**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ,**

**СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

**Санкт-Петербургский колледж телекоммуникаций им. Э.Т. Кренкеля**

**Предметная (цикловая) комиссия** *Информатики и программирования в компьютерных системах*

Допустить к защите

Заместитель директора

по учебной работе

\_\_\_\_\_\_\_\_\_\_ Н.В. Калинина

14 июня 2023 г.

|  |  |  |  |
| --- | --- | --- | --- |
| **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА** | | | |
| Тема | **Разработка корпоративной системы коммуникации** | | |
|  |  | | |
| Вид выпускной квалификационной работы | | | *Дипломный проект(работа)* |
|  | | | *(дипломный проект, дипломная работа)* |
|  | | | |
| Специальность | | *09.02.05 Прикладная информатика (по отраслям)* | |
| Квалификация | | *Техник -программист* | |

|  |  |  |  |
| --- | --- | --- | --- |
| Дипломник | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Преснухина Л. И. |
|  |  | | 10 июня 2023 г. |
| Руководитель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Петров П.П. |
|  |  | | 10 июня 2023 г. |
| Консультант по экономической части | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Афанасьева Н.И. |
|  |  | | 10 июня 2023 г. |
| Консультант по охране труда | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Ожерельева Н.А. |
|  |  | | 10 июня 2023 г. |

Санкт-Петербург

2023

|  |  |  |
| --- | --- | --- |
| Работа написана мною и не содержит неправомерных заимствований | | |
| 10.06.2023 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *(дата)* | | *(подпись студента)* |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
| Текст ВКР размещен в электронно-библиотечной системе университета | | |
| Руководитель отдела комплектования библиотеки | |  |
|  | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  *(ФИО)* |
| 10.06.2023 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *(дата)* | | *(подпись)* |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
|  | |  |
| Коэффициент оригинальности ВКР | | \_\_\_\_\_\_\_ % |
|  | |  |
| Проверил: | Председатель П(Ц)К № 5 Кривоносова Н.В. | |
|  | | *(Должность, ФИО)* |
|  | |  |
| 10.06.2023 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| *(дата)* | | *(подпись)* |

**ОГЛАВЛЕНИЕ**

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ3**

**ВВЕДЕНИЕ4**

**1. ПОСТАНОВКА ЗАДАЧИ5**

* 1. **Обзор аналогов5**
  2. **SWOT-анализ9**
  3. **PEST-анализ10**
  4. **Формирование требований к продукту11**
     1. **Бизнес требования11**
     2. **Пользовательские требования12**
     3. **Функциональные требования12**
     4. **Нефункциональные требования13**
     5. **Ограничения13**
     6. **Требования к интерфейсу13**
     7. **Требования к данным14**
  5. **Программные средства разработки14**
  6. **Аппаратные средства разработки14**

**2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ16**

* 1. **Архитектура системы16**
  2. **Моделирование основных сценариев системы16**
  3. **Проектирование графического интерфейса пользователя25**
  4. **Проектирование и разработка модели данных 27**

**3. РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА30**

* 1. **Реализация серверной части30**
  2. **Реализация клиентской части 31**

**4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ35**

* 1. **История изменений 35**
  2. **Терминология тестирования программного обеспечения35**
  3. **Стратегия тестирования35**
  4. **Определение объектов тестирования35**
  5. **Архитектура тестируемой системы36**
  6. **Описание процесса тестирования37**

**ЗАКЛЮЧЕНИЕ38**

**СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ39**

**ПРИЛОЖЕНИЕ. ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ40**

**ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ**

ПО – программное обеспечение;

БД – база данных;

СУБД – система управления базами данных;

ОС – операционная система;

SQL – Structured Query Language, язык структурированных запросов;SSD – Solid-State Drive, твердотельный накопитель;

HDD – Hard Disk Drive, жесткий диск.

**ВВЕДЕНИЕ**

Современные тенденции рынка информационных технологий – мобильность сотрудников, возрастающее количество удаленно работающего персонала, требуют от компаний гибкости в организации бизнес-процессов. Корпоративные социальные сети помогают компаниям решить эту задачу и, при грамотном использовании, повышают продуктивность работы сотрудников.

Коммуникации внутри организации выполняют множество функций, связанных одной целью – объединить в единое целое организованную деятельность внутри организации, создание эффективной рабочей обстановки на предприятии.

Корпоративная коммуникация — это совокупность действий, связанных с управлением и организацией всех внутренних и внешних коммуникаций, направленных на формирование благоприятной точки зрения среди заинтересованных сторон, от которых зависит компания. Это сообщения, распространяемые корпоративной организацией, органом или учреждением для своей аудитории, такой как сотрудники, средства массовой информации, партнеры по каналам и широкая общественность. Организации стремятся донести одно и то же послание до всех заинтересованных сторон, обеспечить согласованность действий, доверие и этичность.

Целью написания данной курсовой работы является подробно ознакомиться с процессом создания продукта для обеспечения корпоративной коммуникации, а именно корпоративного мессенджера, по завершению – создать продукт.

Объект исследования: корпоративные коммуникации.

Предмет исследования: корпоративный мессенджер.

Для достижения намеченной цели были поставлены следующие задачи:

1. Произвести анализ объекта и предмета исследования.

2. Определить требования к портативному мессенджеру для обеспечения коммуникации сотрудников организации.

3. Реализовать программное приложение мессенджера для корпоративных коммуникаций.

4. Выполнить комплексную оценку реализованной информационной системы.

При написании данной курсовой работы были использованы следующие методы исследования: анализ литературы, сравнение, синтез информации, системный анализ, математическое моделирование СУБД, информационное моделирование данных. В коде применялись стандарты объектно-ориентированного подхода к проектированию и программированию, сетевые технологии, desktop - приложение полностью реализовано на языке С# с выполнение операторов SQL-запросов.

**1. ПОСТАНОВКА ЗАДАЧИ**

* 1. **Обзор аналогов**

Перед тем, как приступить к реализации программного продукта, необходимо провести анализ конкурентов на рынке. С его помощью можно сделать выводы о том, как должен создаваться проект на основе достоинств и недостатков уже существующих ресурсов.

В качестве первого аналога хотелось бы рассмотреть корпоративный мессенджер VK Teams. Данный сервис входит в корпоративный набор инструментов VK Workspace, который является частью социальной сети VK. В рассматриваемой системе представлены такие функции, как синхронизация контактов, создание групп или каналов, переписка, видео- и аудиозвонки.

Внешний вид VK Teams представлен на рис. 1.1.1.

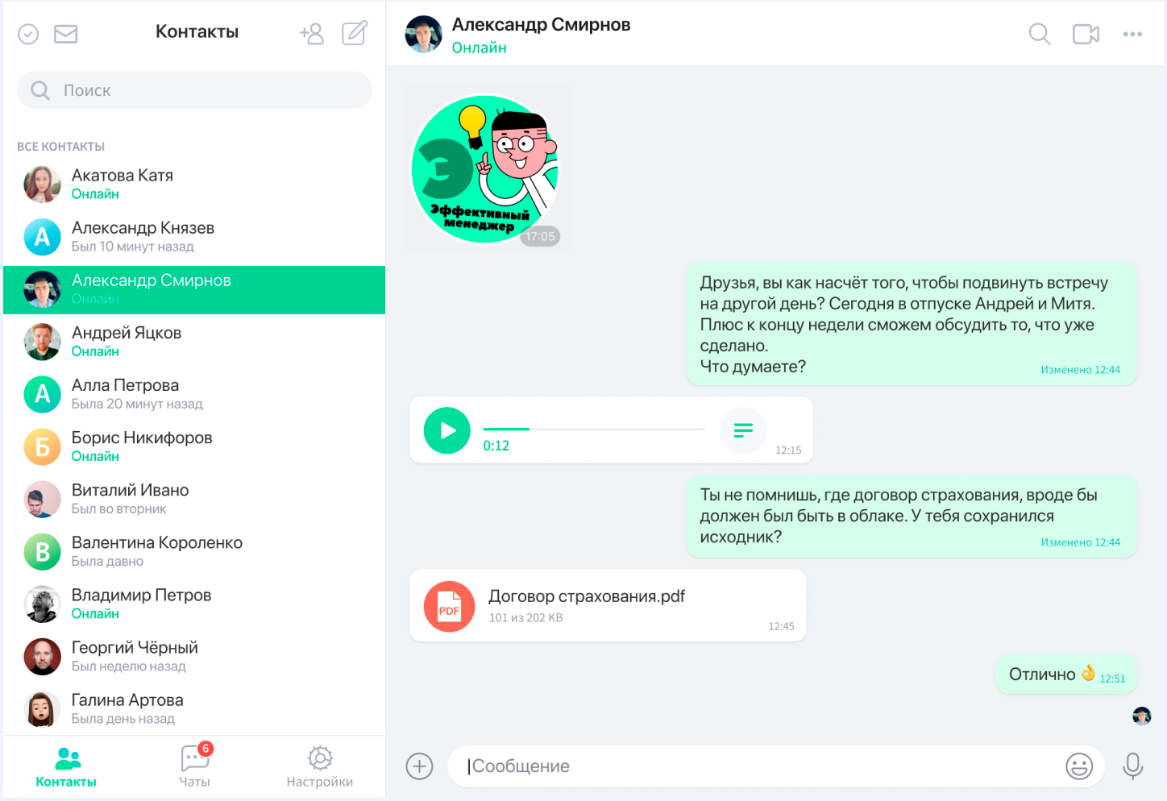


Рисунок 1.1.1. Внешний вид VK Teams

В качестве достоинства данной системы можно отметить возможность создания задачи в таск-менеджер прямо из переписки (рис. 1.1.2). Также в VK Teams можно добавить исполнителя, указать дедлайн и обозначить приоритет задания и управлять статусом задачи (рис. 1.1.3).

