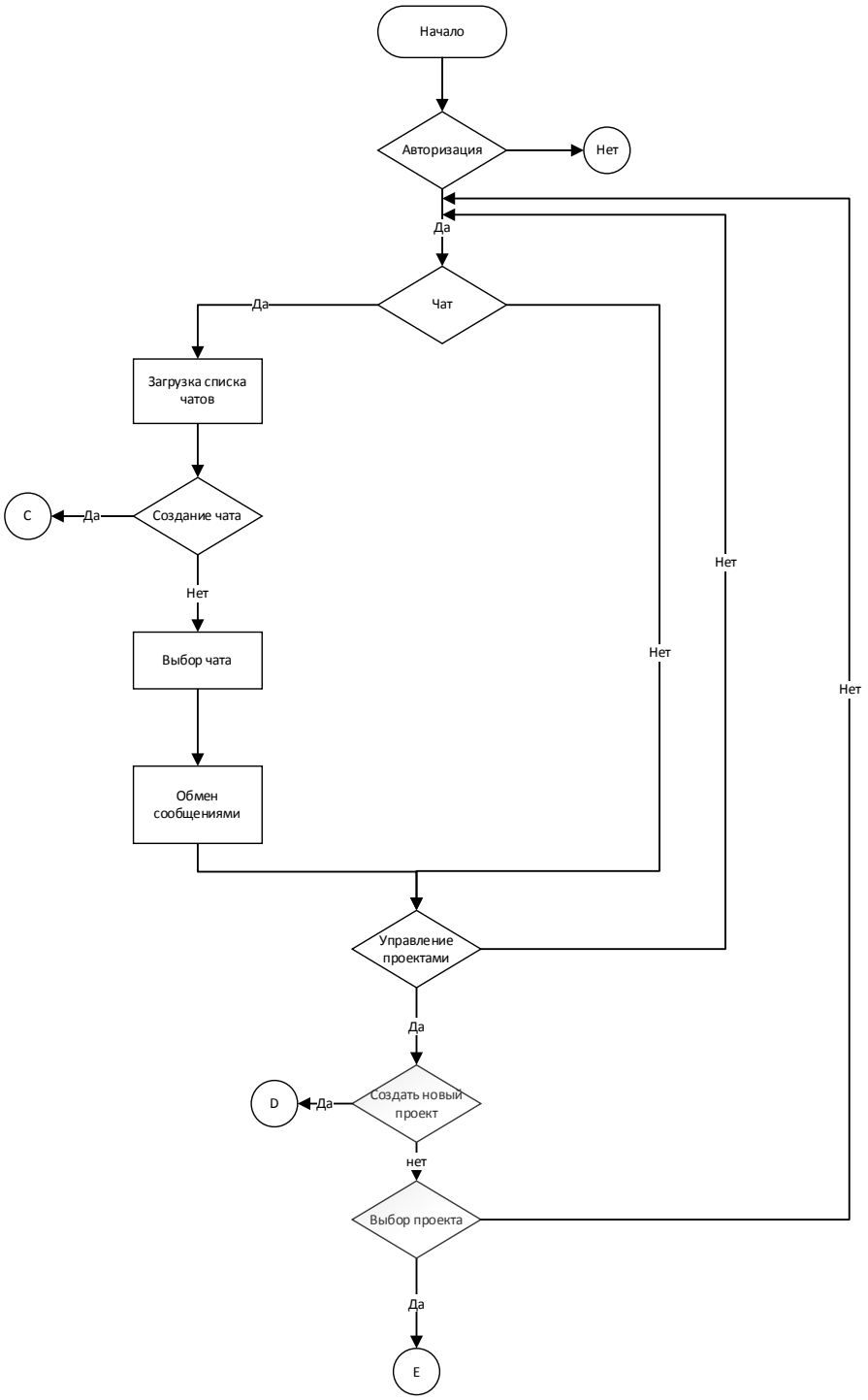

Рисунок 6 – Функциональная схема главной страницы чата

Если пользователь неавторизирован, он не сможет пользоваться возможностями чата и его перекинет на страницу авторизации со следующим функционалом (рисунок 7):

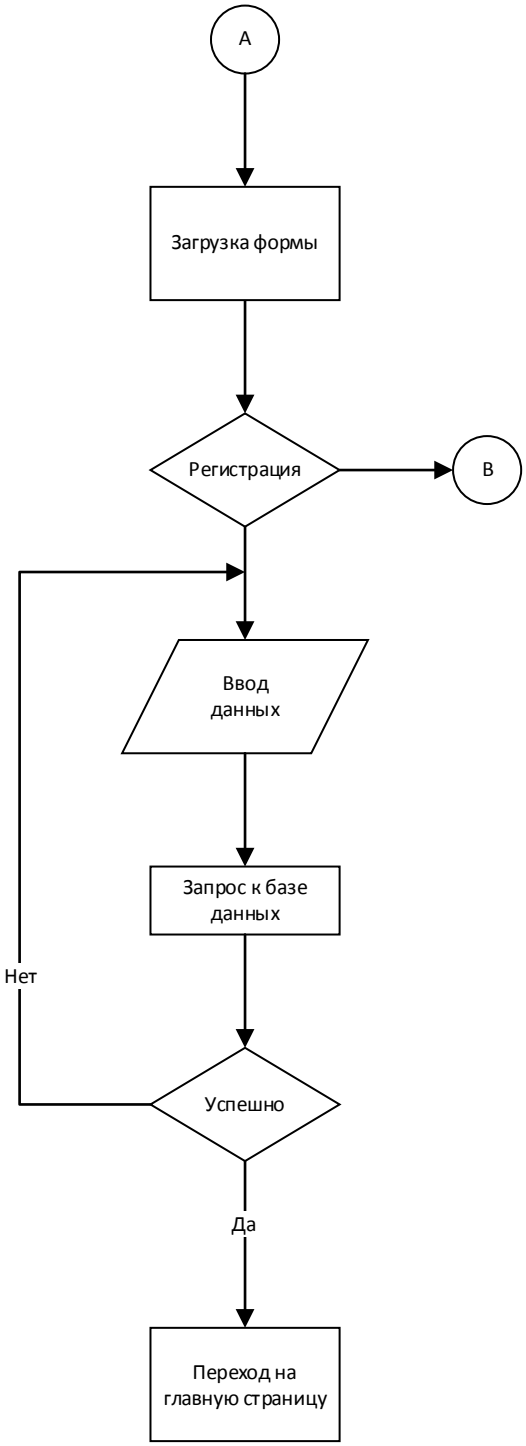

Рисунок 7 – Функциональная схема модуля авторизации пользователя

Если пользователь не имеет учетной записи – он не сможет автиризироваться и ему нужно будет пройти регистрацию. Для этого ему надо будет нажать на кнопку и перейти на форму регистрации (рисунок 8):

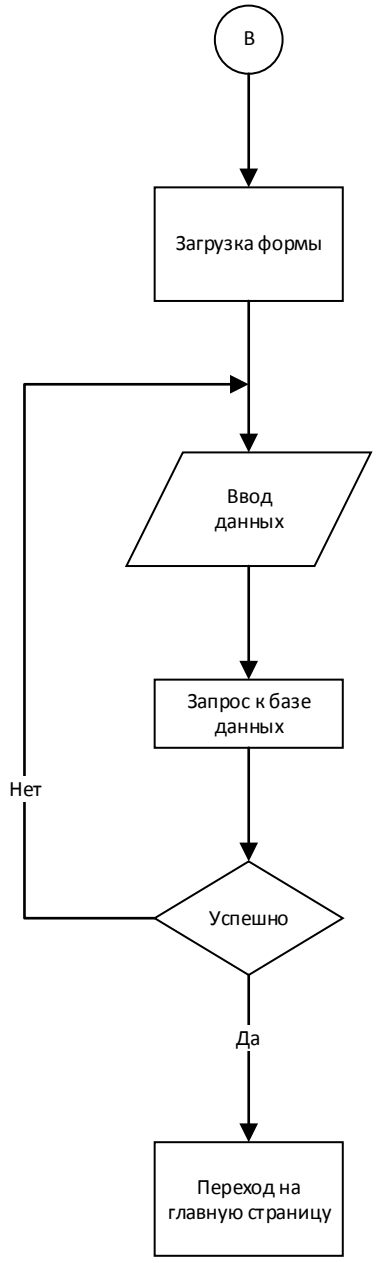

Рисунок 8 – Функциональная схема регистрации нового пользователя

И после авторизации, и после регистрации – пользователя перенесет на главную страницу, где ему откроются возможности программы. Если он перейдет на страницу чата, у него будет возможность создать новый чат или

выбрать существующий. Если он захочет создать чат, ему нужно будет нажать на кнопку и его перенест на форму создания чата (рисунок 9):

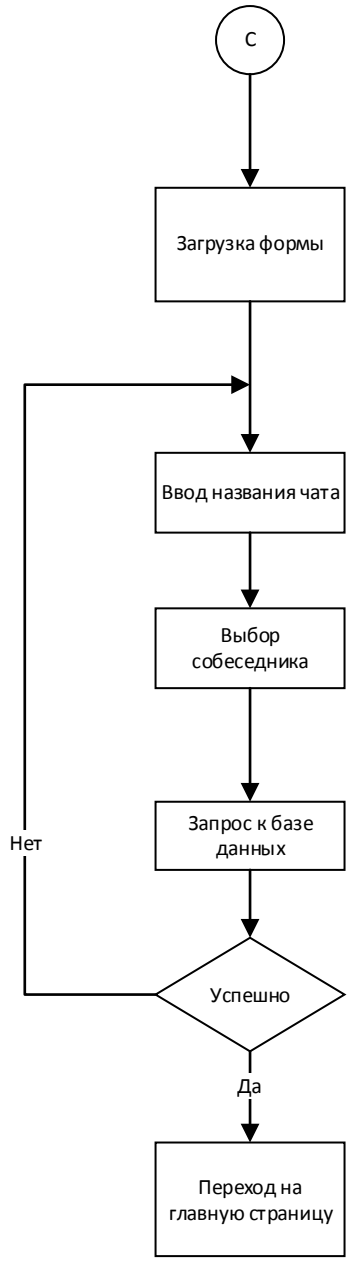

Рисунок 9 – Функциональная схема создания нового чата

После создания нового чата, пользователь снова вернется на главную страницу, где ему будет доступен созданный чат.

На главной странице пользователь, кроме чата, пользователь сможет перейти на вкладку «Управление проектами». В ней он сможет просмотреть все

свои проекты, создать новый или выбрать текущий. Если пользователю нужно создать новый проект, ему следует нажать на кнопку «Создать» и ему откроется форма создания (рисунок 10):

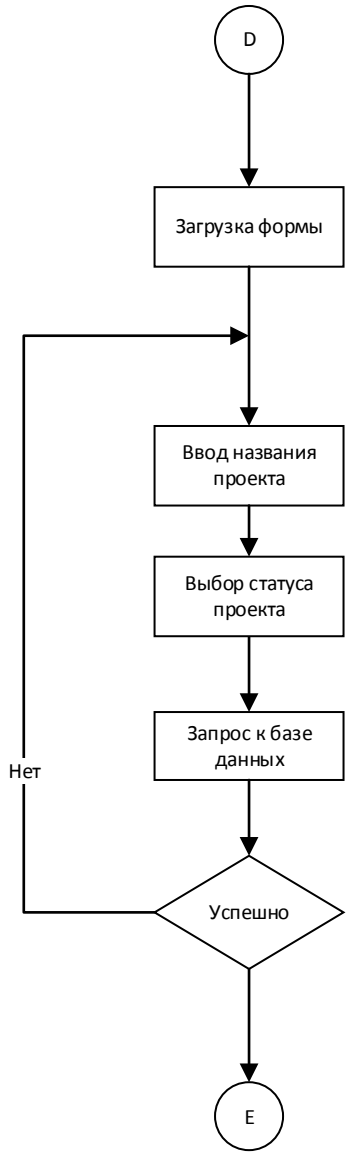

Рисунок 10 – Функциональная схема создания нового проекта

После создания, пользователь перенесется на страницу управления проекта. Там он сможет изменять статус проекта, добавить в него участников, а также вести переписку с участниками проекта. Функциональная схема управления проектом представлена на рисунке 11.

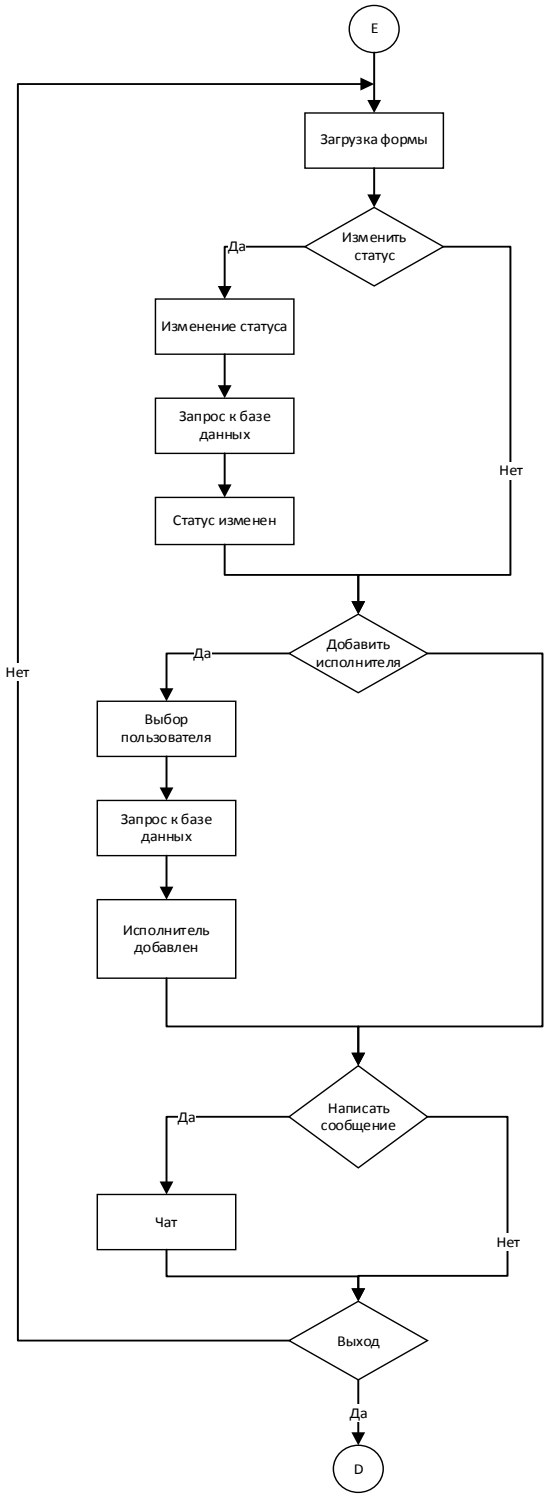

Рисунок 11 – Функциональная схема управления проектом

При помощи разработки функционала, спроектированного в этом разделе, программное обеспечение чата сможет обеспечить организации эффективное внутреннее общение, повысить производительность труда сотрудников и

улучшить их рабочие процессы. Неавторизированные пользователи не будут иметь доступа к функционалу приложения: созданию чатов и общению с другими пользователями.