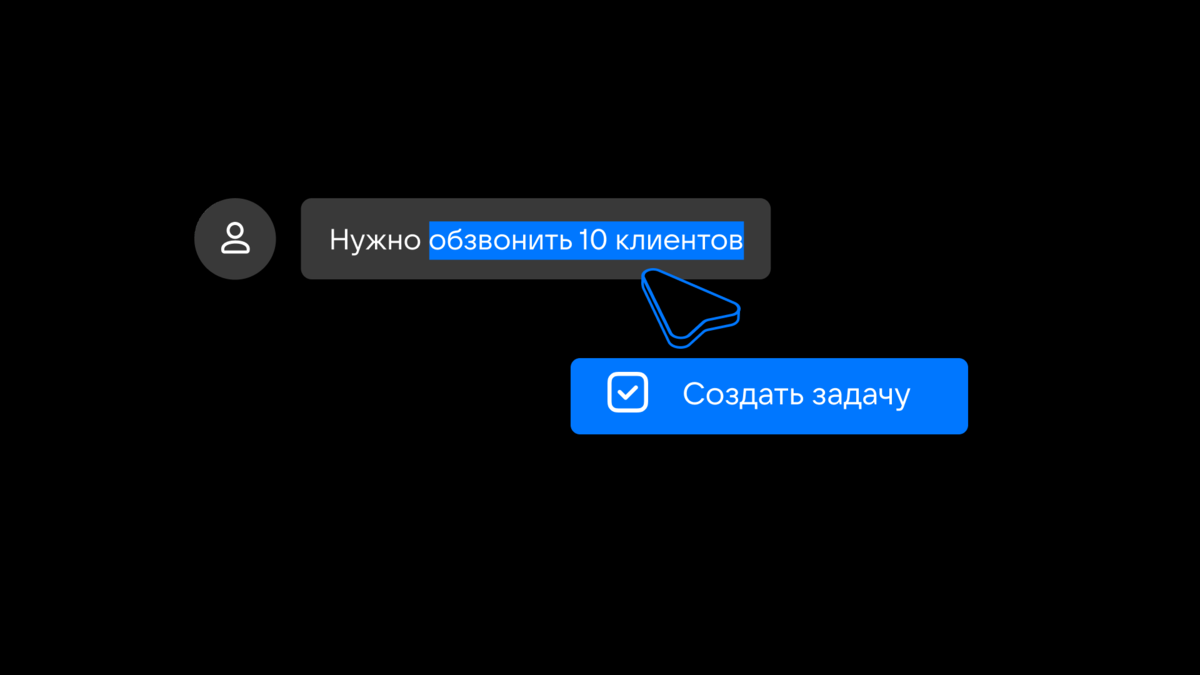


Рисунок 1.1.2. Добавление задачи в таск-менеджер из переписки

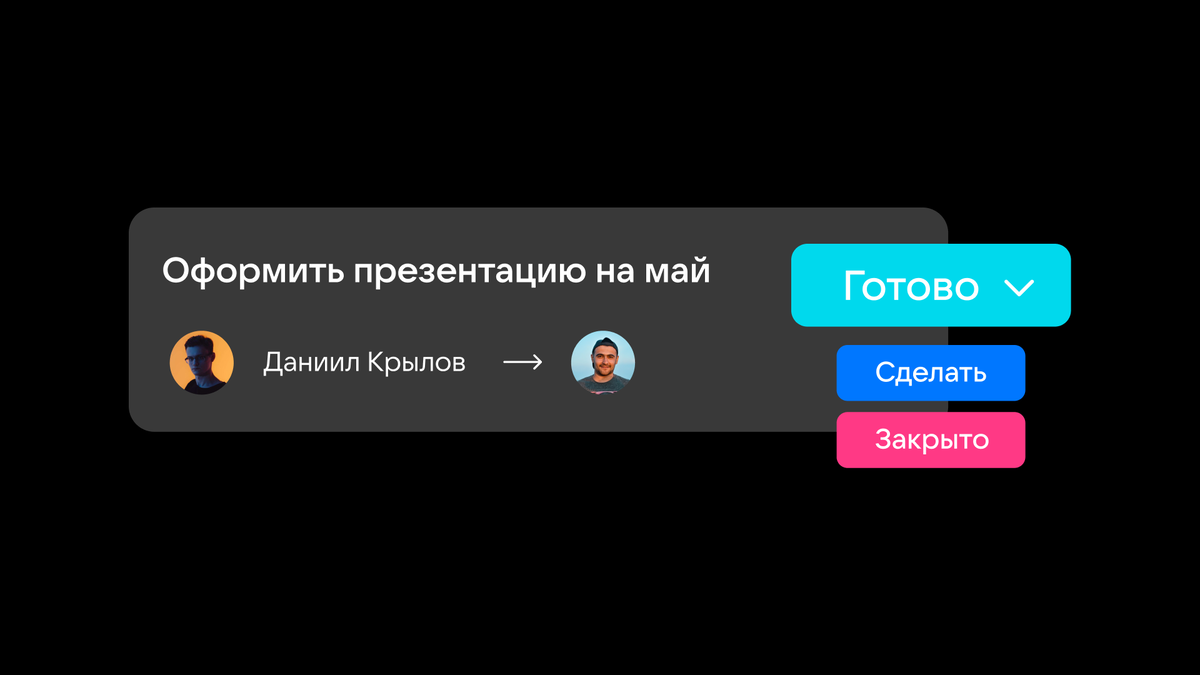


Рисунок 1.1.3

Также достоинством можно считать планирование созвонов во встроенном календаре. Календарь предлагает интервалы, когда все участники свободны, а созвон сам добавляется в расписание каждого коллеги (рис. 1.1.4).

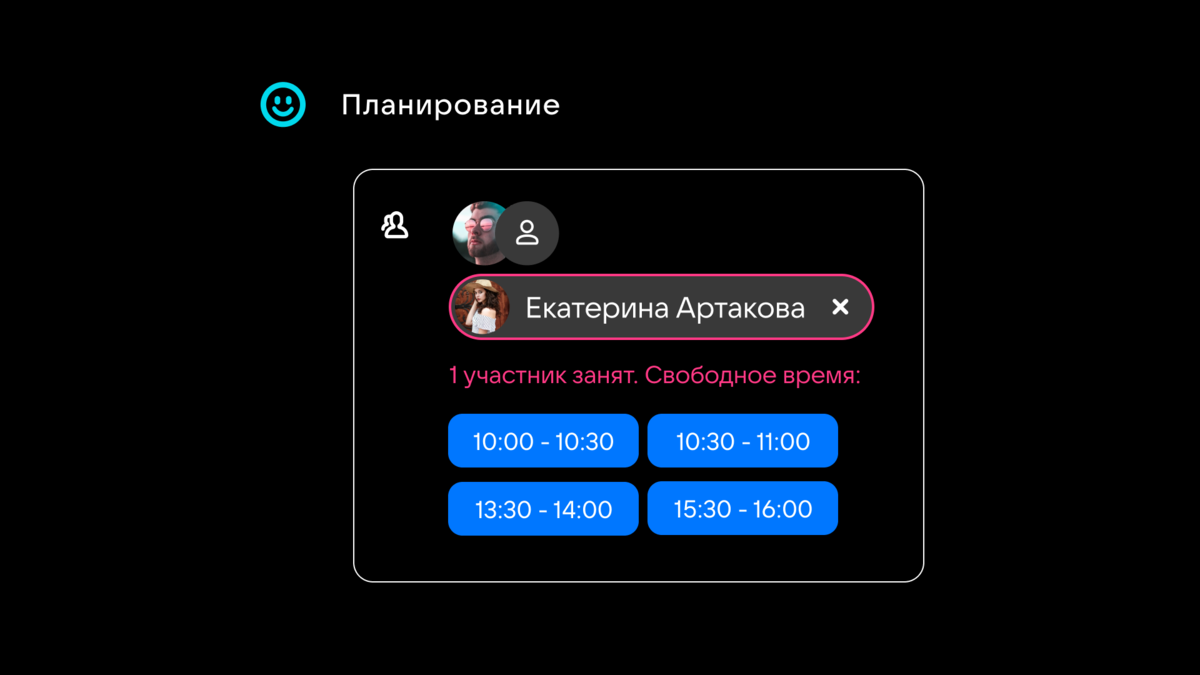


Рисунок 1.1.4

Однако среди недостатков этого сервиса пользователи выделяют то, что приложение не присылает уведомления о новых сообщениях, что вынуждает пользователей каждый раз открывать приложение.

Стоимость подписки на сервис варьируется от 1690 руб. (подписка для 1-30 пользователей на 1 месяц) до 83 900 руб. (101-300 пользователей на 1 год).

Вторым аналогом можно назвать Microsoft Teams. В данном сервисе реализованы следующие функции: создание групп или каналов, переписка, видео- и аудиозвонки, демонстрация экранов.

Внешний вид сервиса представлен на рисунке 1.1.5.

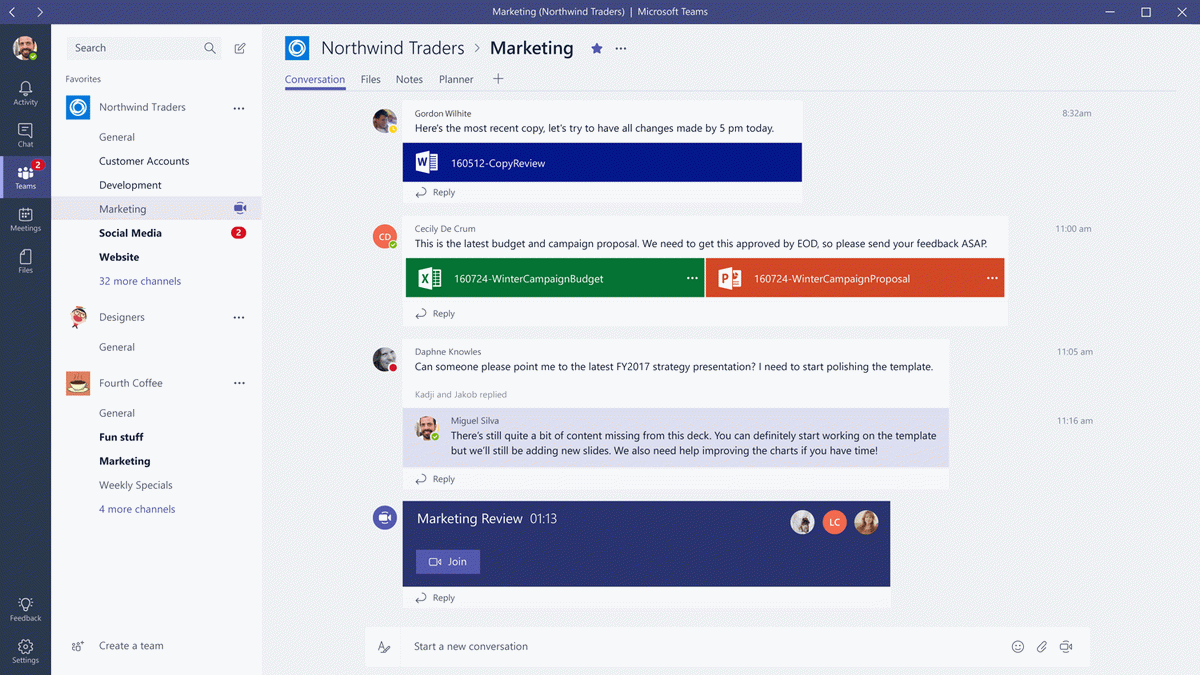


Рисунок 1.1.5. Внешний вид Microsoft Teams

Одним из самых главных достоинств Microsoft Teams можно назвать совместное редактирование документов Word, Excel, Power Point в режиме реального времени. Эта функция позволяет просматривать изменения, вносить правки и оставлять комментарии, не покидая платформу (рис. 1.1.6).

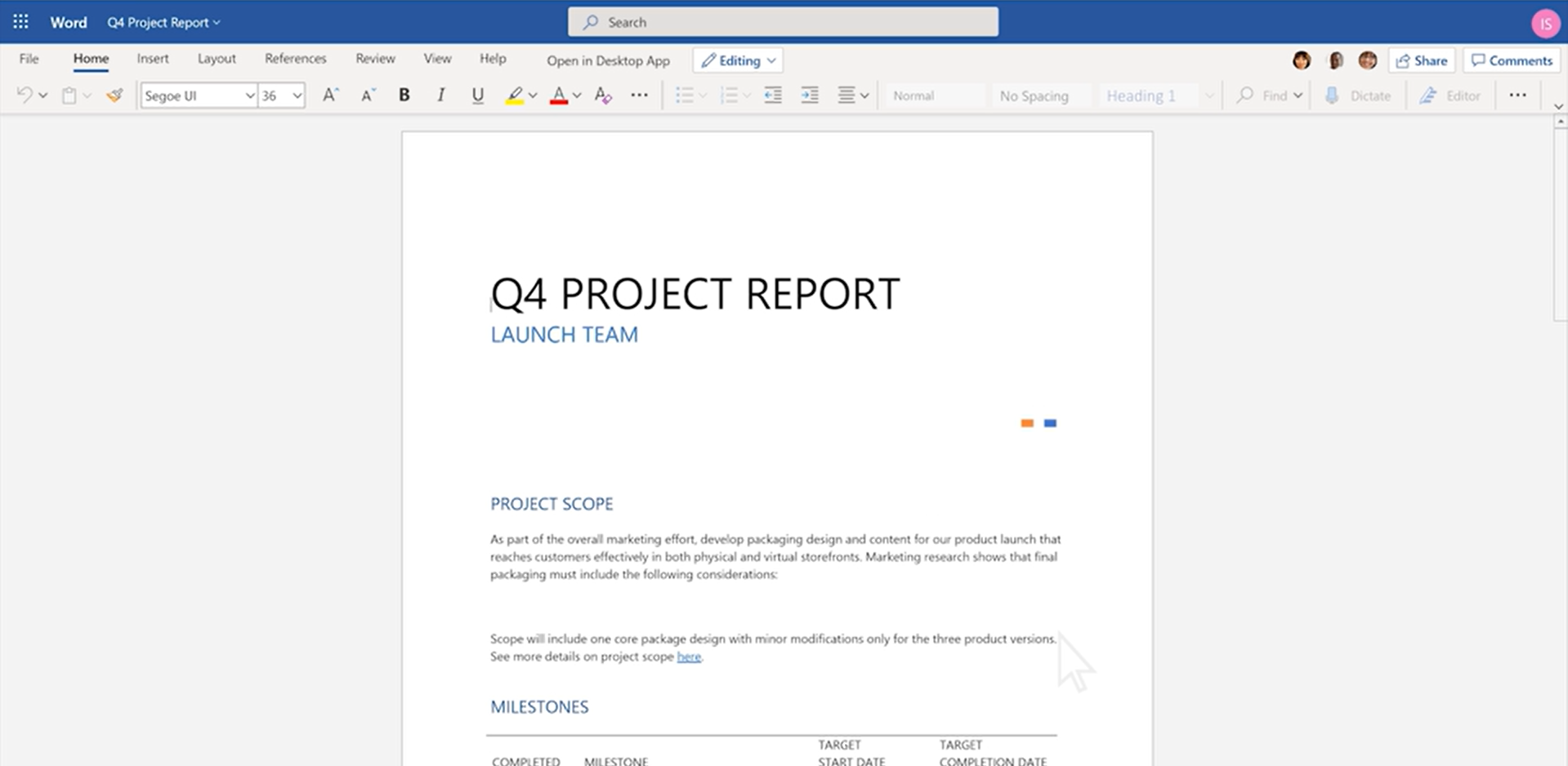


Рисунок 1.1.6

Среди достоинств также нужно выделить интеграцию с другими инструментами. Microsoft Teams создает единое место для работы над проектами и обмена информацией, интегрируясь с другими сервисами и инструментами Microsoft (SharePoint, OneDrive, Outlook и др.).

В качестве третьего аналога хотелось бы назвать TrueConf. В данном корпоративном мессенджере реализованы следующие функции: создание и планирование онлайн-конференций, обмен сообщениями и файлами, демонстрация экрана, трансляция онлайн-мероприятий.

Внешний вид сервиса представлен на рис. 1.1.7.

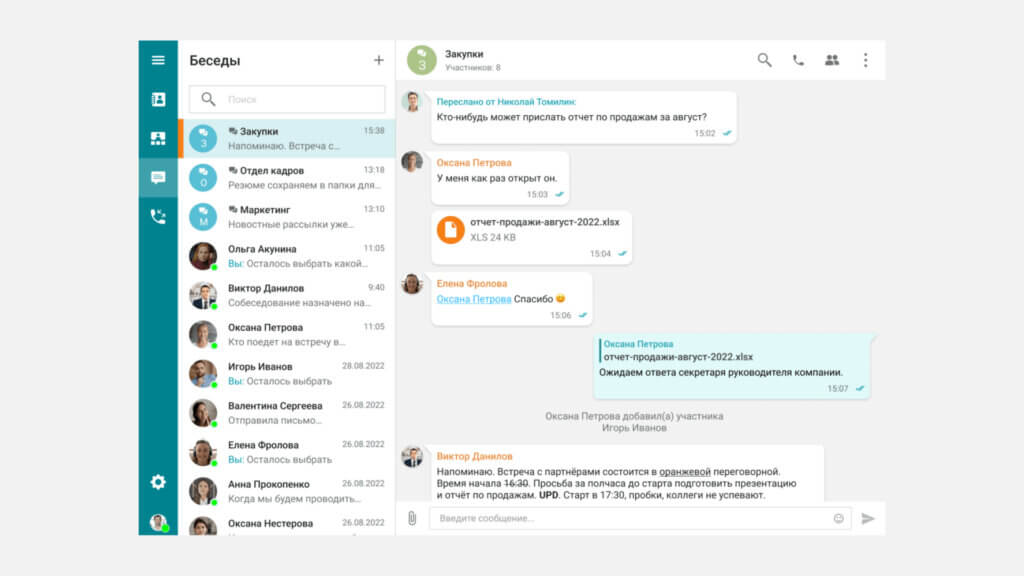


Рисунок 1.1.7. Окно с чатами в TrueConf

В качестве главного преимущества данного сервиса можно назвать встроенный ИИ в видеоконференции, который используется для адаптивного улучшения видео и звука. Сервер TrueConf Server научили распознавать русскую речь всех участников конференции, а также поиск по произнесенным фразам с проигрыванием нужных фрагментов в записи встречи.

Главным недостатком данной системы можно считать отсутствие собственного сервера и необходимость установки системы на сервер компании.

* 1. **SWOT-анализ**

В ходе выполнения курсового проекта для определения функциональных требований был проведен SWOT-анализ (таблица 1.2.1):

Таблица 1.2.1

|  |  |  |
| --- | --- | --- |
|  | **Возможности**  1. Растущие рекламные возможности  2. Огромная база клиентов позволяет открыть новые источники дохода  3. Цифровая трансформация | **Угрозы**  1. Меняющиеся тенденции в экономике  2. Низкие барьеры для входа конкурентов |
| **Сильные стороны**  1. Многопользовательский режим  2. Современные технологии приложения  3. Бесплатное пользование  4. Доступный на разных платформах  5. Групповые чаты  6. Хранилище для файлов | За счет использования многопользовательского режима программа эффективно справляется с потоком новых пользователей, успешно проходя цифровую трансформацию. Также она поддерживает хранилище различный объемов информации, введение групповых чатов, при этом все происходит бесплатно. | Из-за нестабильной ситуации в мире не всегда известно актуальность продукта, поэтому нужно быть вовлеченным во процессы, чтобы оставаться конкурентоспособным на фоне других участников, выходящих на рынок. |
| **Слабые стороны**  1. Проблема с защитой данных пользователей  2. Не работает без интернета  3. Нет групповых звонков | Важно сократить ситуации, при которых может быть затронута безопасность пользователя. Также необходимо дать все возможное, чтобы пользователь хотел пользоваться приложением, независимо от того, что оно работает только с интернетом. Проработать функционал групповых чатов, которые будут заменять групповые звонки. | Необходимо следить за конкурентами, а именно за тем, как они справляются с аналогичными проблемами, чтобы не пропасть с поля зрения пользователей. Также важно придумать особенности, благодаря которым приложение будет выигрывать на рынке. |

* 1. **PEST-анализ**

В ходе выполнения курсового проекта для определения функциональных требований был проведен PEST-анализ (таблица 2):

Таблица 1.3.1

|  |  |  |
| --- | --- | --- |
| Фактор | Возможность | Угроза |
| Политические факторы | 1. Внешнеполитические санкции | 1. Иностранные сервисы запретили российским пользователям пользоваться своими приложениями, появилось необходимость искать аналоги |
| 2. Изменения в законодательстве, появление запретов на распространение той или иной информации | 2. Появилась необходимость отслеживать за каждым словом, используемым в ПО как разработчиками, так и пользователями, чтобы не попасть под нарушение законодательства РФ |
| Экономические факторы | 1. Рост инфляции | 1. Повышение возможностей ПО может быть экономически не выгодна |
| 2. Нестабильный курс валют | 2. От курса валют может зависеть тенденции в мире технологий, важно следить за этим. |
| Социокультурные факторы | 1. Нестабильные тренды в обществе | 1. Важно следить за изменениями, чтобы быть конкурентоспособным |
| 2. Повышение миграционных показателей | 2. Может произойти резкий отток или приток клиентов, поэтому важно следить за этим, чтобы не допустить поломку ПО. |
| Технологические факторы | 1. Активное использование ИИ | 1. Необходимо пользоваться новыми технологиями, чтобы приобретать те же возможности для улучшения возможностей ПО |
| 2. Повышение спроса технических специалистов | 2. Необходимо создать минимальные угрозы для поломки ПО, чтобы избежать того случая, что починить будет некому |
|  | 3. Рост числа гаджетов | 3. Необходимо делать ПО, которое можно использовать на разных типах ОС. |

* 1. **Формирование требований к программному продукту**
     1. **Бизнес требования**

Бизнес требования к корпоративному мессенджеру:

1. Безопасность данных: мессенджер должен обеспечивать высокий уровень защиты конфиденциальных и чувствительных информационных данных компании.

2. Интеграция с другими инструментами: мессенджер должен позволять интеграцию с другими корпоративными инструментами, такими как электронная почта, календари, проектные системы и CRM-системы. Это поможет улучшить производительность, сократить время на обмен информацией и избежать дублирования данных.