Данный программный продукт должен выйти достаточно простым, понятным и удобным в использовании для пользователей любой квалификации. Ожидается, что он быстро получит положительный отклик.

Вывод по второй главе:

Во второй главе было произведено логическое моделирование приложения чата для автоматизации внутренней коммуникации сотрудников предприятия. Построены диаграммы последовательности, вариантов использования и сущностей. Спроектирована и создана база данных и построены функциональные схемы модулей приложения.

Глава 3 Реализация и тестирование программного обеспечения чата для автоматизации внутренней коммуникации сотрудников предприятия

3.1 Описание технологии разработки

Перед началом разработки необходимо определиться с инструментами. Необходимо создать веб-приложение, используя язык программирования Python, язык разметки html и современные технологии разработки.

Главным средством разработки приложения чата является Python. Python – это язык программирования с динамическим контролем типа, в котором имена во время выполнения программы могут представлять значения различных типов. Все данные в Python представлены объектами. Имена являются лишь ссылками на эти объекты и не несут нагрузки по декларации типа [2, с. 8].

Важным инструментом для разработки будет Flask – популярный веб-фреймворк для Python. Это прекрасный выбор для тех, кто уже знаком с основами Python и хочет применить эти знания в создании веб-приложений [12]. Он гораздо проще в освоении, в отличии от своего аналога Django. Тем не менее, не смотря на простоту, Flask позволяет создавать достаточно сложные и крупные приложения.

Flask – это очень маленький фреймворк, такой маленький, что его часто называют «микрофреймворк» но это не означает давать меньше, чем дают другие фреймворки. Flask изначально проектировался как расширяемый фреймворк – он имеет монолитное ядро, реализующее основные службы, а все остальное поддерживается посредством расширений. Он предоставляет достаточно ограниченный комплект программных средств, не поддающийся неконтролируемому разбуханию и в точности соответствующий потребностям разработчика [4, с. 21].

Чату также требуется авторизация в программе, чтобы сотрудники могли просматривать информацию о собеседниках и знать с кем ведут чат. Для этой задачи будет использоваться Flask-Login. Он входит в библиотеку Flask и отвечает за управление авторизацией пользователей. Flask-Login запоминает данные авторизированного пользователя и хранит их на протяжении всей сессии.

Для поддержания работы чата используется Socket.IO. Socket.IO - это мощный инструмент для создания приложений, работающих в режиме реального времени с двунаправленной связью между серверной и клиентской сторонами. Он использует вебсокеты, наряду с JSON и JSONP. Его можно использовать для создания двунаправленных приложений, таких как чат-приложения и многопользовательские игры [5, с. 16].

Для того, чтобы приложение могло обрабатывать запросы к базе данных будет использован SQLAlchemy. SQLALchemy - это библиотека Python, созданная Майком Байером для обеспечения высокоуровневого интерфейса к реляционным базам данных/ SQLAlchemy пытается быть ненавязчивым для Python-кода, позволяя разработчику отображать простые старые Python-объекты (POPO) из таблиц баз данных без существенного изменения существующего Python-кода. SQLAlchemy включает в себя как независимый от сервера базы данных язык выражений SQL, так и объектно-реляционный маппер (ORM), который позволяет использовать SQL для автоматического сохранения объектов приложения [7, с. 15].

SQLAlchemy взаимодействует с базой данных через запросы. База данных является основой для нескольких, часто независимых приложений и систем. Все пользователи (внешние и внутренние) испытывают на себе именно общую производительность системы, и это то, что имеет для них значение. Поэтому важным является оптимизация запросов [3, с. 26].

Для хранения необходимых данных для чата используется СУБД PostgreSQL – старейшая, но одна из самых стабильных систем управления реляционными базами данных. Когда PostgreSQL выполняется, она называется

сервером PostgreSQL, или экземпляром сервера. Данные, которыми управляет PostgreSQL, хранятся в базах данных. Один экземпляр PostgreSQL одновременно работает с несколькими базами, которые вместе называются кластером баз данных [9, с. 23].

Главные преимущества PostgreSQL:

- транзакции,
- возможность комментирования кода,
- параметры,
- расширяемость,
- безопасность.

Одним из самых важных моментов при разработке программ является выбор среды разработки. PyCharm – идеальный вариант для разработки веб-приложений. PyCharm специализируется исключительно на работе с Python. Создавая новый проект в PyCharm, можно увидеть варианты проектов Python, и ничего больше [1, с. 37].

PyCharm содержит в себе библиотеку фреймворков, а также удобен в работе с верстской и стилями, он помогает структурировать html файлы, а также писать стили, выбирать цвет из rgb палитры, что облегчает процесс разработки дизайна приложения.

Веб-страницы, на которых расположен контент приложения, написаны средствами html и css. HTML – язык разметки, предоставляющий разработчиками следующие возможности:

- представлении информации в виде веб страниц,
- включение в веб-страницы медиаконтента,
- загрузка веб-страниц по нажатию на гиперссылке,
- разработка форм для осуществления взаимодействия с контентом [6, с. 12].

Для того, чтобы разработанные веб-страницы могли демонстрировать функционал чата, в том числе и для связи с Socket.Io, используется JavaScript.

JavaScript — мультифункциональный язык, широко используемый в веб разработке. Любое действие на веб-странице — это работа JavaScript [10, с. 19].

Сама идея написания именно веб-приложения, а не десктопного, основана на том, что оно кроссплатформенно. Чтобы им пользоваться, достаточно иметь лишь веб-браузер. Это важный фактор, поскольку Microsoft Windows приостановили работу в России и многие предприятия вынуждены переходить на Alt Linux, Astra Linux и т.п..

Несмотря на то, что для работы приложения как таковых системных требований не предусмотрено, тем не менее, запуск сервера для программного обеспечения чата происходит на отдельном устройстве. Существуют определенные требования к серверу. Минимальные требования к серверу представлены в таблице 2:

Таблица 2 - Минимальные аппаратные требования к серверу

Процессор	4 ядра (8 логических потоков), частота – 3,5 ГГц и больше
Оперативная память	12 ГБ и больше
HDD для IIS и документов	50 ГБ (зависит от размера хранимых в системе документов)
SSD для SQL	300 ГБ и больше
Требования к сети	Стабильный канал связи от 10 Мб/сек

Рекомендуемые требования представлены в таблице 3:

Таблица 3 – Рекомендуемые аппаратные требования к серверу

Процессор	8 ядер (16 логических потоков), частота – 3,5 ГГц и больше
Оперативная память	64 ГБ и больше
HDD для IIS и документов	128 ГБ (зависит от размера хранимых в системе документов)
SSD для SQL	300 ГБ и больше
Требования к сети	Стабильный канал связи от 10 Мб/сек

Соответствие приведенным системным требованиям обеспечит стабильную устойчивость сервера для работы программного обеспечения чата. В ином случае, существует риск возникновения ошибок, сбоев, что вынудит к перезапуску сервера и прекращению сессий.

3.2 Реализация программного обеспечения чата для внутренней коммуникации сотрудников предприятия

Первым делом, при запуске веб-приложения, пользователю необходимо авторизироваться или зарегистрироваться для продолжения работы. После успешной авторизации пользователь получит доступ к приложению и ему откроется главная страница чата (рисунок 12):

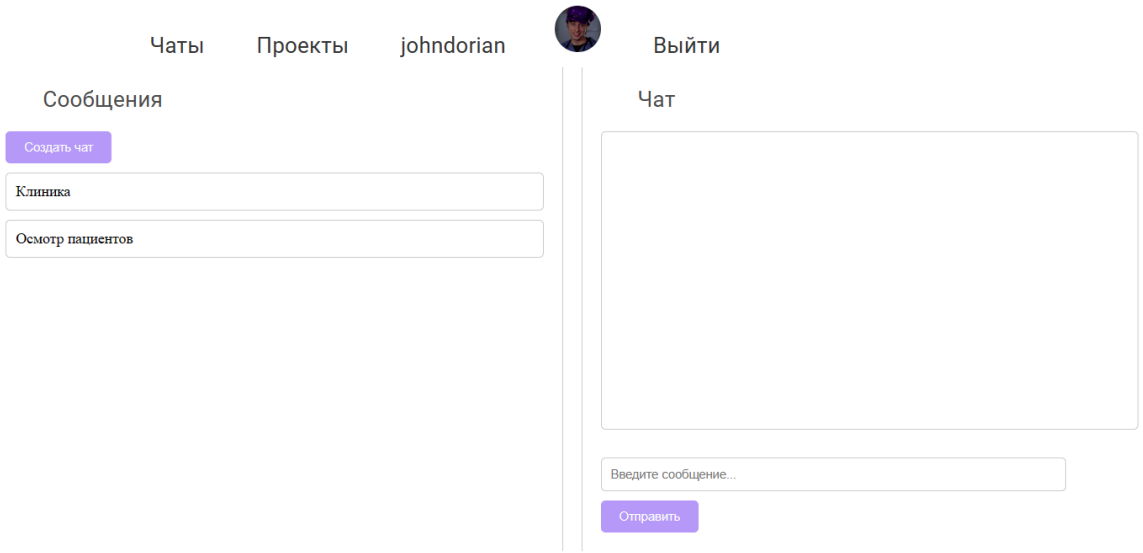

Рисунок 12 – Главная страница чата

По умолчанию, активной вкладкой является «Чаты», поэтому пользователь видит страницу с чатами. Окно чата, что расположен справа, на данный момент не реагирует на нажатия, посколько не выбран ни один чат. Пользователь может выбрать, либо создать новый чат. Для того, чтобы создать чат, необходимо нажать на кнопку «Создать чат» в левой части страницы (рисунок 13):

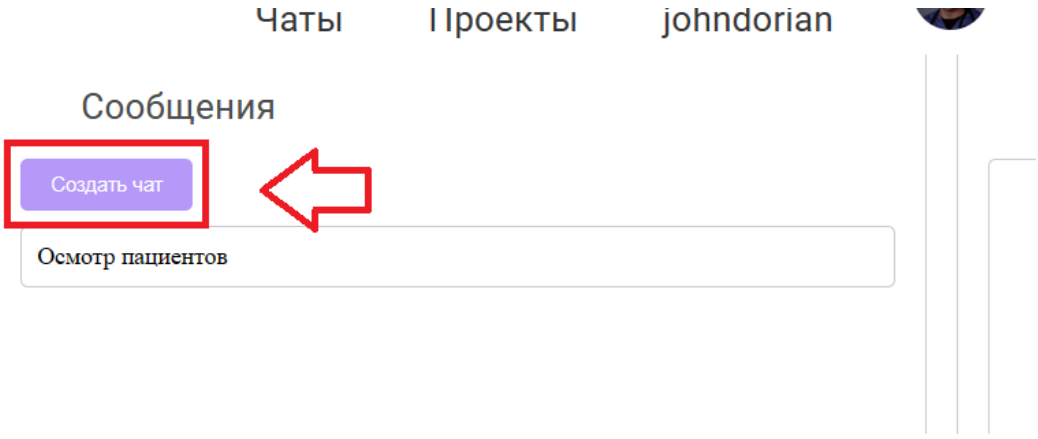

Рисунок 13 – Кнопка «Создать чат»

По нажатию на нее откроется форма создания чата (рисунок 14):

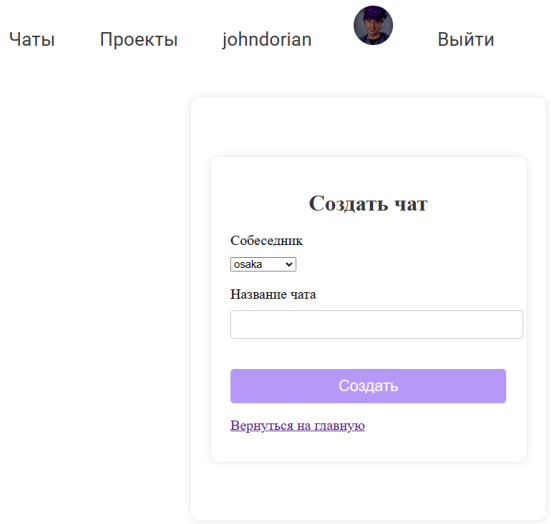

Рисунок 14 – Форма создания чата

Пользователю необходимо ввести название чата и выбрать собеседника из списка пользователей. Процесс выбора собеседника изображен на рисунке 15:

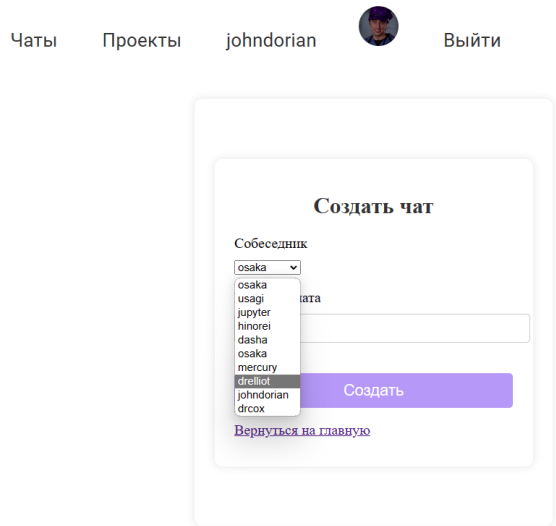

Рисунок 15 – Выбор собеседника для создания чата

После создания чата, пользователя снова переведет на главную страницу, но на этот раз его список бесед уже не будет пустым (рисунок 16):

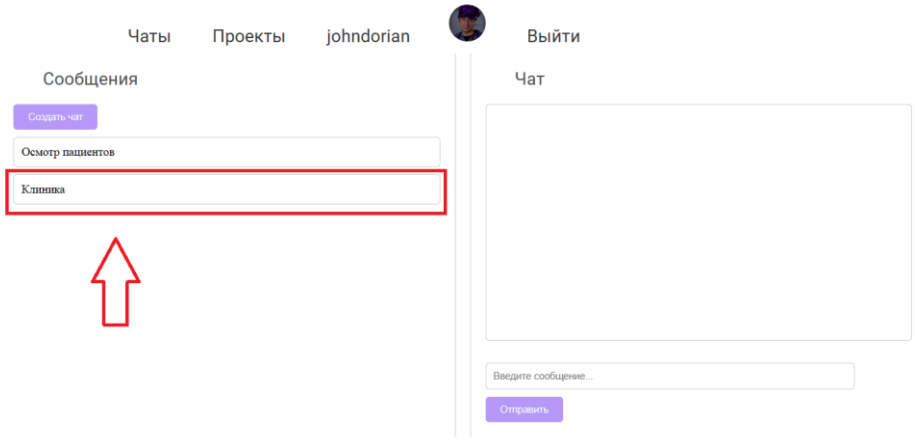

Рисунок 16 – Созданный чат

Теперь пользователь johndorian может начать общение с drelliot. Для этого ему необходимо нажать на чат, ввести сообщение в поле ввода и нажать на кнопку «Отправить» или на клавишу «Enter» (рисунок 17):

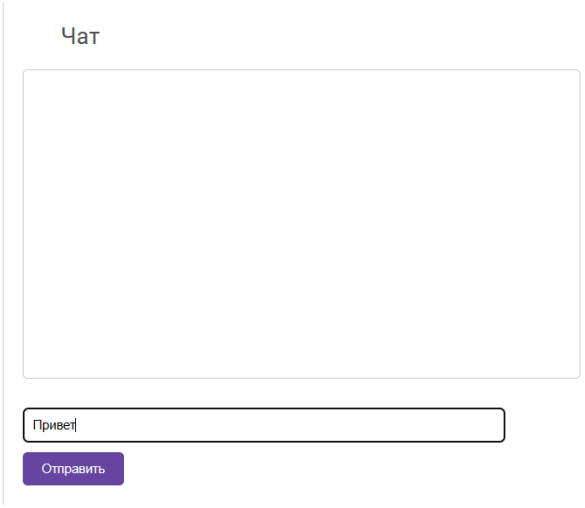

Рисунок 17 – Ввод и отправка сообщения

После успешной отправки сообщения, оно отобразится в окне чата (рисунок 18):

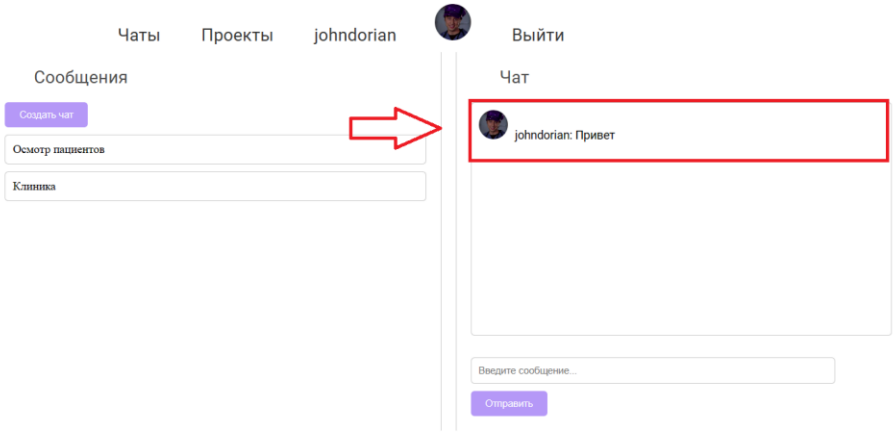

Рисунок 18 – Вывод нового сообщения

После отправки сообщения, пользователю johndorian остается ждать ответа. Тем временем, на другом конце, пользователь drelliot, нажав на чат, сразу увидит сообщение (рисунок 19):

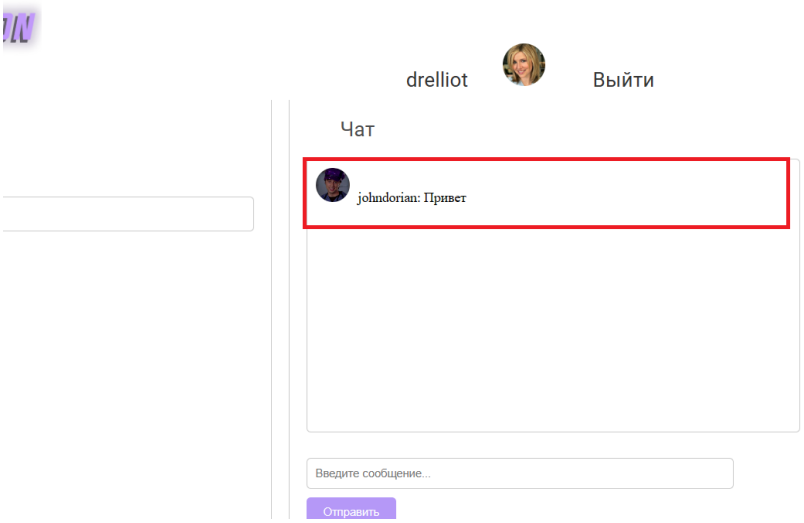

Рисунок 19 – Получение сообщения

Увидев сообщение, пользователь drelliot может сразу на него ответить (рисунок 20) и быть уверенной в том, что пользователь johndorian обязательно сразу его увидит (рисунок 21) .

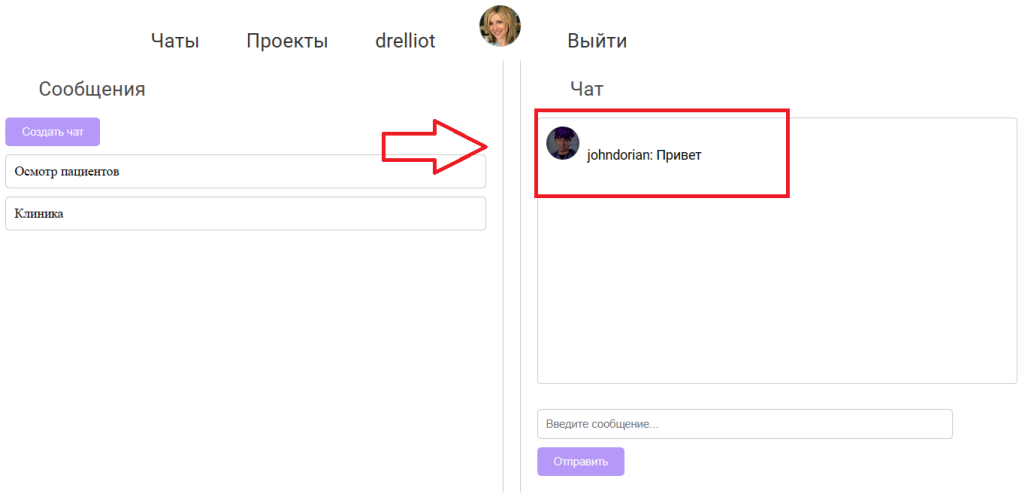

Рисунок 20 – Получение сообщения и ввод ответа

Пользователь drelliot ответит соответствующим «привет» и johndorian сразу это увидит (рисунок 21):

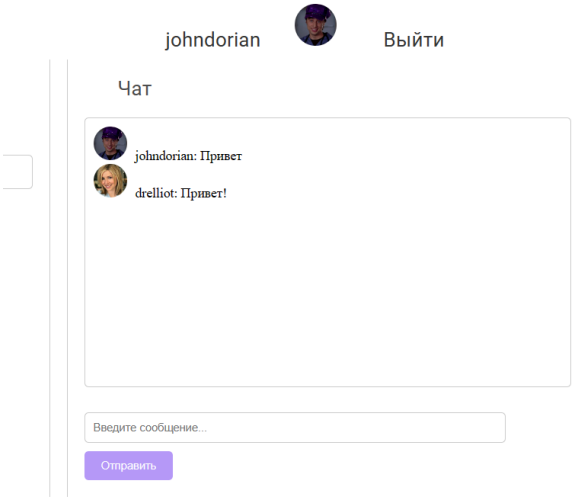

Рисунок 21 – Получение ответного сообщения

Также, в чаты можно добавлять пользователей и тогда они станут многопользовательскими (рисунок 22):

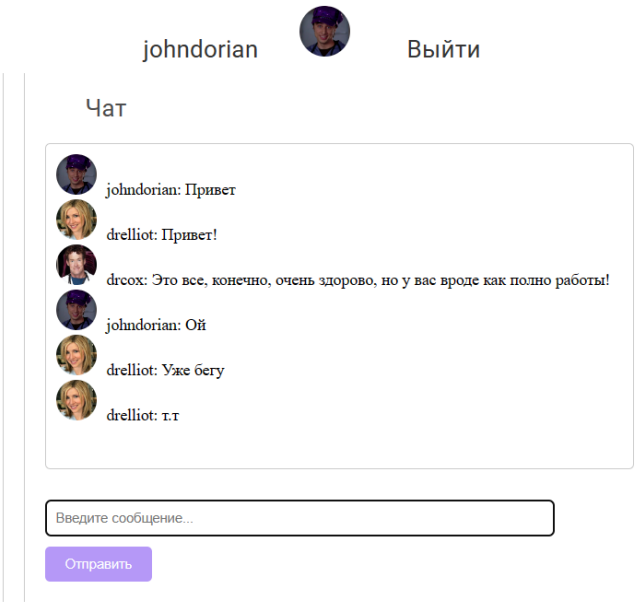

Рисунок 22 – Многопользовательский чат.

После того, как drcox сделал замечание сотрудникам, им пора приняться за работу и перейти во вкладку «Проекты» (рисунок 23):

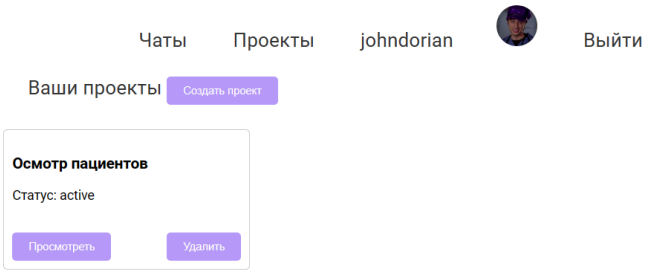

Рисунок 23 – Страница «Проекты»

На данной странице отображены все проекты пользователя. Он видит свои задачи, их статусы. Он может создать проект, нажав на кнопку «Создать проект» (рисунок 24):

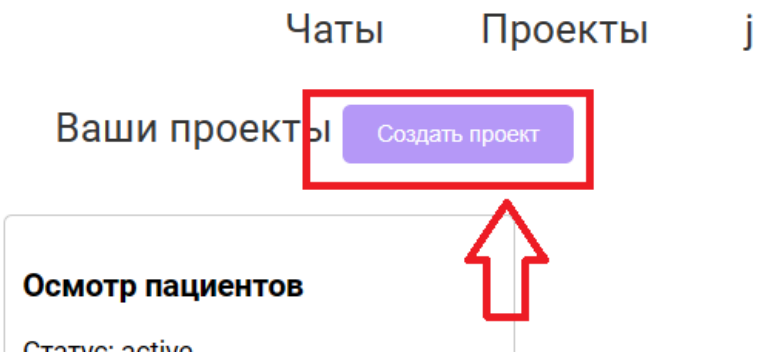

Рисунок 24 – Кнопка «Создать проект»

После этого ему откроется страница для создания проекта. Здесь пользователю нужно ввести название, выбрать статус проекта и нажать на кнопку «Создать» (рисунок 25):

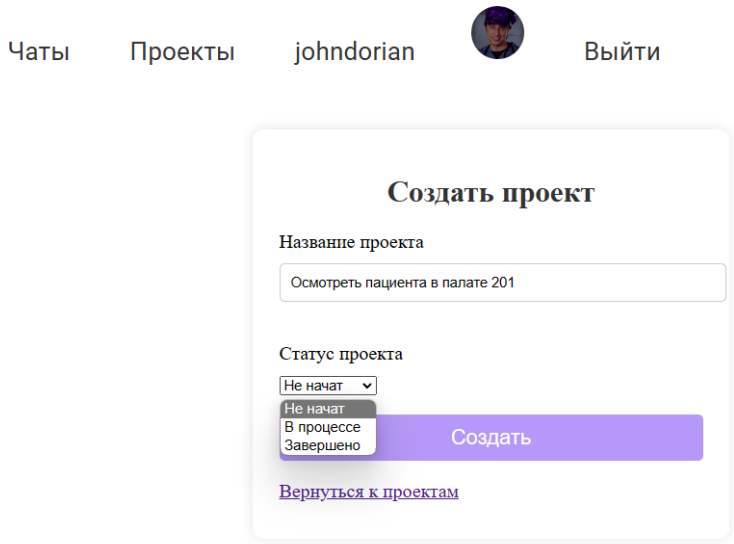

Рисунок 25 – Создание проекта

После удачного создания проекта, пользователь johndorian попадет на страницу управления проектом, изображенную на рисунке 26:

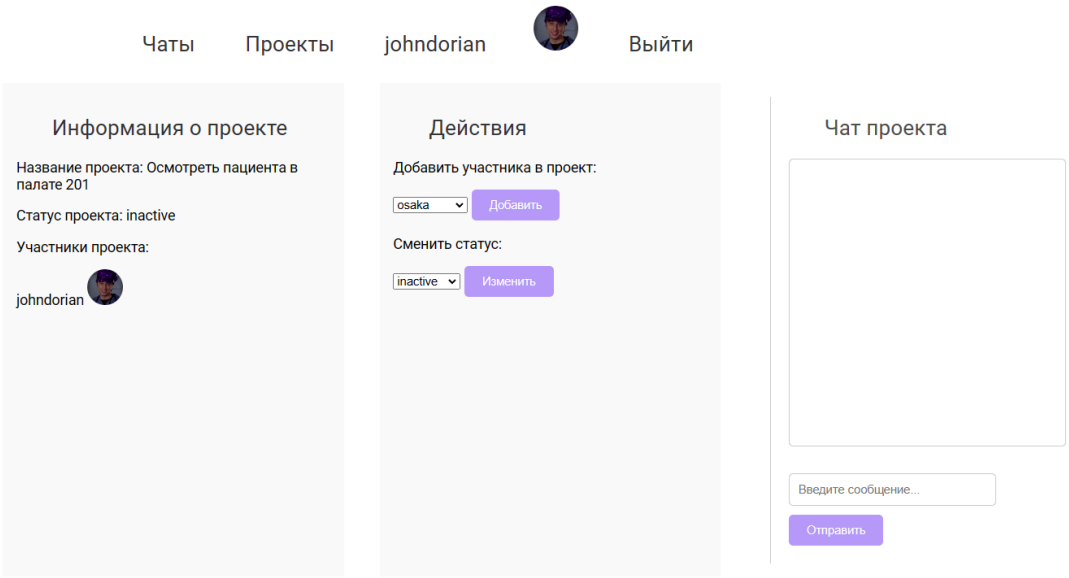

Рисунок 26 – Управление проектом

Теперь он может изменить статус проекта и сделать его активным. Для этого ему нужно выбрать статус и нажать «Изменить» (рисунок 27):

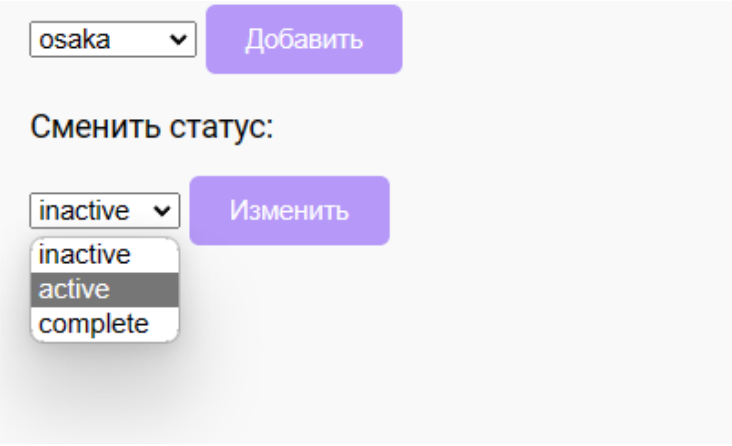

Рисунок 27 – Изменение статуса проекта

После совершения этого действия, статус проекта сразу обновится (рисунок 28):

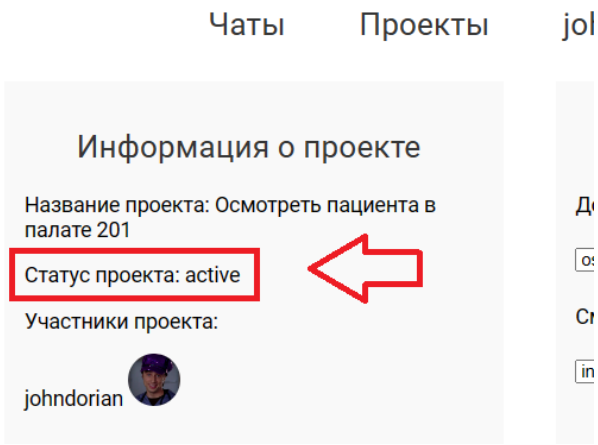

Рисунок 28 – Изменение статуса проекта

Также он может добавить участника в проект. Для этого ему необходимо выбрать пользователя и нажать «Добавить» (рисунок 29):

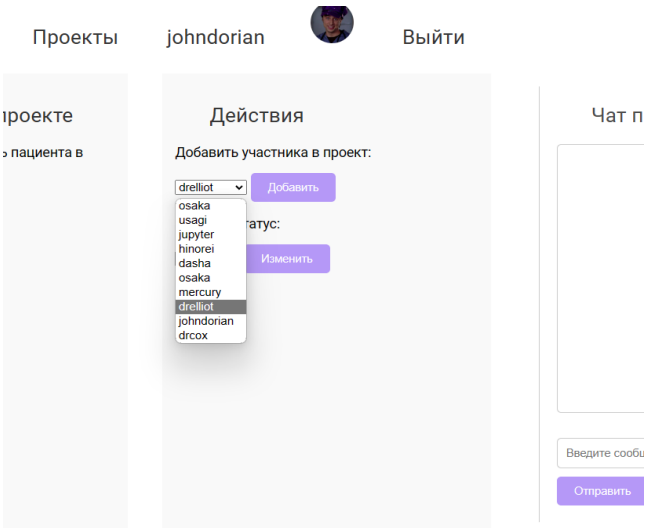

Рисунок 29 – Добавление сотрудника в проект

После выполнения этого действия, к списку участников добавится выбранный пользователь drelliot. Список участников проекта сразу же обновится (рисунок 30):

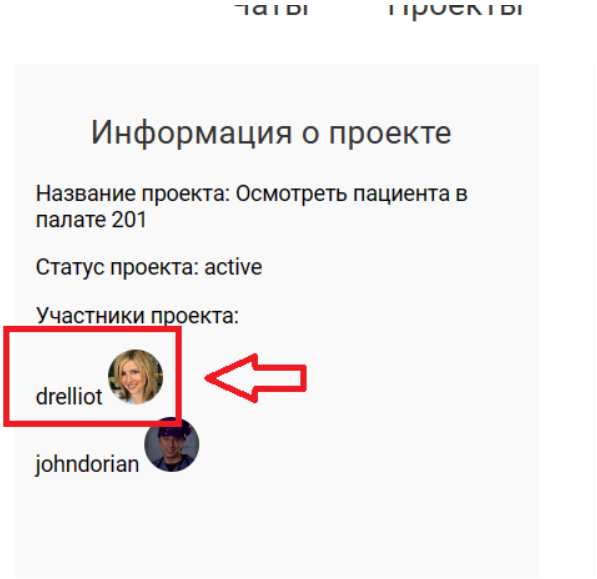

Рисунок 30 – Добавленный участник проекта

Теперь у drelliot в списке проектов есть проект «Осмотреть пациента в палате 201» (рисунок 31):

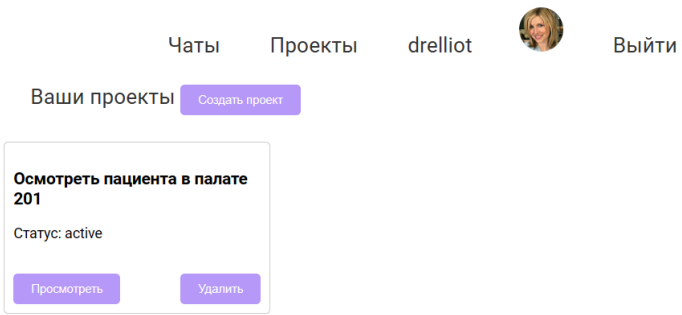

Рисунок 31 – Обновление списка проектов

Ей осталось только нажать на кнопку просмотреть, чтобы увидеть детали проекта (рисунок 32):

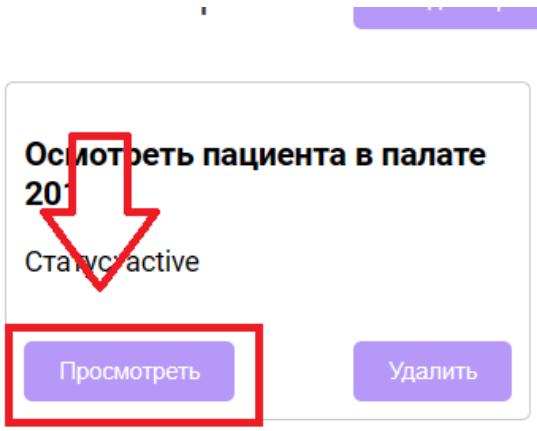

Рисунок 32 – Кнопка «Просмотреть» у проекта

После нажатия, она отправится на страницу проекта, изображенную на рисунке 33:

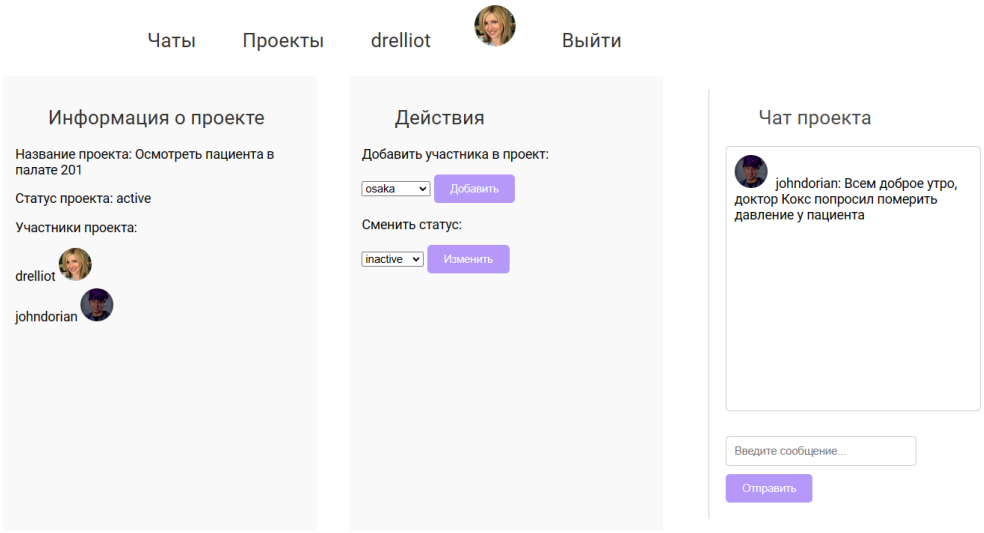

Рисунок 33 – Переход на страницу проекта

Наконец, drelliot может приступить к работе. Она, как участник проекта, также может изменять статус проекта и перекидывать ответственность на других, приглашая сотрудников к исполнению задачи. Также она всегда может воспользоваться чатом и ответить (рисунок 34):

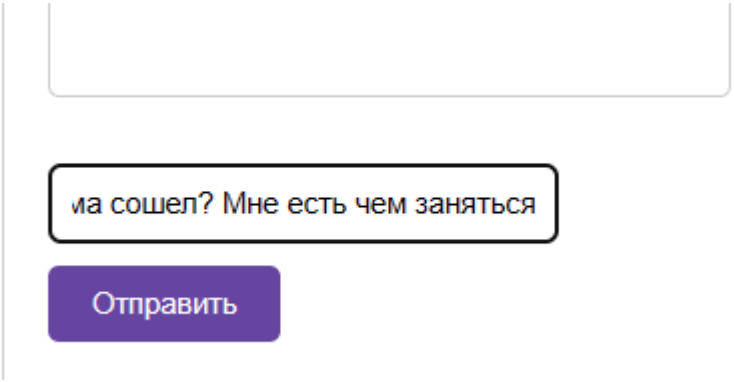

Рисунок 34 – Ввод сообщения ответа

После ввода сообщения, она точно также нажимает на кнопку «Отправить» и сообщение сразу отобразится в меню чата (рисунок 35):

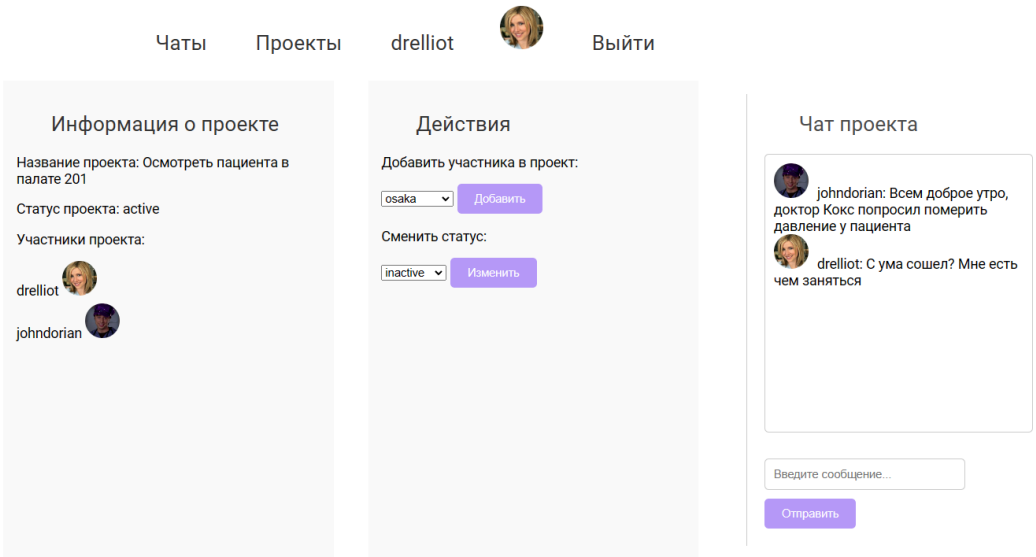

Рисунок 35 – Чат внутри проекта

Написанное приложение обладает простым функционалом, достаточным для корпоративного решения чата для автоматизации внутренней коммуникации сотрудников предприятия. Также, имеет небольшой функционал управления проектами, что отличает его от своих аналогов.