3. Функциональность: мессенджер должен предоставлять широкий набор функций, таких как обмен сообщениями, голосовые и видео вызовы, возможность создания групповых чатов и каналов, возможность обмена файлами и документами, функция поиска и архивирования сообщений.

4. Географическая доступность: мессенджер должен быть доступным для использования в любой точке мира и на любых устройствах - компьютерах, смартфонах и планшетах.

5. Удобный пользовательский интерфейс: мессенджер должен иметь простой и интуитивно понятный интерфейс, приятный дизайн и настраиваемые настройки, чтобы каждый сотрудник мог настроить его под свои потребности.

6. Функциональная мобильная версия: мессенджер должен иметь полноценную мобильную версию, чтобы сотрудники могли общаться и работать вне офиса.

7. Аналитика и отчетность: мессенджер должен иметь функционал аналитики и отчетности, чтобы предоставлять данные о количестве сообщений, активности пользователей, эффективности коммуникаций и т.д.

8. Разграничение доступа: мессенджер должен позволять установку прав доступа на уровне отдельных пользователей или групп, чтобы обеспечить безопасность и контроль доступа к информации.

9. Масштабируемость: мессенджер должен быть готов к масштабированию и способен адаптироваться к растущим потребностям компании, поддерживая большое количество пользователей и высокую производительность.

**1.4.2. Пользовательские требования**

Пользовательские требования к корпоративному мессенджеру:

1. Простота использования: мессенджер должен иметь интуитивно понятный интерфейс и простые настройки, чтобы пользователи могли легко освоить и начать пользоваться системой без необходимости обучения.

2. Обмен сообщениями: мессенджер должен позволять отправлять текстовые сообщения с возможностью форматирования и вложениями, чтобы пользователи могли коммуницировать и передавать информацию в удобной форме.

3. Голосовые и видео вызовы: мессенджер должен обеспечивать возможность проводить голосовые и видео вызовы для более продуктивного взаимодействия и обсуждения вопросов с коллегами.

4. Групповые чаты и каналы: мессенджер должен поддерживать возможность создания групповых чатов и каналов для организации работы в команде, обмена информацией и совместной работы над проектами.

5. Обмен файлами и документами: мессенджер должен позволять пользователям обмениваться файлами и документами различных форматов, чтобы упростить совместное редактирование и обмен материалами.

6. Уведомления: мессенджер должен предоставлять возможность получать уведомления о новых сообщениях и других активностях, чтобы пользователи могли быть в курсе событий и отреагировать своевременно.

8. Архивирование и поиск сообщений: мессенджер должен хранить и архивировать сообщения, чтобы пользователи могли легко находить нужную информацию в случае необходимости.

**1.4.3. Функциональные требования**

Функциональные требования к корпоративному мессенджеру:

1. Регистрация и аутентификация: мессенджер должен предоставить возможность регистрации новых пользователей и аутентификации существующих пользователей с использованием безопасных методов (например, паролей, двухфакторной аутентификации и др.).
2. Обмен сообщениями: мессенджер должен позволять пользователям обмениваться текстовыми сообщениями в режиме реального времени.
3. Голосовые и видео вызовы: мессенджер должен обеспечивать возможность проводить голосовые и видео вызовы между пользователями с использованием оптимального вызова (прямого или через сервер).
4. Групповые чаты и каналы: мессенджер должен поддерживать создание и управление групповыми чатами и каналами, где пользователи могут общаться и обмениваться информацией с несколькими участниками сразу.
5. Обмен файлами и документами: мессенджер должен позволять пользователям отправлять и получать файлы и документы различных форматов, поддерживать возможность просмотра и скачивания файлов.
6. Уведомления: мессенджер должен предоставлять уведомления о новых сообщениях, голосовых и видео вызовах, изменениях в групповых чатах и других активностях.
7. Архивирование и поиск сообщений: мессенджер должен сохранять и архивировать сообщения, обеспечивать возможность поиска по ключевым словам, датам и отправителям для облегчения поиска и доступа к информации.
8. Управление доступом: мессенджер должен позволять устанавливать права доступа к чатам и файлам для различных пользователей, администраторов и групп, обеспечивая безопасность и конфиденциальность информации.
9. Шифрование и безопасность: мессенджер должен обеспечивать шифрование передаваемых данных и сохраненных сообщений, а также обеспечивать защиту от несанкционированного доступа к чатам и файлам.

**1.4.4. Нефункциональные требования**

Нефункциональные требования к корпоративному мессенджеру:

1. Безопасность: мессенджер должен обеспечивать высокий уровень защиты данных, включая защиту от несанкционированного доступа, шифрование информации, защиту от вредоносного программного обеспечения и анонимности пользователей.
2. Производительность: мессенджер должен обеспечивать высокую производительность при отправке и получении сообщений, голосовых и видео вызовов, а также при выполнении других операций для обеспечения плавной и эффективной коммуникации.
3. Интеграция: мессенджер должен обеспечивать легкую интеграцию с другими существующими системами и инструментами, такими как почта, календари, CRM и др., для удобства и эффективности коммуникации.
4. Пользовательский опыт: мессенджер должен предоставлять удобный и интуитивно понятный интерфейс, приятный дизайн и гибкие настройки, чтобы пользователи могли легко освоить систему и настроить ее под свои потребности.
5. Поддержка: мессенджер должен предоставлять эффективную техническую поддержку, и регулярные обновления для устранения возможных проблем и обеспечения бесперебойной работы системы.
6. Совместимость: мессенджер должен быть совместим с широким спектром устройств и операционных систем, чтобы пользователи могли использовать его на своих предпочтительных платформах и устройствах.
7. Локализация: мессенджер должен поддерживать возможность локализации для адаптации интерфейса и коммуникации на разных языках, соответствуя потребностям пользователей из разных стран и регионов.

**1.4.5. Ограничения**

Основные ограничения, которые могут повлиять на проектирование и разработку корпоративного мессенджера:

1. Бюджет: наличие ограниченного бюджета может влиять на выбор технологий, функциональности и возможностей, которые можно реализовать в мессенджере.
2. Ресурсы: доступные ресурсы, включая команду разработчиков, аппаратное и программное обеспечение, могут оказать влияние на сложность и сроки разработки мессенджера.
3. Совместимость и интеграция: наличие существующих систем и инструментов в организации, с которыми необходимо интегрировать мессенджер, может ограничивать выбор технологий и возможности интеграции.
4. Технические ограничения: имеющиеся технические ограничения, такие как доступные технологии, совместимость устройств и операционных систем, могут оказывать влияние на выбор и реализацию функциональности мессенджера.

**1.4.6. Требования к интерфейсам**

Требования к интерфейсу у корпоративного мессенджера:

1. Простота использования: интерфейс должен быть интуитивно понятным и легким в освоении для новых пользователей, а также предоставлять удобные и понятные элементы управления для выполнения основных операций.
2. Четкая структура и навигация: интерфейс должен иметь ясную структуру, упрощающую поиск и доступ к основным функциям, а также эффективные механизмы навигации для перемещения между различными разделами и чатами.
3. Понятные и информативные элементы: элементы интерфейса, такие как кнопки, иконки, метки полей и т.д., должны быть понятными и информативными для пользователей, предоставлять достаточную информацию и подсказки для правильного использования функций.
4. Адаптивность и отзывчивость: интерфейс должен быть адаптивным к различным размерам экранов и высоким разрешениям, а также обеспечивать отзывчивость в реальном времени для быстрого и плавного взаимодействия.
5. Удобство работы с чатами: интерфейс должен обеспечивать удобство работы с чатами, включая возможность быстрого переключения между чатами, отображения прочитанных и непрочитанных сообщений, а также поддержки ответов и цитирования сообщений.
6. Визуальная привлекательность: интерфейс должен быть эстетически привлекательным и приятным для глаза, с использованием соответствующих цветов, шрифтов и графических элементов.

**1.4.7. Требования к данным**

Требования к данным у корпоративного мессенджера:

1. Конфиденциальность: данные, передаваемые и хранящиеся в мессенджере, должны быть конфиденциальными и защищенными от несанкционированного доступа.
2. Шифрование: мессенджер должен обеспечивать шифрование данных во время их передачи и хранения, чтобы предотвратить возможность перехвата и чтения информации третьими лицами.
3. Резервное копирование: мессенджер должен иметь механизм резервного копирования данных, чтобы защитить информацию от потери или повреждения и обеспечить возможность восстановления.
4. Управление доступом: мессенджер должен предоставлять возможность управления доступом к данным на уровне пользователей и групп, чтобы ограничить доступ только необходимым пользователям и защитить информацию от несанкционированного распространения.
5. Сохранение данных: мессенджер должен обеспечивать сохранность и целостность данных в течение заданного срока, чтобы пользователи могли хранить и получать доступ к истории сообщений.
6. Архивирование и поиск данных: мессенджер должен предоставлять возможность архивирования и поиска сообщений и файлов, чтобы пользователи могли быстро находить нужные данные по ключевым словам, датам и участникам.
   1. **Программные средства разработки**

Корпоративный мессенджер «PigeonGram» был разработан в среде разработки Visual Studio с на языке программирования C#. Для гибкой работы с графическими стилями xml-код был написан вручную.

Для реализации сервера был использован язык программирования C#.

Для хранения данных о пользователях, сообщений и файлов был выбран SQL Server.

* 1. **Аппаратные средства разработки**

К основным аппаратным средствам разработки корпоративного мессенджера относятся:

1. Сервер. Является мощным вычислительным узлом с достаточным объемом оперативной памяти, процессором высокой производительности и большим объемом хранилища данных.
2. В ходе реализации были использованы два ноутбука на операционной системе Windows.
   1. Сервер был реализован на ПК с процессором Intel Core i7, оперативной памятью 8 МБ и SSD на 512 ГБ;
   2. Клиент был написан на ПК с процессором Intel Core i5, оперативной памятью 8 МБ и HDD на 256 Гб и SSD на 128 Гб.

Все аппаратные средства взаимодействуют между собой, чтобы обеспечить работу корпоративного мессенджера на высоком уровне производительности и безопасности.

**2. ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ**

**2.1. Архитектура системы**

Архитектура разрабатываемого приложения представлена на рис. 2.1.1.

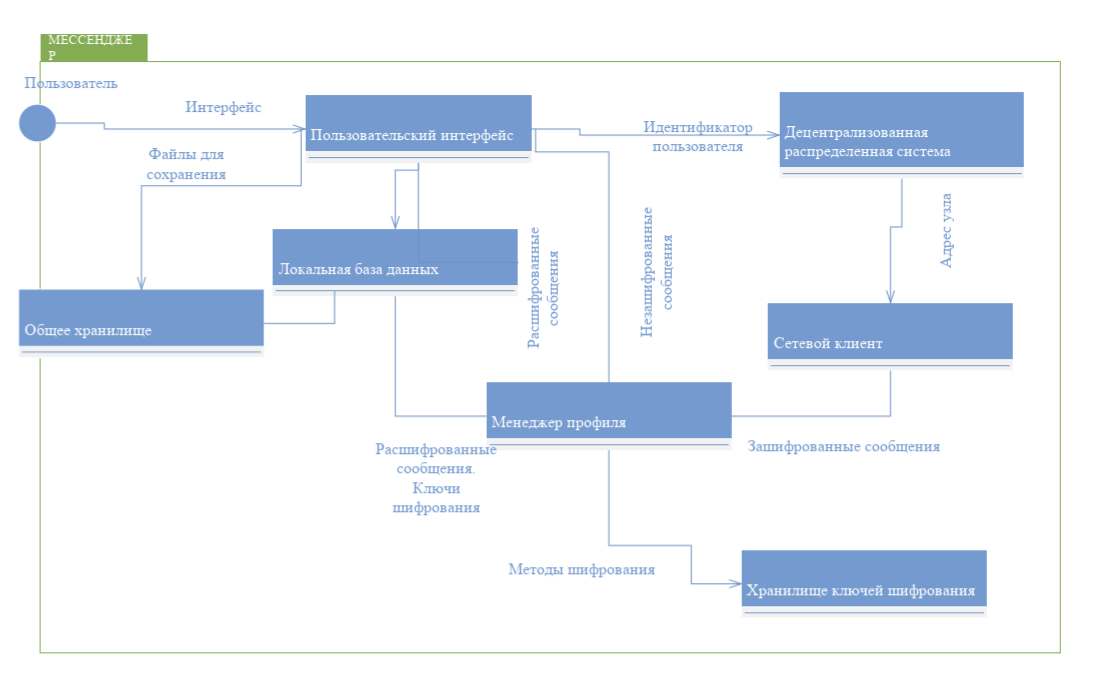


Рисунок 2.1.1. Архитектура корпоративного мессенджера

Проект состоит из следующих компонентов:

* Пользовательский интерфейс;
* Локальная база данных;
* Общее хранилище;
* Менеджер профиля;
* Хранилище ключей шифрования;
* Сетевой клиент;
* Децентрализованная распределительная система.

Пользователь взаимодействует с мессенджером через пользовательский интерфейс, который взаимодействует с общим хранилищем, локальной базой данных, менеджером профиля и децентрализованной распределительной системой.

В общем хранилище пользователь может сохранить файлы, которые будут доступны другим пользователям. Файлы сохраняются в локальной базе данных.

Через децентрализированную распределительную систему программа идентифицирует пользователя, посылая адрес узла сетевому клиенту.

Сетевой клиент посылает зашифрованные сообщения менеджеру профиля, который использует хранилище ключей шифрования для шифровки сообщений.

**2.2. Моделирование основных сценариев системы**

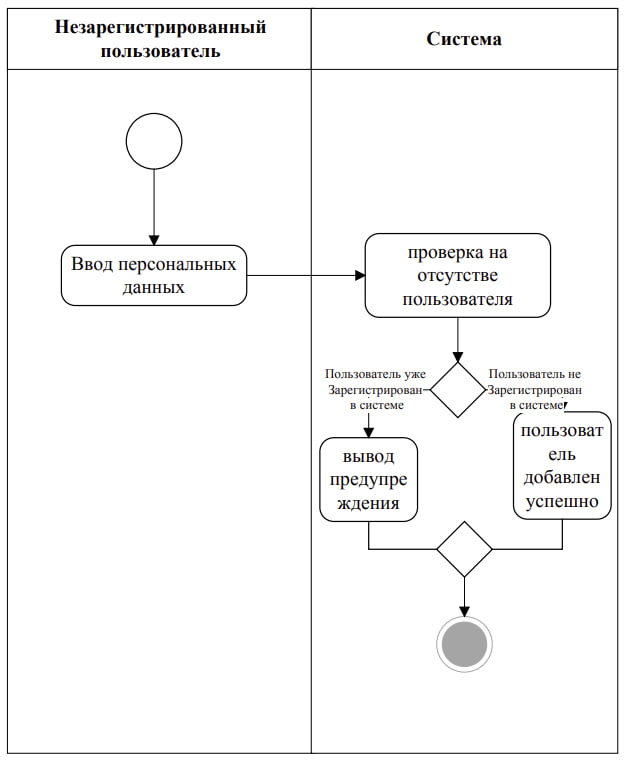


Рисунок 2.2.1. Диаграмма деятельности. Регистрация пользователя

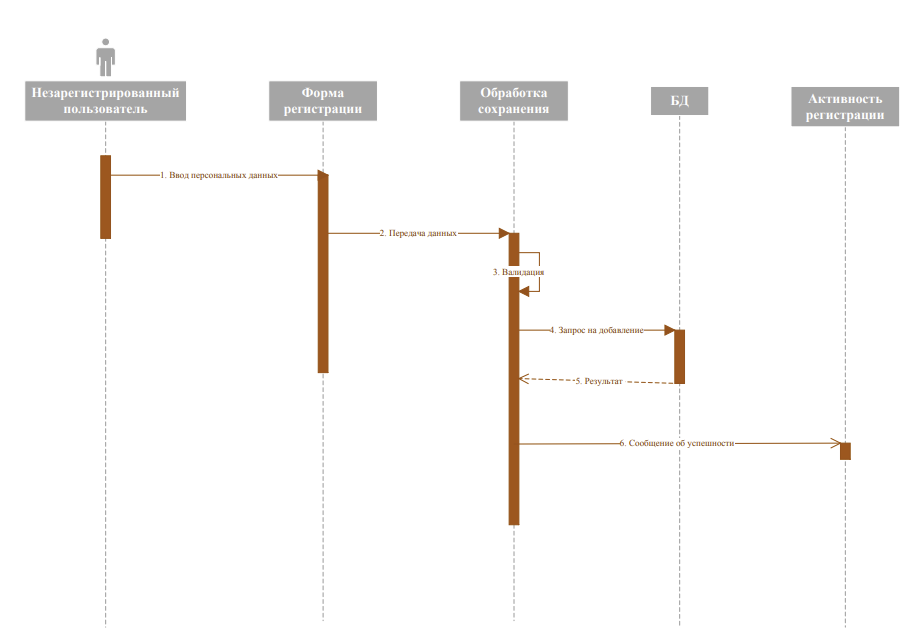


Рисунок 2.2.2. Диаграмма последовательности. Регистрация пользователя

На рис. 2.2.1-2.2.2 представлены диаграммы последовательности и деятельности на прецедент «Регистрация пользователя». Регистрация проходит следующих образом: незарегистрированный пользователь вводит персональные данные в форму регистрации. Данные передаются на сервер, где происходит валидация. После проверки правильности введенных данных делается запрос на добавление пользователя в БД. БД сообщает результат, после выводится сообщение об успешности.

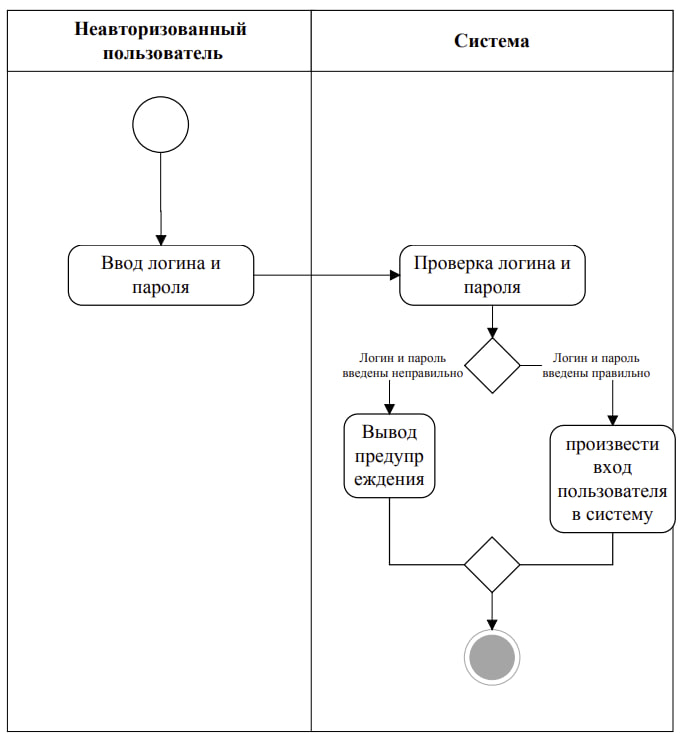


Рисунок 2.2.3. Диаграмма деятельности. Авторизация пользователя

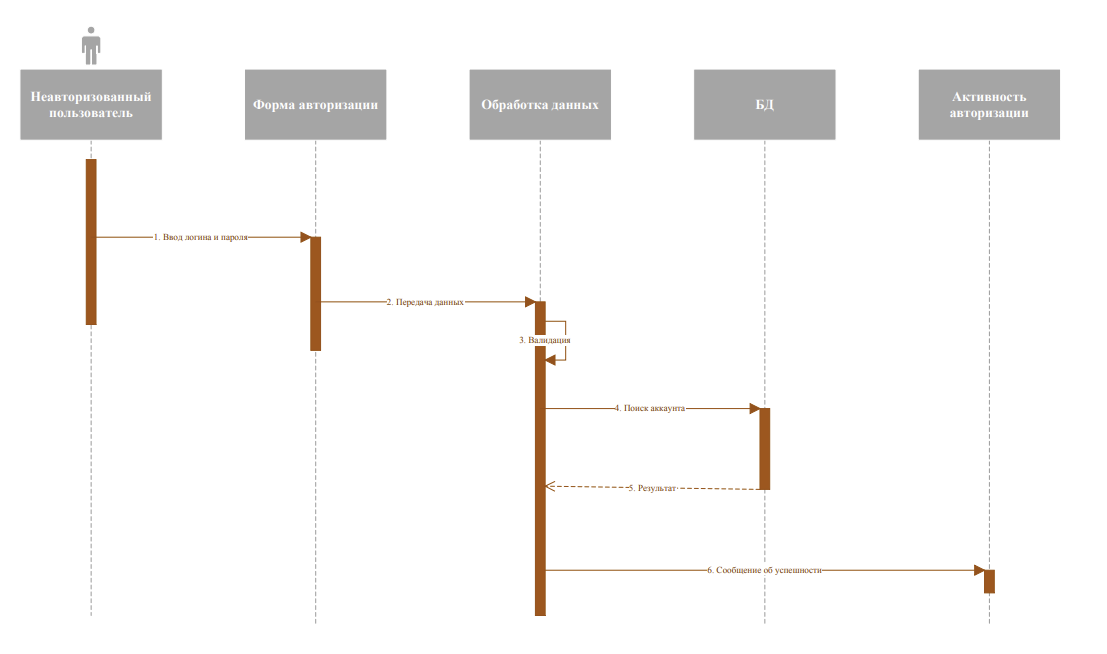


Рисунок 2.2.4. Диаграмма последовательности. Авторизация пользователя

На рис. 2.2.3-2.2.4 представлены диаграммы последовательности и деятельности на прецедент «Авторизация пользователя». Авторизация проходит следующих образом: неавторизованный пользователь вводит логин и пароль в форму авторизации. Данные передаются на сервер, где происходит валидация. После проверки правильности введенных данных происходит поиск аккаунта в БД. БД сообщает результат, после выводится сообщение об успешности.