Полный код программы расположен в Приложении А.

3.3 Тестирование программного обеспечения

Этап тестирования программы на ошибочные ситуации играет ключевую роль в обеспечении надежности и стабильности программного продукта. Используемый метод тестирования – метод «Черного ящика» - это когда пользователь вводит всевозможныеданные, не смотря в код.

Важно отметить, что тестирование позволяет выявить проблемы, которые могут возникнуть в реальной эксплуатационной среде, и предотвратить негативное влияние на вероятный негативный пользовательский опыт.

В случае неудачной авторизации, пользователь увидит соответствующее сообщение об ошибке (рисунок 35):

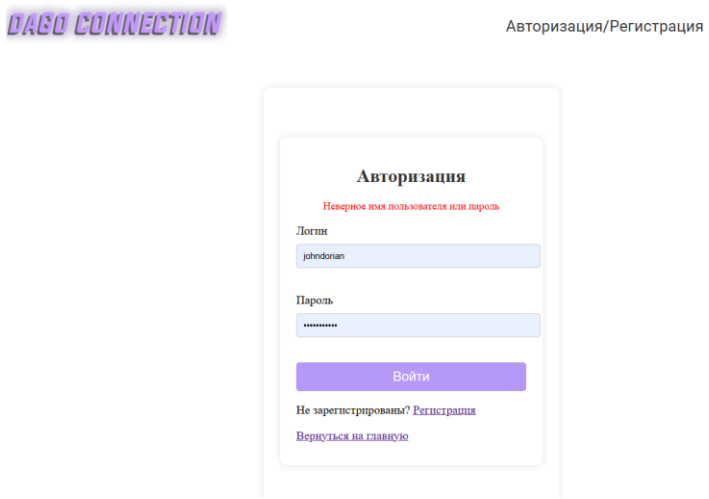

Рисунок 35 – Обработка неудачной авторизации

Во время регистрации, в случае ввода слишком короткого имени пользователя, появится сообщение, изображенное на рисунке 36:

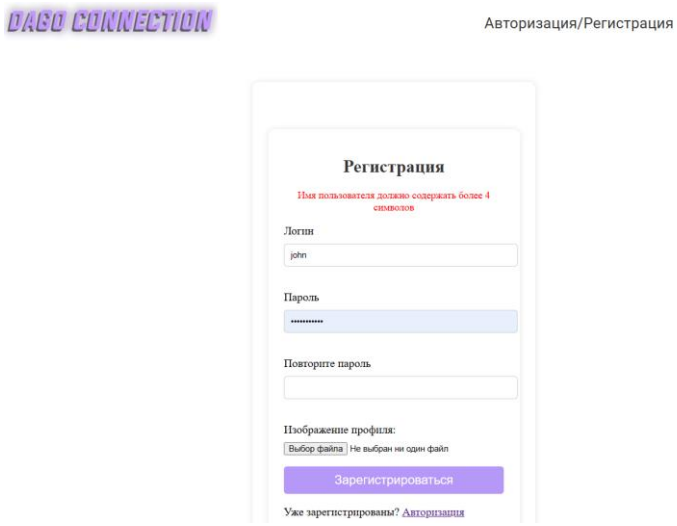

Рисунок 36 – Обработка короткого имени пользователя

Если пользователь некорректно вводит повторно пароль в поле «Повторите пароль», он увидит сообщение, изображенное на рисунке 37:

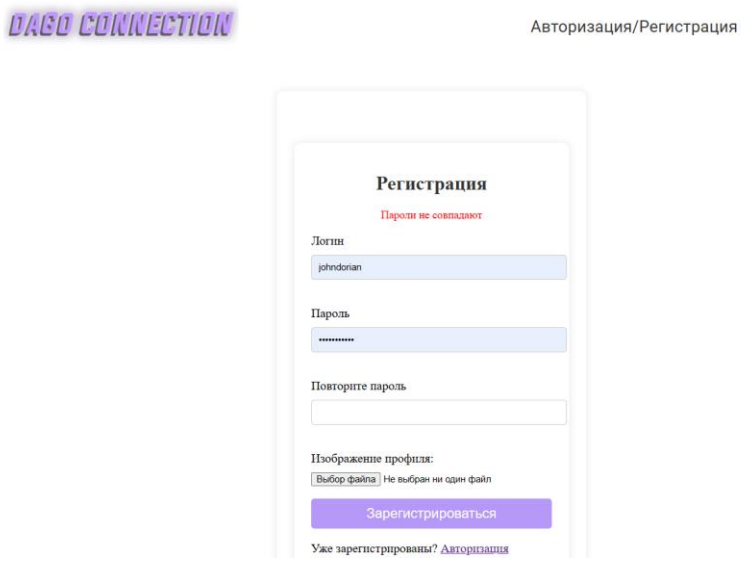

Рисунок 37 – Обработка некорректного повторного ввода пароля

При попытке зарегистрироваться с логином, уже зарегистрированном в бд, появится ошибка, изображенная на рисунке 38:

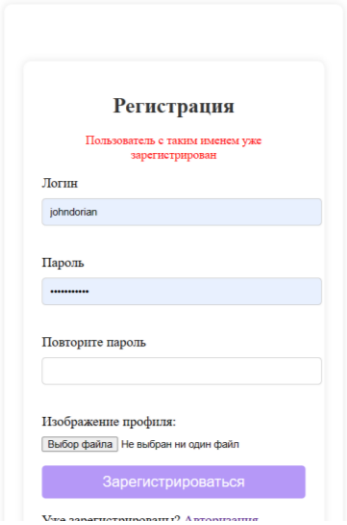

Рисунок 38 – Обработка уникальности имени пользователя

Если пользователь забудет ввести название чата, то увидит соответствующее сообщение об ошибке, изображенное на рисунке 39:

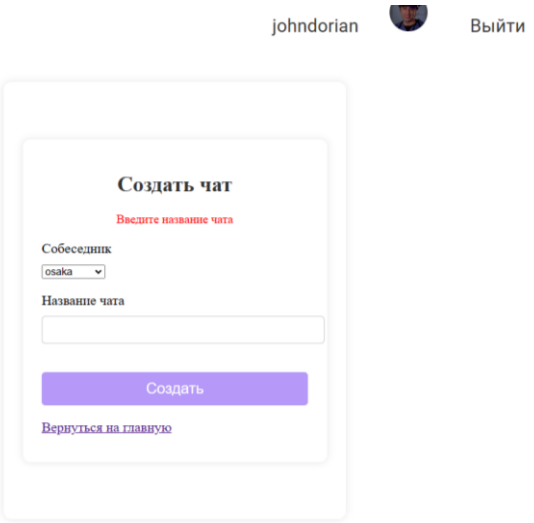

Рисунок 39 – Обработка пустого названия чата

Создать безымянный проект у пользователя тоже не выйдет (рисунок 40):

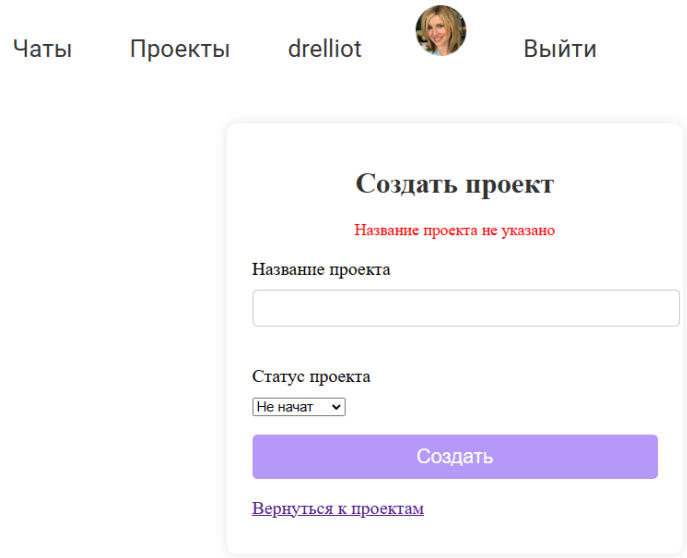

Рисунок 40 – Обработка создания безымянного проекта

В результате тестирования, были проработаны возможные сценарии ошибочного ввода пользователем информации. В случае возникновения стрессовых ситуаций, в программе выводятся соответствующие сообщения, информирующие о допущенных ошибках, которые дадут понять, в чем проблема.

Пользователь не сможет создать свой дубликат в базе данных, так как в программе написаны соответствующие проверки. Зарегистрироваться с коротким именем пользователя или небезопасным паролем у него тоже не выйдет, так как есть определенные требования к вводу, который обрабатываются программой. Зарегистрироваться со случайно введенным или ошибочным паролем пользователь не сможет, так как ему нужно будет его продублировать.

Пользователь не сможет создать чат или проект без названия. Программа предусматривает возникновение подобных ситуаций и реагирует на них, соответствуя ожиданиям.

Но тестирование также выявило такие необработанные ситуации, что любой пользователь может удалить проект. Удалить чаты и сообщения можно только через базу данных.

Вывод по третьей главе:

Были выбраны инструменты разработки программного обеспечения чата для внутренней коммуникации сотрудников предприятия. Определены требования к аппаратному обеспечению сервера. Выявлены преимущества каждого программного средства и обоснован их выбор.

Было разработано программное обеспечение и описан порядок действий по работе с приложением:

- создание чатов,
- коммуникация,
- создание проектов,
- управление проектами.

Проведено тестирование методом «Черного ящика» на внештатные ситуации. Оно выявило как правильные реакции программы на ошибки, так и необработанные ситуации.

В соответствии с вышеупомянутой проделанной работой с разработанным программным обеспечением чата для внутренней коммуникации сотрудников предприятия, были выявлены главные преимущества и недостатки. Несмотря на общую примитивность приложения в сравнении с более продвинутыми аналогами, оно может предложить все необходимые функции. Кроме того, имеет возможность совместного управления проектами, что послужит приятным дополнением к рабочему процессу любой организации.

Заключение

В процессе выполнения бакалаврской работы, была проанализирована проблема взаимодействия сотрудников внутри предприятия, нуждающаяся в автоматизации, исследованы различные возможности для автоматизации коммуникации, что позволило получить более эффективное решение на выходе.

В результате выполненной бакалаврской работы, было разработано программное обеспечение для внутренней коммуникации сотрудников предприятия в целях обеспечения удобного взаимодействия сотрудников внутри организации. Оно работает в локальной сети, что сохраняет конфиденциальность передаваемой внутри приложения информации. Функционал программы ограничен лишь самыми необходимыми возможностями, что не позволит неквалифицированным пользователям подхватить вредоносное программное обеспечение.

В процессе работы над бакалаврской работой, то можно выделить следующие выполненные задачи:

- был выбран и использован качественный инструментарий разработки для лучшей эффективности;
- спроектирована и создана база данных для хранения информации о пользователях, чатах и сообщениях;
- разработано адаптированное веб-приложение, запускаемое как на десктопных устройствах, так и на мобильных;
- написаны стили для интерфейса программного обеспечения в целях удобной и эффективной работы сотрудников;
- проведено тестирование возможных стрессовых ситуаций и исправлены выявленные ошибки.

Программа является простой в использовании. Имеет интуитивно понятный и стилизированный интерфейс. Работает быстро и снабжена защитой от сбоев в случае возникновения стрессовых ситуаций.

Полученный результат полностью соответствует ожиданиям. Разработанный функционал программного обеспечения выполняет требуемые задачи.

В конечном итоге, разработанное решение готово к использованию и подходит для оптимизации бизнес-процесса внутренней коммуникации сотрудников предприятия. Оно позволяет оптимизировать рабочие процессы и повысить качество условий труда и выполняемых задач.

Список используемой литературы и используемых источников

1. Брюс М., Ван Хорн II., Куан Нгуен PYCHARM: профессиональная работа на PYTHON [Текст] / М.: ДМК Пресс, 2024 — 618 с.;
2. Боклаг Н. Ю., Буйначев С. К. Основы программирования на языке Python [Текст] / СПб: М., 2016 — 93 с.;
3. Домбровская Г.Р., Новиков Б. Оптимизация запросов в PostgreSQL // М. : ДМК Пресс, 2022 — 278 с.;
4. Гринберг М. Flask Web Development. М. : ДМК Пресс, 2018 — 274 с.;
5. Каденхед Т. Socket.IO Cookbook [Книга] / Бирмингем: Packt Publishing, 2015 — 184 с.;
6. Кириченко А. В., Хрусталев А А HTML5 + CSS3. Основы современного WEB-дизайна [Текст] — 2-е изд. / СПб: Наука и техника, 2019 — 352 с.;
7. Коплэнд Р. Essential SQLAlchemy [Текст] / Себастопол: O'Reilly Media, 2020 — 230 с.;
8. Одна платформа для общения [Электронный ресурс] : Zoom. URL: https://zoom.us (дата обращения: 06.04.2024);
9. Рогов Е.В. PostgreSQL 16 изнутри — 16 изд. — М.: ДМК Пресс, 2024 — 664 с.;
10. Свекис Л.Л., ван Путтен М. Javascript с нуля до профи // СПб. : ПИТЕР, 2023 — 476 с.;
11. Фаулер М. UML. Основы [Книга] — 3-е издание. / СПб: Символ-Плюс, 2017 — 181 с.;
12. Flask [Электронный ресурс] : Full Stack Python. URL: https://pahaz.github.io/flask.html (дата обращения: 30.03.2024) ;
13. Google Chat [Электронный ресурс] : Google Workspace. URL: https://workspace.google.com/intl/ru/products/chat (дата обращения: 04.04.2024);

14. Made for people. Built for productivity [Электронный ресурс] : Slack. URL: https://slack.com;

15. Microsoft Teams [Электронный ресурс] : Microsof. URL: https://www.microsoft.com/ru-ru/microsoft-teams/group-chat-software (дата обращения: 04.04.2024);