Рисунок 2.2.5. Диаграмма деятельности. Отправка сообщений



Рисунок 2.2.6. Диаграмма последовательности. Отправка сообщений

На рис. 2.2.5-2.2.6 представлены диаграммы деятельности и последовательности на прецедент «Отправка сообщений». Отправка происходит по следующему алгоритму: авторизованный пользователь пишет сообщение в интерфейсе клиента (под вопросом как назвать). Затем клиент делает запрос на отправку сообщения серверу. После обработки сервер отправляет сообщение другому клиенту.

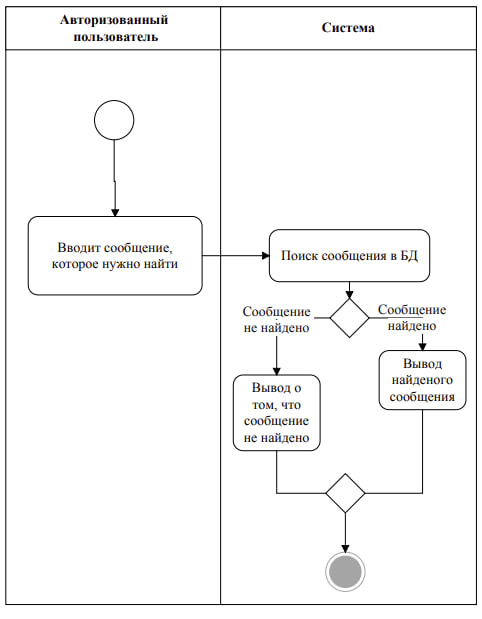


Рисунок 2.2.7. Диаграмма деятельности. Поиск сообщения

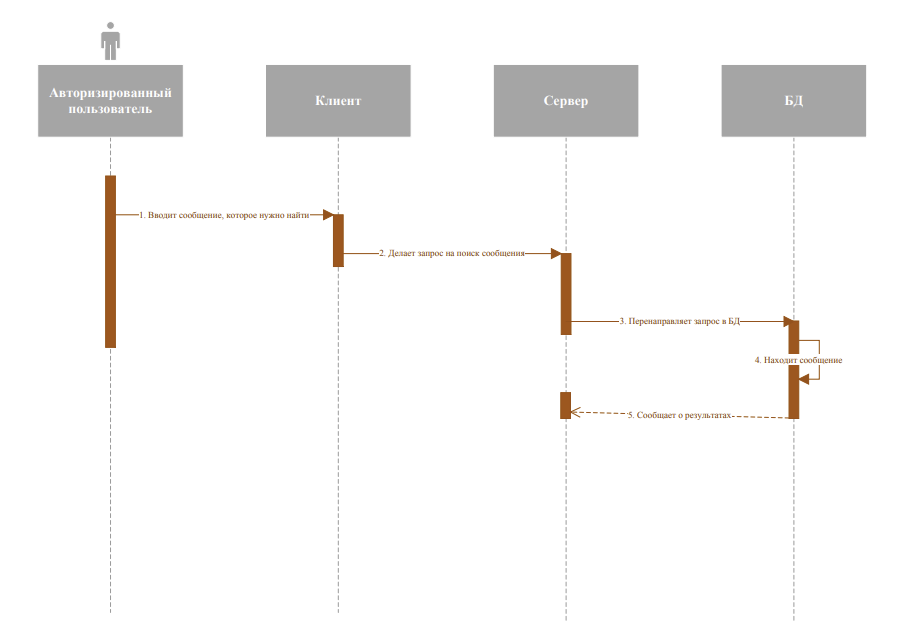


Рисунок 2.2.8. Диаграмма последовательности. Поиск сообщения

На рис. 2.2.7-2.2.8 представлены диаграммы деятельности и последовательности на прецедент «Поиск сообщения». Авторизованный пользователь вводит через интерфейс клиента сообщение, которое ему необходимо найти. Клиент делает серверу запрос на поиск сообщения, сервер перенаправляет запрос к БД. БД ищет сообщение и сообщает о результате серверу.

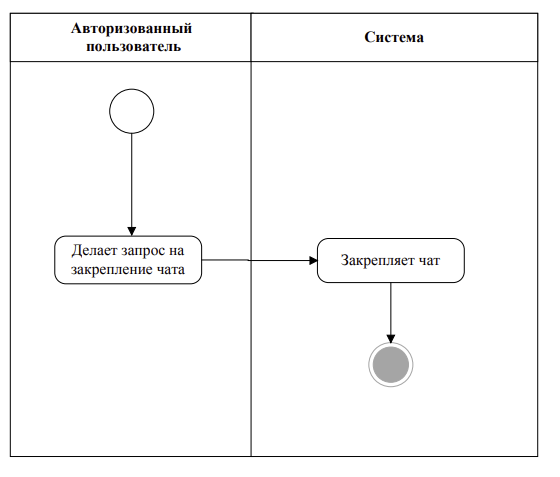


Рисунок 2.2.9. Диаграмма деятельности. Закрепление чата

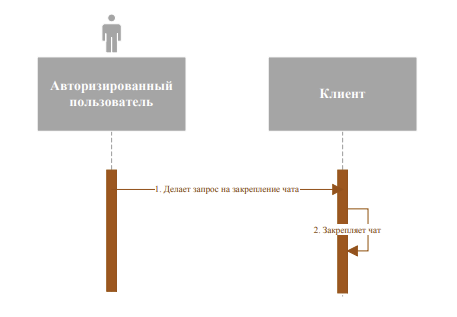


Рисунок 2.2.10. Диаграмма последовательности. Закрепление чата

На рис. 2.2.9-2.2.10 представлены диаграммы деятельности и последовательности на прецедент «закрепление чата». Пользователь через интерфейс клиента делает запрос на закрепление чата, клиент закрепляет чат.

* 1. **Проектирование графического интерфейса пользователя**

На рисунках ниже представлены интерфейс и основные функции корпоративного мессенджера PigeonGram.

На рис. 2.3.1 представлен непосредственно сам интерфейс приложения.

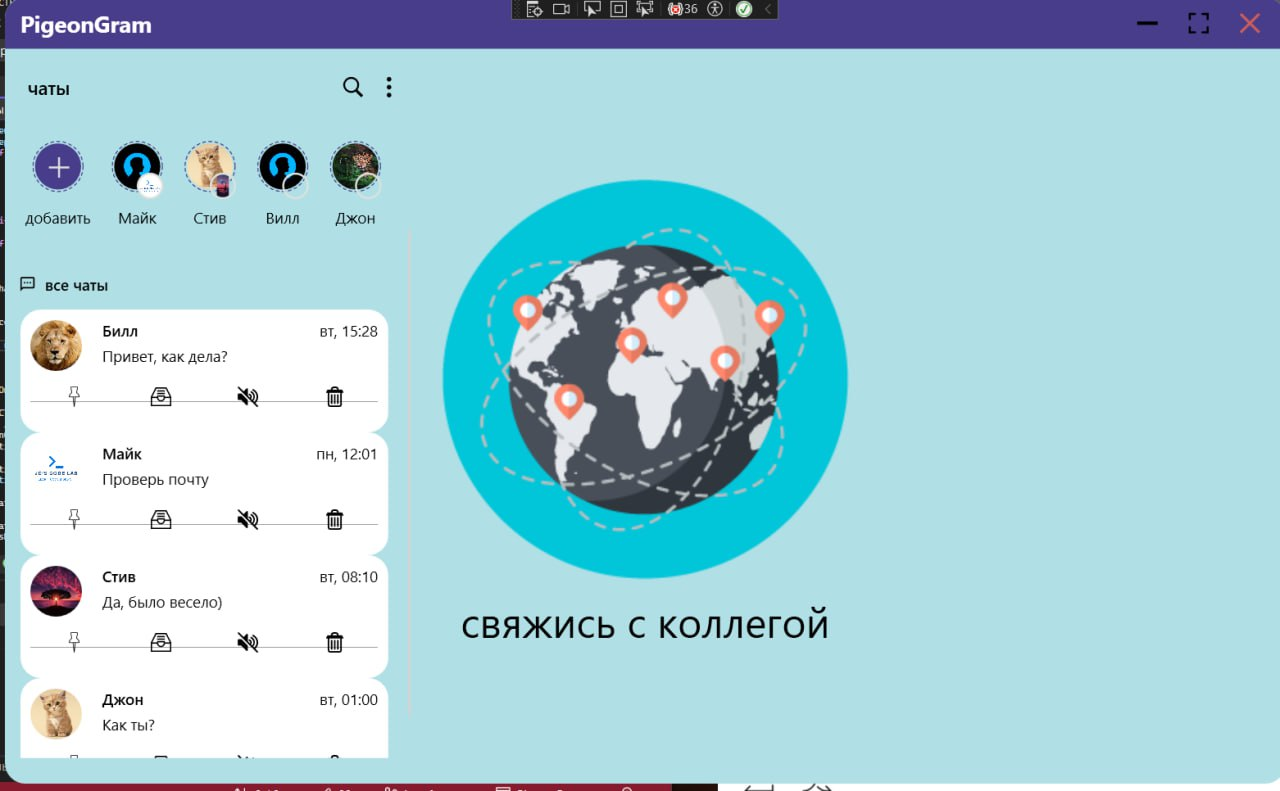


Рисунок 2.3.1. Интерфейс приложения

На рис. 2.3.2 показаны основные функции приложения, а именно: создание новой группы, начать видеозвонок, избранные сообщения и настройки.

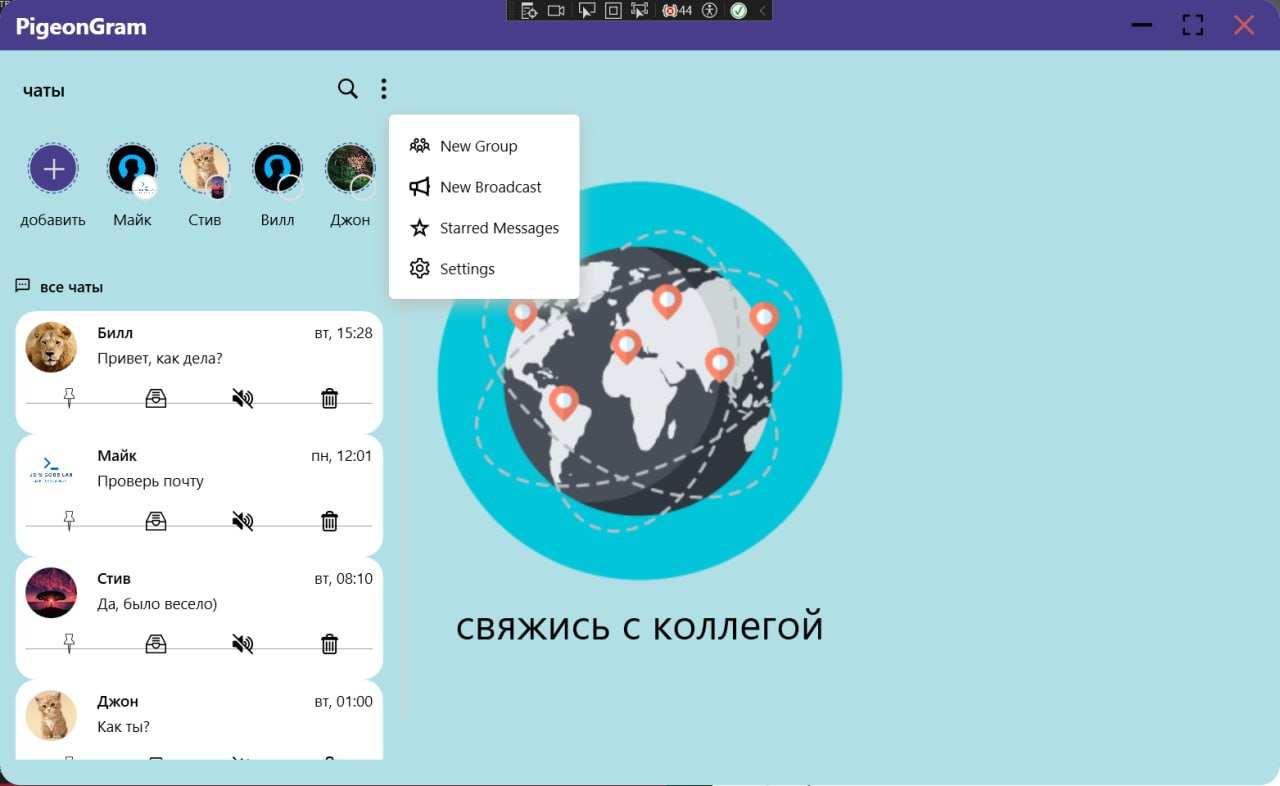


Рисунок 2.3.2. Дополнительные функции

На рис. 2.3.3 показана реализация функции закрепления сообщения.

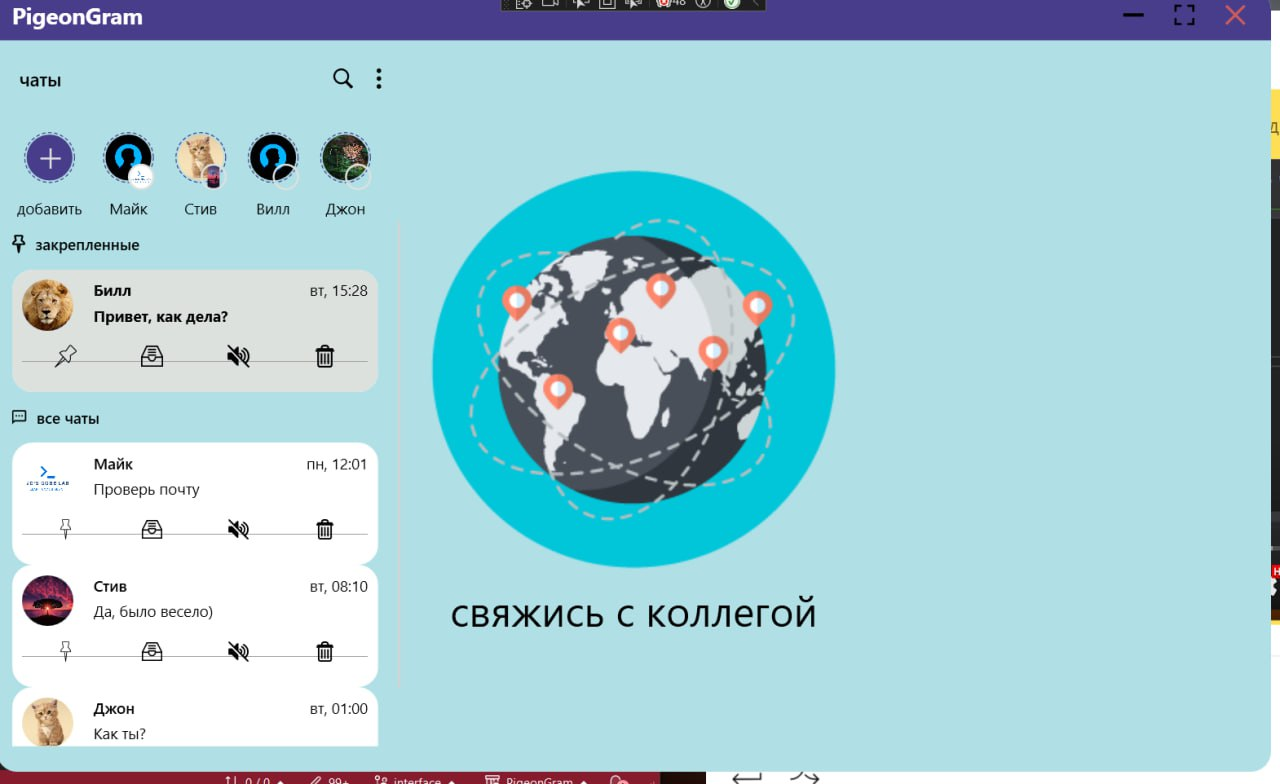


Рисунок 2.3.3. Закрепленные сообщения

На рис. 2.3.4 показана реализация функции архивирования сообщения.

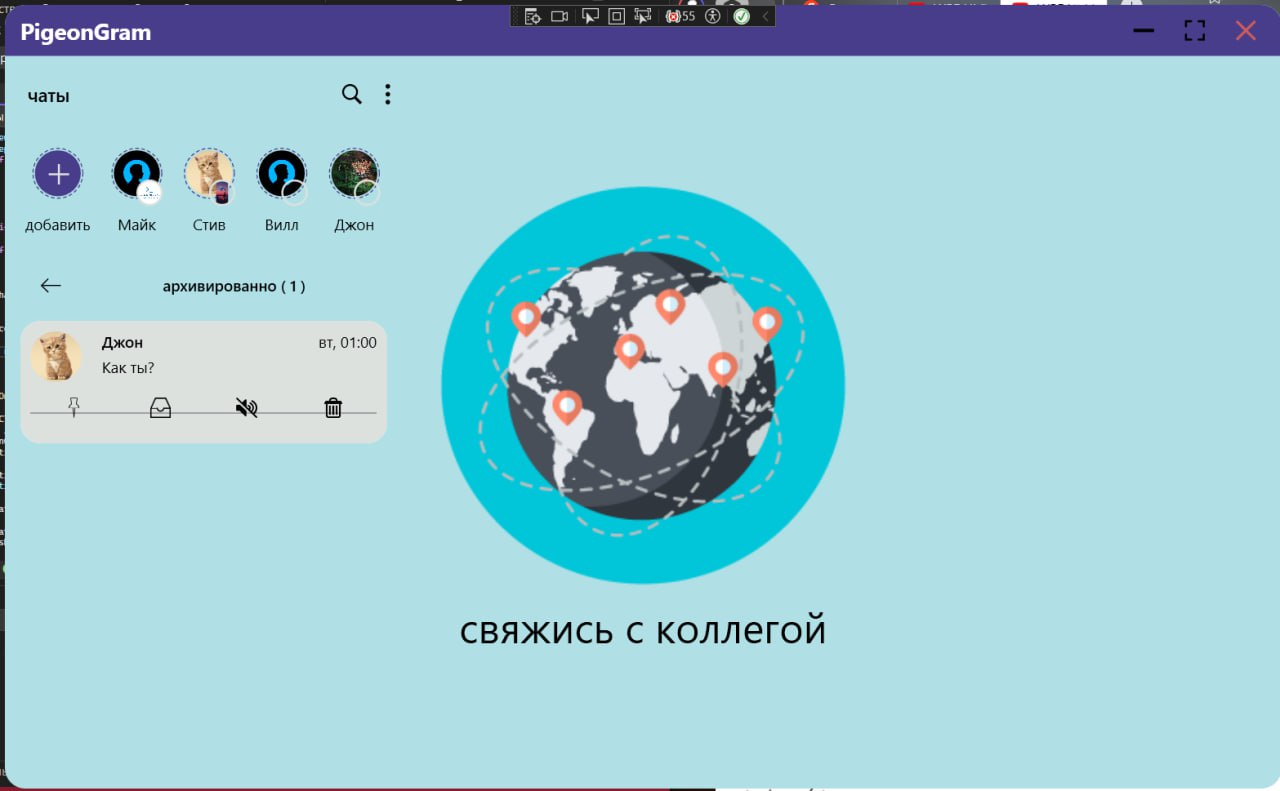


Рисунок 2.3.4. Архивированные сообщения

* 1. **Проектирование и разработка модели данных**

В предметной области можно выделить следующие сущности:

* Сущность «Контакты»;
* Сущность «Чаты».

Разрабатываемая БД предназначена для хранения всей долговременной информации системы, является реляционной и реализована при помощи СУБД MS SQL Server.

Ниже приведено описание назначений и свойств каждого поля таблиц в БД:

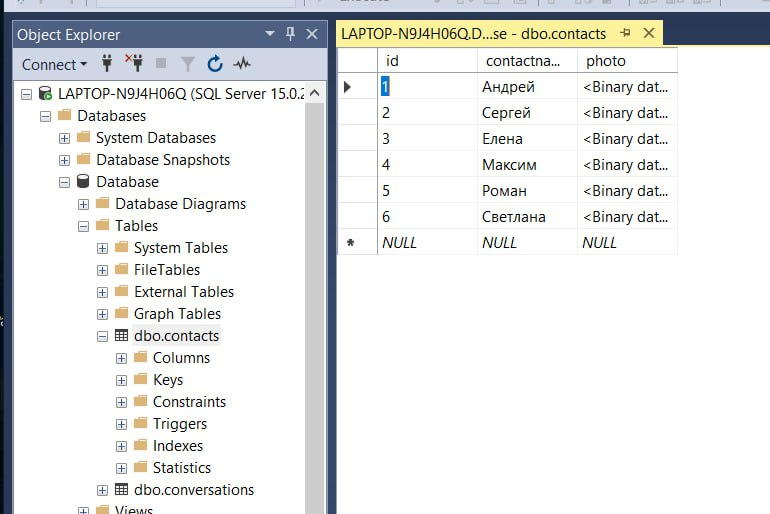


Рисунок 2.4.1. Таблица "Contacts" – Контакты

В данной таблице хранятся данные о контактах – Id контакта (id), имя контакта (ContactName) и фото контакта (Photo). Ни одно поле не может иметь нулевого значения. Id контакта является первичным ключом.