16. Simply the most Secure Messenger [Электронный ресурс] : Wire. URL: https://wire.com/en (дата обращения: 04.04.2024);

17. Slack: обзор корпоративного мессенджера, который считался лучшим решением для бизнеса [Электронный ресурс] : COMPASS. URL: https://getcompass.ru/blog/posts/slack?ysclid=lv4apz8oza352876065 (дата обращения: 04.04.2024);

18. Telegram FAQ [Электронный ресурс] : Telegram. URL: https://telegram.org/faq (дата обращения: 04.04.2024);

19. Threema - The Secure Messenger For Individuals and Companies [Электронный ресурс] : Threema. URL: https://threema.ch/en (дата обращения: 04.04.2024);

20. WhatsApp [Электронный ресурс] : WhatsApp. URL: https://www.whatsapp.com (дата обращения: 04.04.2024).

Приложение А

Листинг программы

main.py:

```
import os.path
import secrets

from flask import Flask, render_template, request, flash, redirect, url_for, jsonify
from flask_login import LoginManager, login_user, login_required, logout_user, current_user
from flask_socketio import SocketIO, send, emit

from dataBase import dataBase
from userlogin import userlogin
from flask_sqlalchemy import SQLAlchemy
from werkzeug.utils import secure_filename

from models import User

app = Flask(__name__)
app.secret_key = secrets.token_hex(16)
app.config['UPLOAD_FOLDER'] = 'static/uploads/'

app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql://postgres:Abdfik45@localhost:5433/msgrdb'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db_instance = dataBase()
db = SQLAlchemy(app)
# db = dataBase()
loginManager = LoginManager(app)
socketio = SocketIO(app)


@loginManager.user_loader
def load_user(user_id):
 return userlogin().fromDB(user_id, db_instance)

@app.route('/')
def index():
 if current_user.is_authenticated:
 chats = dataBase().getUserChats(current_user.getId())
 return render_template('index.html', chats=chats, current_user=current_user)
 else:
 chats = []
 return render_template('index.html', chats=chats)

@app.route('/authorization', methods=["POST", "GET"])
def authorization():
 if request.method == "POST":
 login = request.form['login']
 password = request.form['pass']
 user = dataBase().getUserByLogin(login, password)
 if user:
 userLogin = userlogin().create(user)
 login_user(userLogin)
 return redirect(url_for('index'))
 else:
```

```
            flash("Неверное имя пользователя или пароль")
 return render_template('authorization.html')


@app.route('/registration', methods=["POST", "GET"])
def registration():
 if request.method == "POST":
 login = request.form['login']
 password = request.form['pass']
 repeat_password = request.form['repeatPass']

 # Проверка условий для успешной регистрации
 if len(login) <= 4:
 flash("Имя пользователя должно содержать более 4 символов", "error")
 return redirect(url_for('registration'))
 elif len(password) <= 4:
 flash("Пароль должен содержать более 4 символов", "error")
 return redirect(url_for('registration'))
 elif password != repeat_password:
 flash("Пароли не совпадают", "error")
 return redirect(url_for('registration'))

 profilepic = request.files['profilepic']
 if profilepic:
 filename = secure_filename(profilepic.filename)
 filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
 profilepic.save(filepath)
 res = dataBase().addUser(login, password, filepath)
 else:
 res = dataBase().addUser(login, password)

 if res:
 return redirect(url_for('authorization'))
 else:
 flash("Пользователь с таким именем уже зарегистрирован", "error")
 return redirect(url_for('registration'))

 return render_template('/registration.html')


@app.route('/logout')
@login_required
def logout():
 logout_user()
 return redirect(url_for('index'))


@app.route('/create_chat', methods=['GET', 'POST'])
@login_required
def create_chat():
 users = User.query.all()

 if request.method == 'POST':
 title = request.form['title']
 userOne = request.form['currentUser']
 userTwo = request.form['partner']
 chat_type = 0
```

```
 if len(title) < 1:
 flash("Введите название чата", "error")
 return redirect(url_for('create_chat'))
 res = dataBase().CreateChat(userOne, userTwo, title, chat_type)
 if res:
 return redirect(url_for('index'))
 else:
 flash("Неудача")

 return render_template('/createChat.html', users=users)


@app.route('/load_chat_messages')
def load_chat_messages():
 chat_id = request.args.get('chat_id')
 messages = dataBase().getChatMessages(chat_id)
 return jsonify(messages)


@app.route('/send_message', methods=['POST'])
@login_required
def send_message():
 if request.method == 'POST':
 data = request.json
 chat_id = data.get('chat_id')
 sender_id = current_user.getId()
 message_text = data.get('text')

 dataBase().sendMessage(chat_id, sender_id, message_text)

 return jsonify({'success': True}), 200
 else:
 return jsonify({'error': 'Method not allowed'}), 405


@socketio.on('message_sent')
def handle_message(data):
 chat_id = data['chat_id']
 sender_id = data['sender_id']
 message_text = data['message_text']
 dataBase().sendMessage(chat_id, sender_id, message_text)
 emit('new_message', {'sender_id': sender_id, 'message_text': message_text}, room=chat_id)


@app.route('/projects')
@login_required
def projects():
 user_id = current_user.getId()
 projects = dataBase().getProjectsForUser(user_id)
 return render_template('projects.html', projects=projects)


@app.route('/create_project', methods=['GET', 'POST'])
@login_required
def create_project():
 if request.method == 'POST':
 title = request.form.get('title')
 if title:
```

```
            project_id = dataBase().createProject(title, current_user.getId())
 return redirect(url_for('project', project_id=project_id))
 else:
 flash('Название проекта не указано', 'error')
 return render_template('createProject.html')


@app.route('/project/<int:project_id>')
@login_required
def project(project_id):
 project_info = dataBase().getProject(project_id)
 chat_id = dataBase().getProjectChat(project_id)
 chat_messages = dataBase().getChatMessages(chat_id)
 members = dataBase().getProjectMembers(project_id)
 statuses = dataBase().getAllStatuses()
 users = dataBase().getAllUsers()
 return render_template('project.html', statuses=statuses, current_user=current_user, members=members,
project=project_info, chat_messages=chat_messages, users=users, chat_id=chat_id)


@app.route('/add_member_to_project', methods=['POST'])
@login_required
def add_member_to_project():
 if request.method == 'POST':
 project_id = request.form.get('projectId')
 member_id = request.form.get('member_id')

 project = dataBase().getProject(project_id)
 member = dataBase().getUser(member_id)
 if not project or not member:
 flash('Проект или пользователь не найдены', 'error')
 return redirect(url_for('project', project_id=project_id))

 if dataBase().addMemberToProject(project_id, member_id):
 flash('Участник успешно добавлен к проекту', 'success')
 else:
 flash('Ошибка при добавлении участника к проекту', 'error')

 return redirect(url_for('project', project_id=project_id))

 return redirect(url_for('index'))


@app.route('/change_status_to_project', methods=['POST'])
@login_required
def change_status_to_project():
 if request.method == 'POST':
 project_id = request.form.get('projectId')
 status_id = request.form.get('status_id')

 if dataBase().changeProjectStatus(project_id, status_id):
 flash('Статус проекта успешно изменен', 'success')
 else:
 flash('Ошибка при изменении статуса проекта', 'error')

 return redirect(url_for('project', project_id=project_id))
 return redirect(url_for('index'))
```

```
@app.route('/delete_project', methods=['POST'])
@login_required
def delete_project():
 project_id = request.form.get('project_id')
 if project_id:
 if dataBase().deleteProject(project_id):
 flash('Проект успешно удален', 'success')
 else:
 flash('Ошибка при удалении проекта', 'error')
 return redirect(url_for('projects'))


@app.route('/send_project_message', methods=['POST'])
@login_required
def send_project_message():
 if request.method == 'POST':
 data = request.json
 project_id = data.get('project_id')
 chat_id = dataBase().getProject(project_id)
 sender_id = current_user.getId()
 message_text = data.get('text')

 dataBase().sendMessage(chat_id, sender_id, message_text)

 return jsonify({'success': True}), 200
 else:
 return jsonify({'error': 'Method not allowed'}), 405


if __name__ == '__main__':
 socketio.run(app, allow_unsafe_werkzeug=True)
```

db.py:

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

userlogin.py:

```
from dataBase import dataBase

class userlogin():
 def fromDB(self, user_id, db):
 self.__user = db.getUser(user_id)
 return self
 def create(self, user):
 self.__user = {
 'id': user[0],
 'username': user[1],
 'password': user[2],
 'email': user[3],
 'registrationdate': user[4],
 'profilepic': user[5]
 }
 return self

 def is_authenticated(self):
```

```
        return True

        def is_active(self):
        return True

        def is_anonymous(self):
        return False

        def get_id(self):
        return str(self.__user['id'])

        def getId(self):
        return str(self.__user[0])

        def get_login(self):
        return str(self.__user[1])

        def get_profilePic(self):
        return str(self.__user[5])
```

dataBase.py:

```
        import sqlite3
import psycopg2
from flask import flash


class dataBase:

 def __init__(self):
 self.conn = psycopg2.connect(
 host="localhost",
 database="msgrdb",
 user="postgres",
 password="Abdfik45",
 port="5433"
 )
 self.cur = self.conn.cursor()


 def getUser(self, user_id):
 try:
 self.cur.execute(f"select * from users where id = '{user_id}' limit 1")
 res = self.cur.fetchone()
 if not res:
 print("Пользователь не найден")
 return False
 return res
 except psycopg2.Error as e:
 print("Ошибка получения данных из БД " + str(e))
 return False


 def getUserByLogin(self, login, password):
 try:
 self.cur.execute(f"select * from users where username='{login}' and password='{password}' limit 1")
 res = self.cur.fetchone()
 if not res:
```

```
            print("Пользователь не найден")
return False
return res
except psycopg2.Error as e:
print("Ошибка получения данных из БД " + str(e))


def addUser(self, login, password):
result = False
try:
self.cur.execute(f"select username from users where username = '{login}'")
res = self.cur.fetchone()
except psycopg2.Error as e:
print("Ошибка подключения к БД" + str(e))
if not res:
try:
self.cur.execute(f"insert into users (username, password) values ('{login}', '{password}')")
self.conn.commit()
result = True
except sqlite3.Error as e:
print("Ошибка добавления данных в БД " + str(e))
return result


def addUser(self, login, password, profilepic):
result = False
try:
self.cur.execute(f"select username from users where username = '{login}'")
res = self.cur.fetchone()
except psycopg2.Error as e:
print("Ошибка подключения к БД" + str(e))
if not res:
try:
self.cur.execute(
f"insert into users (username, password, profilepic) values ('{login}', '{password}', '{profilepic}')")
self.conn.commit()
result = True
except psycopg2.Error as e:
print("Ошибка добавления данных в БД " + str(e))
return result


def CreateChat(self, user1, user2, title, chat_type):
res = False
try:
self.cur.execute(f"insert into chats (title, type) values ('{title}', '{chat_type}') returning id")
chatId = self.cur.fetchone()[0]
self.cur.execute(f"insert into chatmembers (chatid, userid) values ('{chatId}', '{user1}')")
self.cur.execute(f"insert into chatmembers (chatid, userid) values ('{chatId}', '{user2}')")
self.conn.commit()
res=True
except psycopg2.Error as e:
print("Ошибка подключения к БД" + str(e))
return res


def getUserChats(self, user_id):
```

```
 res = []
 try:
 self.cur.execute(f"select chats.id as id, chats.title as title from chats, chatmembers where chats.id=chatmembers.chatid"
 f" and chatmembers.userid='{user_id}'")
 res = self.cur.fetchall()
 except psycopg2.Error as e:
 res = "damn"
 print(str(e))
 return res


 def getChatMessages(self, chat_id):
 res = []
 try:
 self.cur.execute(
 f"SELECT m.senderid, m.message, u.username, CONCAT('/', u.profilepic) AS profilepic FROM messages m INNER
JOIN users u ON m.senderid = u.id WHERE m.chatid = '{chat_id}' order by m.timestamp")
 rows = self.cur.fetchall()
 for row in rows:
 sender_id, message, username, profile_pic = row
 res.append({'sender': username, 'text': message, 'profile_pic': profile_pic})
 except psycopg2.Error as e:
 print(str(e))
 return res


 def sendMessage(self, chat_id, sender_id, message_text):
 try:
 self.cur.execute(f"insert into messages (chatid, senderid, message) values('{chat_id}', '{sender_id}', '{message_text}')")
 self.conn.commit()
 except psycopg2.Error as e:
 print(str(e))


 def getSenderName(self, id):
 self.cur.execute(f"select username from users where id='{id}'")
 res = self.cur.fetchone()
 return res


 def getSenderPic(self, id):
 self.cur.execute(f"select profilepic from users where id='{id}'")
 res = self.cur.fetchone()
 return res


 def createProject(self, title, manager_id, status=0):
 try:
 self.cur.execute(f"INSERT INTO projects (title, status, manager_id) VALUES ('{title}', {status}, {manager_id})
RETURNING id")
 project_id = self.cur.fetchone()[0]
 self.conn.commit()
 self.cur.execute(
 f"INSERT INTO chats (title, type, project) VALUES ('{title}', 0, {project_id}) RETURNING id")
 chat_id = self.cur.fetchone()[0]
```

```
self.conn.commit()
self.cur.execute(
f"INSERT INTO chatmembers (chatid, userid) VALUES ('{chat_id}', {manager_id})")
self.conn.commit()
return project_id
except psycopg2.Error as e:
print("Ошибка при создании проекта: " + str(e))
return None


def getProjectsForUser(self, user_id):
try:
self.cur.execute(f"SELECT projects.*, project_statuses.title AS status_title FROM projects INNER JOIN
project_statuses ON projects.status=project_statuses.id WHERE projects.id IN (SELECT project_id FROM
project_members WHERE user_id='{user_id}') OR projects.manager_id='{user_id}'")
projects = self.cur.fetchall()
return projects
except psycopg2.Error as e:
print("Ошибка при получении проектов для пользователя: " + str(e))
return None


def updateProjectStatus(self, project_id, new_status):
try:
self.cur.execute(f"UPDATE projects SET status={new_status} WHERE id='{project_id}'")
self.conn.commit()
return True
except psycopg2.Error as e:
print("Ошибка при обновлении статуса проекта: " + str(e))
return False


def getAllUsers(self):
try:
self.cur.execute("SELECT * FROM users")
users = self.cur.fetchall()
return users
except psycopg2.Error as e:
print("Ошибка при получении списка пользователей из БД: ", e)
return []


def getAllStatuses(self):
try:
self.cur.execute("SELECT * FROM project_statuses")
users = self.cur.fetchall()
return users
except psycopg2.Error as e:
print("Ошибка при получении списка статусов: ", e)
return []



def getChatMessagesForProject(self, chat_id):
res = []
try:
self.cur.execute(
```

```
 f"SELECT m.senderid, m.message, u.username, CONCAT('/', u.profilepic) AS profilepic FROM messages m INNER
JOIN users u ON m.senderid = u.id WHERE m.chatid = '{chat_id}' order by m.timestamp")
 rows = self.cur.fetchall()
 for row in rows:
 sender_id, message, username, profile_pic = row
 res.append({'sender': username, 'text': message, 'profile_pic': profile_pic})
 except psycopg2.Error as e:
 print(str(e))
 return res


 def getProject(self, project_id):
 try:
 self.cur.execute(f"SELECT projects.*, project_statuses.title AS status_title FROM projects INNER JOIN
project_statuses ON projects.status=project_statuses.id WHERE projects.id = '{project_id}'")
 # self.cur.execute(f"SELECT * FROM projects WHERE id = '{project_id}'")
 project_info = self.cur.fetchone()
 return project_info
 except psycopg2.Error as e:
 print("Ошибка при получении информации о проекте из БД: ", e)
 return None


 def deleteProject(self, project_id):
 try:
 self.cur.execute(f"DELETE FROM messages WHERE chatid IN (SELECT id FROM chats WHERE
project='{project_id}')")
 self.conn.commit()
 self.cur.execute(f"DELETE FROM chatmembers WHERE chatid IN (SELECT id FROM chats WHERE
project='{project_id}')")
 self.conn.commit()
 self.cur.execute(f"DELETE FROM chats WHERE project='{project_id}'")
 self.conn.commit()
 self.cur.execute(f"DELETE FROM project_members WHERE project_id='{project_id}'")
 self.conn.commit()
 self.cur.execute(f"DELETE FROM projects WHERE id='{project_id}'")
 self.conn.commit()
 return True
 except psycopg2.Error as e:
 print("Ошибка при удалении проекта из БД: ", e)
 return False


 def getProjectChat(self, project_id):
 try:
 self.cur.execute(f"SELECT id from chats where project='{project_id}'")
 chat_id = self.cur.fetchone()[0]
 return chat_id
 except psycopg2.Error as e:
 print("Ошибка при получении сообщений проектного чата из БД: ", e)


 def addMemberToProject(self, project_id, member_id):
 try:
 self.cur.execute(
 f"INSERT INTO project_members (project_id, user_id) VALUES ('{project_id}', '{member_id}')")
 self.conn.commit()
```

```
self.cur.execute(f"SELECT id FROM chats WHERE project='{project_id}'")
chat_id = self.cur.fetchone()[0]

self.cur.execute(f"INSERT INTO chatmembers (chatid, userid) VALUES ('{chat_id}', '{member_id}')")
self.conn.commit()

return 1
except psycopg2.Error as e:
print("Ошибка при добавлении пользователя в проект: " + str(e))
return 0

def getProjectMembers(self, project_id):
members = []
try:
self.cur.execute(
f"SELECT u.username, CONCAT('/', u.profilepic) FROM users u JOIN project_members pm ON pm.user_id = u.id
JOIN projects p ON pm.project_id = p.id WHERE pm.project_id = '{project_id}'")
rows = self.cur.fetchall()
for row in rows:
username, profilepic = row
members.append({'username': username, 'profilepic': profilepic})

# Добавление информации о менеджере проекта
self.cur.execute(
f"SELECT u.username, CONCAT('/', u.profilepic) FROM users u JOIN projects p ON p.manager_id = u.id WHERE
p.id = '{project_id}'")
manager_info = self.cur.fetchone()
if manager_info:
manager_username, manager_profilepic = manager_info
members.append({'username': manager_username, 'profilepic': manager_profilepic})

except psycopg2.Error as e:
print("Ошибка при получении информации о проекте из БД: ", e)
return members


def changeProjectStatus(self, project_id, status_id):
try:
self.cur.execute(f"UPDATE projects SET status = {status_id} WHERE id = {project_id}")
self.conn.commit()
return True
except psycopg2.Error as e:
print("Ошибка при изменении статуса проекта в БД: ", e)
return False
```

base.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap" rel="stylesheet">
<link rel="stylesheet" href="{{ url_for('static', filename='/styles/styles.css') }}">
<script src="{{ url_for('static', filename='/scripts/mainscript.js') }}"></script>
{% block title %}{% endblock %}
</head>
<body>
```

```
<header>
 <a href="{{ url_for('index') }}">
{# <img src="{{ url_for('static', filename='src/logo.png') }}" alt="logo.png"> <!-- Логотип сайта -->#}
 </a>
 <nav>
 {% if current_user.is_authenticated %}
 <a href="{{ url_for('index') }}" class="navlink"> Чаты </a>
 <a href="{{ url_for('projects') }}" class="navlink"> Проекты </a>
 <a href="#" class="navlink" id="profilelogin">{{ current_user.get_login() }} <img id="profilepicture" src="{{ url_for('static',
filename=current_user.get_profilePic().lstrip('static/')) }}" alt="" class="src"> </a>
 <a href="{{ url_for('logout') }}" class="navlink">Выйти</a>
 {% else %}
 <a href="{{ url_for('authorization') }}" class="navlink">Авторизация/Регистрация</a>
 {% endif %}
 </nav>
</header>
{% block body %}{% endblock %}
</body>
</html>
```

index.html:

```
{% extends 'base.html' %}

{% block title %}
{% if current_user.is_authenticated %}
<script>
 var currentUser = {
 username: "{{ current_user.get_login() }}",
 profilepic: "{{ current_user.get_profilePic() }}"
 };

</script>
{% endif %}
<title>DaGoMessenger</title>
{% endblock %}

{% block body %}
{% if current_user.is_authenticated %}
<div class="container">
 <div class="conversation-list">
 <h2>Сообщения</h2>
 <a href="{{ url_for('create_chat') }}">
 <button id="createChatButton">Создать чат</button>
 </a>
 <ul>
 {% for chat in chats %}
 <li class="conversation-item" data-chat-id="{{ chat[0] }}" data-sender-id="{{ current_user.getId() }}">{{
chat[1] }}</li>
 {% endfor %}
 </ul>
 </div>

 <div class="chat-window">
 <h2>Чат</h2>
 <div class="chat-messages" id="chat-messages">
 </div>
 <input type="text" id="message-input" placeholder="Введите сообщение...">
 <button id="send-button">Отправить</button>
```

```
 </div>
</div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.3.2/socket.io.min.js"></script>
{% else %}
<h1>Войдите в систему</h1>
{% endif %}
{% endblock %}
```

authorization.html:

```
{% extends 'base.html' %}


{% block title %}
<title>Авторизация</title>
{% endblock %}


{% block body %}
 <form action="" method="post">
 <div class="loginform">
 <div class="loginform">
 <h1>Авторизация</h1>
 {% with messages = get_flashed_messages(with_categories=true) %}
 {% for category, message in messages %}
 <div id="formAlert" class="alert alert-{{ category }}">
 {{ message }}
 </div>
 {% endfor %}
 {% endwith %}
 <p>Логин</p>
 <input type="text" name="login">
 <p>Пароль</p>
 <input type="password" name="pass">
 <p><input type="submit" value="Войти"></p>
 <p>Не зарегистрированы? <a href="{{ url_for('registration') }}">Регистрация</a></p>
 <p><a href="{{ url_for('index') }}">Вернуться на главную</a></p>
 </div>
 </div>
 </form>

{% endblock %}
```

registration.html:

```
{% extends 'base.html' %}


{% block title %}
<title>Регистрация</title>
{% endblock %}


{% block body %}
<form action="" method="post" enctype="multipart/form-data">
 <div class="loginform">
 <div class="loginform">
 <h1>Регистрация</h1>
 {% with messages = get_flashed_messages(with_categories=true) %}
```

```
 {% for category, message in messages %}
 <div id="formAlert" class="alert alert-{{ category }}">
 {{ message }}
 </div>
 {% endfor %}
 {% endwith %}
 <p>Логин</p>
 <input type="text" name="login">
 <p>Пароль</p>
 <input type="password" name="pass">
 <p>Повторите пароль</p>
 <input type="password" name="repeatPass">
 <p>Изображение профиля:</p>
 <input type="file" name="profilepic">
 <p><input type="submit" value="Зарегистрироваться"></p>
 <p>Уже зарегистрированы? <a href="{{ url_for('authorization') }}">Авторизация</a></p>
 <p><a href="{{ url_for('index') }}">Вернуться на главную</a></p>
 </div>
 </div>
</form>
{% endblock %}
```

createChat.html:

```
{% extends 'base.html' %}

{% block title %}
<title>Создание чата</title>
{% endblock %}
{% block body %}
<form action="{{ url_for('create_chat') }}" method="post">
 <div class="loginform">
 <div class="loginform">
 <h1>Создать чат</h1>
 {% with messages = get_flashed_messages(with_categories=true) %}
 {% for category, message in messages %}
 <div id="formAlert" class="alert alert-{{ category }}">
 {{ message }}
 </div>
 {% endfor %}
 {% endwith %}
 <input type="hidden" name="currentUser" value="{{ current_user.getId() }}">
 <p>Собеседник</p>
 <select name="partner">
 {% for user in users %}
 <option value="{{ user.id }}">{{ user.username }}</option>
 {% endfor %}
 </select>
 <p>Название чата</p>
 <input type="text" name="title">
 <p><input type="submit" value="Создать"></p>
 <p><a href="{{ url_for('index') }}">Вернуться на главную</a></p>
 </div>
 </div>
</form>
{% endblock %}
```

styles.css:

```
            html {
 width: 1280px;
 height: 100%;
 margin-left: auto;
 margin-right: auto;
}

body {
 height: 100%;
}

header {
 padding: 10px;
 text-align: center;
 display: flex;
 flex: 0;
 width: 100%;
}

header img {
 float: left;
 width: 60%;
 margin-top: 30px;
}

header img:hover {
 opacity: 80%;
 transition-duration: 1s;
}

nav {
 float: right;
 margin-top: 55px;
 width: 75%;
}

.navlink {
 text-decoration: none;
 font-family: 'Roboto', sans-serif;
 font-size: 24px;
 color: rgb(53, 53, 53);
 padding: 25px;
}

.navlink:hover {
 opacity: 50%;
 transition-duration: 1s;
}

#profilelogin img {
 object-fit: cover;
 width: 50px;
 height: 50px;
 margin-left: 5%;
 border-radius: 50%;
 float: none;
}
```

```
section {
 flex: 1;
 padding-top: 20px;
 text-align: center;
}

.mainposter {
 width: 100%;
 height: 500px;
 background-color: rgb(37, 37, 37);
 position: relative
}

.mainposter img {
 position: absolute;
 top: 50%;
 left: 10%;
 transform: translate(0, -50%);
 object-fit: cover;
 width: 80%;
 height: 80%;
}

.mainposter img:hover {
 opacity: 50%;
 transition-duration: 1s;
}

.mainpostertext {
 position: absolute;
 top: 86%;
 left: 10%;
 transform: translate(0, -50%);
 object-fit: cover;
 width: 80%;
 height: 8%;
 text-decoration: none;
 font-family: 'Roboto', sans-serif;
 font-size: 24px;
 color: white;
 margin-bottom: 0;
 background-color: #000000a8;
}

.mainpostertext:hover {
 color: #0afff3;
 opacity: 50%;
 transition-duration: 1s;
}

h2 {
 font-family: 'Roboto', sans-serif;
 font-weight: normal;
 float: left;
 font-size: 24px;
 width: 100%;
 text-align: left;
```

```
 color: rgb(53, 53, 53);
 padding-left: 40px;
}

.column {
 width: 45%;
 display: inline-block;
 vertical-align: top;
 padding: 0 2%;
}

ul {
 list-style: none;
 padding: 0;
}

li {
 margin-bottom: 10px;
}

footer {
 flex: 2;
 /*display: flexbox;*/
 margin-top: 50px;
 width: 100%;
 height: 300px;
 background-color: #333;
 color: #fff;
 text-align: center;
}

.ploginform {
 display: flex;
 width: 100%;
 height: 100%;
 justify-content: center;
 align-items: center;
}

/* Стили для формы авторизации */
.loginform {
 max-width: 400px;
 margin: 50px auto;
 padding: 25px;
 background-color: #fff;
 border-radius: 10px;
 box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.loginform h1 {
 font-size: 28px;
 text-align: center;
 margin-bottom: 20px;
 color: #333;
}

.loginform p {
```

```
 font-size: 18px;
 margin-bottom: 10px;
}

.loginform input[type="text"],
.loginform input[type="password"] {
 width: 100%;
 padding: 10px;
 margin-bottom: 20px;
 border: 1px solid #ccc;
 border-radius: 5px;
}

.loginform input[type="submit"] {
 width: 100%;
 padding: 10px;
 border: none;
 border-radius: 5px;
 background-color: #b598f7;
 color: white;
 font-size: 20px;
 cursor: pointer;
}

.loginform input[type="submit"]:hover {
 background-color: #6645a0;
}

#formAlert {
 color: red;
 font-size: 16px;
 text-align: center;
}

.profile-pic {
 width: 40px;
 height: 40px;
 border-radius: 50%;
 margin-right: 10px;
}

.message-text {
 font-size: 14px;
}

.sender-name {
 font-weight: bold;
 margin-right: 5px;
}

/* Стили для списка чатов */

#createChatButton {
 padding: 10px 20px;
 margin-bottom: 10px;
 background-color: #b598f7;
```

```
 border: none;
 border-radius: 5px;
 color: white;
 cursor: pointer;
}

#createChatButton:hover {
 background-color: #6645a0;
}

.conversation-list {
 flex: 1;
 margin-right: 20px;
}

.conversation-list h2 {
 color: #4d4d4d;
}

.conversation-list ul {
 list-style-type: none;
 padding: 0;
}

.conversation-item {
 cursor: pointer;
 padding: 10px;
 margin-bottom: 10px;
 background-color: #fff;
 border: 1px solid #ccc;
 border-radius: 5px;
}

.conversation-item:hover {
 background-color: #f2f2f2;
}

.chat-window {
 flex: 2;
}

.chat-window h2 {
 color: #4d4d4d;
}

.chat-messages {
 max-height: 400px;
 overflow-y: auto;
 background-color: #fff;
 border: 1px solid #ccc;
 border-radius: 5px;
 padding: 10px;
 margin-bottom: 20px;
}

.profile-pic {
 width: 40px;
```

```
 height: 40px;
 border-radius: 50%;
 margin-right: 10px;
}

.message {
 margin-bottom: 10px;
}

.projects-container {
 display: flex;
 flex-wrap: wrap; /* Перенос на новую строку при нехватке места */
}

.projects-list {
 list-style-type: none;
 padding: 0;
 margin: 0;
 display: flex;
 flex-wrap: wrap;
}

.projectchat {
 margin-top: 20px;
}

.projects-item {
 width: 300px;
 margin: 10px;
 border: 1px solid #ccc;
 border-radius: 5px;
}

.projects-info {
 padding: 10px;
 font-family: 'Roboto', sans-serif;
}

.projects-actions {
 padding: 10px;
 display: flex;
 justify-content: space-between;
}

#message-input {
 width: calc(100% - 100px);
 padding: 10px;
 margin-right: 10px;
 border: 1px solid #ccc;
 border-radius: 5px;
}


.project-container {
 display: flex;
 flex-direction: row;
 justify-content: space-between;
```

```
}

.project-info,
.project-actions,
.project-chat {
 font-family: 'Roboto', sans-serif;
 width: 30%;
 margin: 20px;
}

.project-info,
.project-actions {
 font-family: 'Roboto', sans-serif;
 background-color: #f9f9f9;
 padding: 15px;
}

.project-chat {
 font-family: 'Roboto', sans-serif;
 background-color: #fff;
 padding: 15px;
}

.members-list {
 list-style-type: none;
 padding: 0;
}

.members-list li {
 font-family: 'Roboto', sans-serif;
 margin-bottom: 5px;
}


#send-button {
 padding: 10px 20px;
 background-color: #b598f7;
 border: none;
 border-radius: 5px;
 color: white;
 cursor: pointer;
}

#send-button:hover {
 background-color: #6645a0;
}

.project-button{
 padding: 10px 20px;
 background-color: #b598f7;
 border: none;
 border-radius: 5px;
 color: white;
 cursor: pointer;
}

.project-button:hover{
```

```
 background-color: #6645a0;
}

@media screen and (max-width: 980px) {
 html {
 width: 100%;
 margin: 0;
 padding: 0;
 }

 body {
 margin: 0;
 width: 100%;
 }

 footer {
 width: 100%;
 }

 header {
 /* width: 100%; */
 display: inline;
 }

 header img {
 float: none;
 position: relative;
 left: 10%;
 width: 80%;
 padding-bottom: 20px;
 }

 nav {
 float: none;
 width: 80%;
 margin: 0 auto;
 }

 .navlink {
 display: inline-flex;
 width: 100%;
 font-size: 20px;
 color: rgb(53, 53, 53);
 padding-top: 30px;
 padding: 20px;
 }

 .mainposter img {
 left: 10%;
 }

 .mainpostertext {
 position: absolute;
 top: 83%;
 left: 10%;
 transform: translate(0, -50%);
 object-fit: cover;
```

```
 width: 80%;
 height: 15%;
 text-decoration: none;
 font-family: 'Roboto', sans-serif;
 font-size: 24px;
 color: white;
 margin: auto;
 background-color: #000000a8;
 }

 .block-container {
 display: inline;
 }

 .block {
 width: 100%;
 margin: 0;
 }

 .down {
 padding: 20px 0 0 0;
 }

 .down ul {
 padding: 20px;
 margin: 0;
 width: 100%;
 font-size: 14px;
 }

 h2 {
 padding: 0;
 float: none;
 text-align: center;
 }

 .copyright {
 padding-top: 20px;
 }
}

.container {
 display: flex;
 flex-direction: row;
}

/* Стили для списка бесед */
.conversation-list {
 width: 200px;
 padding: 20px;
 border-right: 1px solid #ccc;
}

.conversation-list h2 {
 margin-top: 0;
}
```

```
.conversation-list ul {
 list-style-type: none;
 padding: 0;
 margin: 0;
}

.conversation-item {
 padding: 10px;
 border-bottom: 1px solid #ccc;
 cursor: pointer;
}

.conversation-item:hover {
 background-color: #f0f0f0;
}

/* Стили для окна чата */
.chat-window {
 flex-grow: 1;
 padding: 20px;
 border-left: 1px solid #ccc;
}

.chat-window h2 {
 margin-top: 0;
}

.chat-window .chat-messages {
 height: 300px;
 overflow-y: auto;
 border: 1px solid #ccc;
 padding: 10px;
 font-family: 'Roboto', sans-serif;
}

.chat-window input[type="text"] {
 width: calc(100% - 70px);
 margin-top: 10px;
 padding: 5px;
 border: 1px solid #ccc;
}

.chat-window button {
 margin-top: 10px;
 padding: 5px 10px;
 border: none;
 background-color: #007bff;
 color: #fff;
 cursor: pointer;
}

@media screen and (max-width: 768px) {
 .container {
 flex-direction: column;
 align-items: stretch;
 }
```

```
.conversation-list,
.chat-window {
margin-right: 0;
}

#message-input {
width: calc(100% - 80px);
}
}

@media screen and (max-width: 480px) {
.container {
padding: 0 10px;
}

#message-input {
width: calc(100% - 60px);
}
}
```

projects.html:

```
{% extends 'base.html' %}
{% block title %}
<title>Проекты</title>
{% endblock %}
{% block body %}
<div class="projects-container">
<h2>
Ваши проекты
<a href="{{ url_for('create_project') }}">
<button class="project-button" id="createProjectButton">Создать проект</button>
</a>
</h2>

<ul class="projects-list">
{% for project in projects %}
<li class="projects-item" data-project-id="{{ project.id }}">
<div class="projects-info">
<h3>{{ project[5] }}</h3>
<p>Статус: {{ project[6] }}</p>
</div>
<div class="projects-actions">
<a href="{{ url_for('project', project_id=project[0]) }}">
<button class="project-button" data-project-id="{{ project[0] }}">Просмотреть</button>
</a>
<form action="{{ url_for('delete_project') }}" method="post">
<input type="hidden" name="project_id" value="{{ project[0] }}">
<button type="submit" class="project-button" data-project-id="{{ project[0] }}">Удалить</button>
</form>
</div>
</li>
{% endfor %}
</ul>
</div>
{% endblock %}
```

project.html:

```
{% extends 'base.html' %}

{% block title %}

<title>Проект</title>
{% endblock %}

{% block body %}
<input type="hidden" id="project-id" value="{{ chat_id }}">
<div class="project-container">
 <div class="project-info">
 <h2>Информация о проекте</h2>
 <p>Название проекта: {{ project[5] }}</p>
 <p>Статус проекта: {{ project[6] }}</p>
 <p>Участники проекта:</p>
 <ul class="members-list">
 {% for member in members %}
 <li>{{ member.username }} <img class="profile-pic" src="{{ member.profilepic }}" alt="{{ member.profilepic }}"></li>
 {% endfor %}
 </ul>
 </div>

 <div class="project-actions">
 <h2>Действия</h2>
 <form action="{{ url_for('add_member_to_project') }}" method="post">
 <input name="projectId" type="hidden" value="{{ project[0] }}">
 <p>Добавить участника в проект:</p>
 <select name="member_id">
 {% for user in users %}
 <option value="{{ user[0] }}">{{ user[1] }}</option>
 {% endfor %}
 </select>
 <input type="submit" class="project-button" value="Добавить">
 </form>
 <form action="{{ url_for('change_status_to_project') }}" method="post">
 <input name="projectId" type="hidden" value="{{ project[0] }}">
 <p>Сменить статус:</p>
 <select name="status_id">
 {% for status in statuses %}
 <option value="{{ status[0] }}">{{ status[1] }}</option>
 {% endfor %}
 </select>
 <input type="submit" class="project-button" value="Изменить">
 </form>
 </div>
 <div class="project-chat">
 <div class="chat-window" >
 <h2>Чат проекта</h2>
 <div class="chat-messages" id="chat-messages" data-project-id="{{ project[0] }}">
 </div>
 <input type="text" id="message-input" placeholder="Введите сообщение...">
 <button id="send-button">Отправить</button>
 </div>
 </div>
</div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.3.2/socket.io.min.js"></script>
{% endblock %}
```

socketServer.js:

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const cors = require('cors');

const app = express();
const server = http.createServer(app);

// Подключаем cors middleware
app.use(cors());

const io = new socketIo.Server(server, {
 cors: {
 origin: '*',
 methods: ['GET', 'POST'],
 allowedHeaders: ['Content-Type'],
 credentials: true,
 }
});

io.on('connection', (socket) => {
 console.log('A user connected');

 socket.on('disconnect', () => {
 console.log('User disconnected');
 });

 socket.on('message_sent', (data) => {
 console.log('Message sent:', data);
 // Обрабатываем полученное сообщение и рассылаем его другим подключенным клиентам
 io.emit('new_message', data);
 });
});

const PORT = process.env.PORT || 3000;
server.listen(PORT, () => {
 console.log(`Server is running on port ${PORT}`);
});
```

mainscript.js:

```
var modal = document.getElementById("createChatModal");
var btn = document.getElementById("createChatButton");
var span = document.getElementsByClassName("close")[0];

document.addEventListener('DOMContentLoaded', function () {
 var socket = io.connect('http://' + document.domain + ':' + 3000);
 var activeChatId = null; // Инициализация переменной activeChatId

 // Получаем project_id из скрытого поля, если оно существует
 var projectIdField = document.getElementById('project-id');
 var projectId = projectIdField ? projectIdField.value : null;

 // Если projectId определен, вызываем функцию loadChatMessages
 if (projectId) {
 loadChatMessages(projectId);
 }
```

```
socket.on('connect', function () {
console.log('Connected');
});
// Добавление нового сообщения в окно чата при получении через сокеты
socket.on('new_message', function (data) {
var chatMessagesDiv = document.getElementById('chat-messages');
var messageContainer = document.createElement('div');
var profilePicImg = document.createElement('img');
profilePicImg.src = data.profile_pic ;
profilePicImg.alt = 'Profile Picture';
profilePicImg.className = 'profile-pic';
console.log(data)
var messageTextSpan = document.createElement('span');
messageTextSpan.textContent = data.username + ': ' + data.message_text;
messageContainer.appendChild(profilePicImg);
messageContainer.appendChild(messageTextSpan);
chatMessagesDiv.appendChild(messageContainer);
chatMessagesDiv.scrollTop = chatMessagesDiv.scrollHeight;
});

// Функция отправки сообщения
function sendMessage() {
var messageInput = document.getElementById('message-input');
var messageText = messageInput.value.trim();

if (activeChatId !== null && messageText !== '') {
// Отправка через AJAX
var xhr = new XMLHttpRequest();
xhr.open('POST', '/send_message', true);
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onreadystatechange = function () {
if (xhr.readyState == 4 && xhr.status == 200) {
messageInput.value = '';
}
};
xhr.send(JSON.stringify({chat_id: activeChatId, text: messageText}));

// Отправка через сокеты
var senderId = document.querySelector('.conversation-item[data-chat-id="' + activeChatId + '"]').dataset.senderId;
socket.emit('message_sent', {'chat_id': activeChatId, 'sender_id': senderId, 'message_text': messageText, 'profile_pic':
currentUser.profilepic, 'username': currentUser.username});
}
}

// Назначение обработчика события на кнопку "Отправить"
var sendButton = document.getElementById('send-button');
sendButton.addEventListener('click', function () {
sendMessage(); // Вызываем функцию sendMessage() при клике на кнопку "Отправить"
});
// Инициализация загрузки сообщений при открытии чата
function loadChatMessages(chatId) {
activeChatId = chatId;
var chatMessagesDiv = document.getElementById('chat-messages');
chatMessagesDiv.innerHTML = '';
var xhr = new XMLHttpRequest();
```

```
xhr.open('GET', '/load_chat_messages?chat_id=' + activeChatId, true);
xhr.onreadystatechange = function () {
if (xhr.readyState == 4 && xhr.status == 200) {
var messages = JSON.parse(xhr.responseText);
for (var i = 0; i < messages.length; i++) {
var message = messages[i];
var messageContainer = document.createElement('div');
var profilePicImg = document.createElement('img');
profilePicImg.src = message.profile_pic;
profilePicImg.alt = 'Profile Picture';
profilePicImg.className = 'profile-pic';
var messageTextSpan = document.createElement('span');
messageTextSpan.textContent = message.sender + ': ' + message.text;
messageContainer.appendChild(profilePicImg);
messageContainer.appendChild(messageTextSpan);
chatMessagesDiv.appendChild(messageContainer);
}
}
};
xhr.send();
}

// Вызов loadChatMessages при клике на элемент чата
var conversationItems = document.querySelectorAll('.conversation-item');
conversationItems.forEach(function (item) {
item.addEventListener('click', function () {
var chatId = item.dataset.chatId;
loadChatMessages(chatId);
});
});

// Также можно добавить обработку нажатия клавиши Enter для отправки сообщения
var messageInput = document.getElementById('message-input');
messageInput.addEventListener('keydown', function (event) {
if (event.key === 'Enter') {
sendMessage();
}
});
```