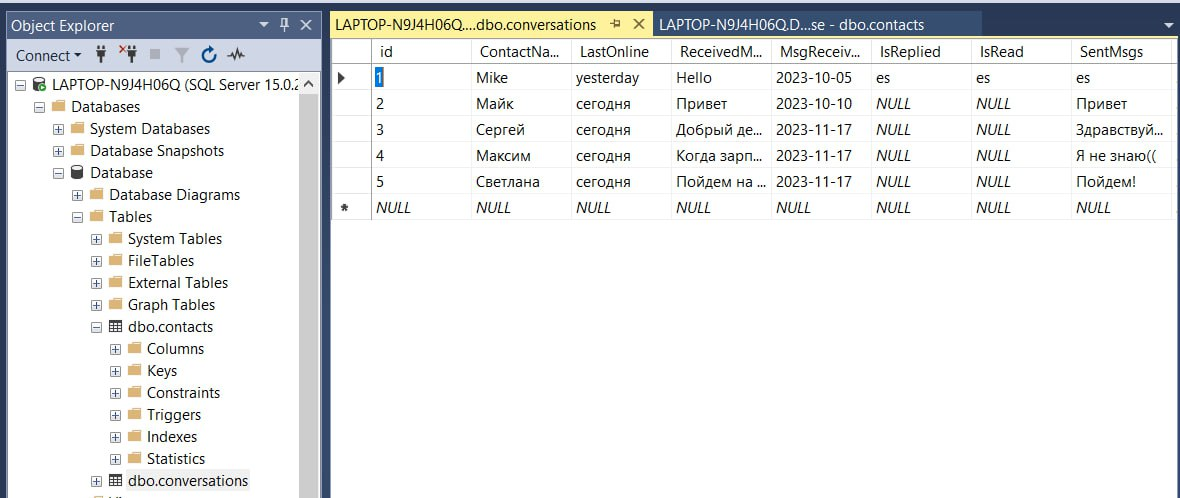


Рисунок 2.4.2. Таблица "Conversation" – Чаты

В данной таблице хранятся данные о чатах – id, имя контакта (ContactName), последнее посещение (LastOnline), текст полученного сообщения (ReceivedMsg), время, когда сообщение было получено (MsgReceivedTime), информация об ответе на сообщение (IsReplied), информация о прочтении сообщения (IsRead). Id является первичным ключом. Все поля, кроме информации об ответе на сообщение и информации о прочтении сообщения не могут иметь нулевого значения.

1. **РАЗРАБОТКА ПРОГРАММНОГО ПРОДУКТА**

Поскольку программный продукт является клиент-серверным приложением, он состоит из двух частей – сервера (который, в свою очередь, состоит из хоста и сервиса) и клиента. Ниже представлены описания и листинги кода реализации функций.

**3.1. Реализация серверной части**

Функция «Connect» добавляет нового пользователя и принимает следующие значения: ID, которое присваивает новое значение id каждому пользователю, Name, которое хранит поступившее имя пользователя, и operationContext, значение для которого поступают вместе с соединением пользователя. После соединения, все пользователи получают уведомление о том, что был добавлен новый юзер. Позже будет реализовано добавление пользователя в БД.

public int Connect(string name)

{

ServerUser user = new ServerUser()

{

ID = nextId,

Name = name,

operationContext = OperationContext.Current

};

nextId++;

SendMsg(": " + user.Name + " was added", 0);

users.Add(user);

return user.ID;

}

Функция «SendMsg» отправляет сообщения пользователю. Функция принимает значения msg и id; перебирает всех пользователей из списка и формирует сообщение из параметра msg и текущего времени. После функция перебирает всех пользователей и проверяет, являются ли они ненулевыми. После отправляет сообщение пользователям.

public void SendMsg(string msg, int id)

{

foreach (var item in users)

{

string answer = DateTime.Now.ToShortTimeString();

var user = users.FirstOrDefault(i => i.ID == id);

if (user != null)

{

answer += ": " + user.Name + " ";

}

answer += msg;

item.operationContext.GetCallbackChannel<IServerChatCallback>().MsgCallback(answer);

}

}

Также в серверной части присутствует хост, задача которого заключается в том, чтобы распределять данные сервиса.

static void Main(string[] args)

{

using (var host = new ServiceHost(typeof(wcf\_chat.ServiceChat)))

{

host.Open();

Console.WriteLine("host is started!");

Console.ReadLine();

}

}

**3.2. Реализация клиентской части**

Функция отправки сообщения.

В представленной функции элементы конвертируются и передаются в момент, когда сообщение отправляется, если текстбокс не пустой. Благодаря методу происходит доваление сообщения в лист диалога. И текстбокс становится пустым, после отправки сообщения.

public void SendMessage()

{

//send message only when the textbox is not empty

if(!string.IsNullOrEmpty(MessageText))

{

var conversation = new ChatConversation()

{

ReceivedMessage = MessageToReplyText,

SentMessage = MessageText,

MsgSentOn = DateTime.Now.ToString("MMM dd, hh:mm tt"),

MessageContainsReply = IsThisAReplyMessage

};

//add msg to conversation list

FilteredConversations.Add(conversation);

Conversations.Add(conversation);

//clear message properties and textbox when message is sent

MessageText=string.Empty;

IsThisAReplyMessage = false;

MessageToReplyText= string.Empty;

UpdateChatAndMoveUp(Chats, conversation);

UpdateChatAndMoveUp(PinnedChats, conversation);

UpdateChatAndMoveUp(FilteredChats, conversation);

UpdateChatAndMoveUp(FilteredPinnedChats, conversation);

UpdateChatAndMoveUp(ArchiveChats, conversation);

//update

OnPropertyChanged("FilteredConversations");

OnPropertyChanged("Conversations");

OnPropertyChanged("MessageText");

OnPropertyChanged("IsThisAReplyMessage");

OnPropertyChanged("MessageToReplyText");

}

}

На рис. 3.2.1 представлен результат работы функции отправки сообщений.

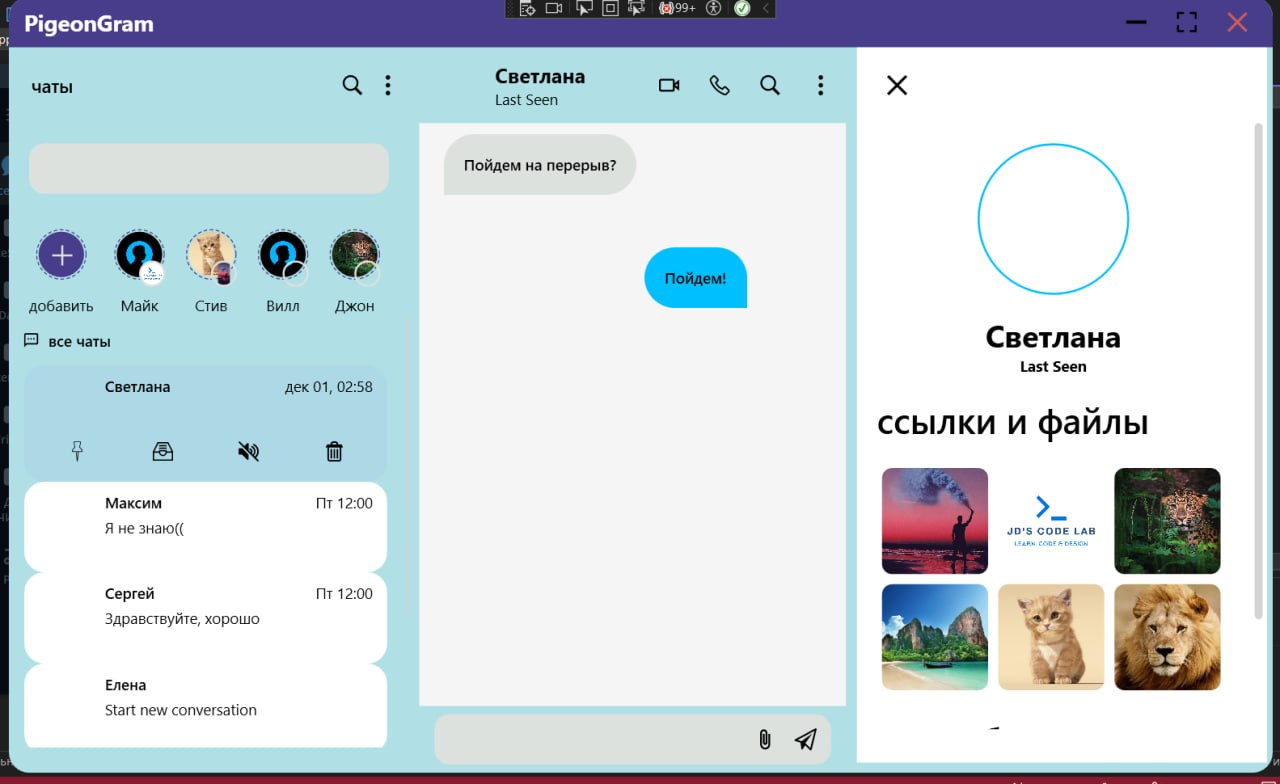


Рисунок 3.2.1. Результат работы функции отправки сообщения

Во время функции закрепления чата считывается значение mPinnedChats, после чего реализуется функционал. От значения будет зависеть закрепится чат или - нет.

public ObservableCollection<ChatListData> PinnedChats

{

get => mPinnedChats;

set

{

if (mPinnedChats == value) return;

mPinnedChats = value;

FilteredPinnedChats = new ObservableCollection<ChatListData>(mPinnedChats);

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredPinnedChats");

}

}

На рис. 3.2.2 представлен результат работы функции закрепления чата.



Рисунок 3.2.2. Результат работы функции закрепления чата

Во время функции архивирования чата проверяется нулевое значение, после чего выполняется команда по архивации чата. При выборе архивации, чат архивируется.

public ICommand ArchiveChatCommand => \_archiveChatCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatListData v)

{

if (!ArchiveChats.Contains(v))

{

//add chat in archive list

ArchiveChats.Add(v);

v.ChatIsArchive = true;

v.ChatIsPinned= false;

//remove chat from pinned and unpinned chat list

Chats.Remove(v);

FilteredChats.Remove(v);

PinnedChats.Remove(v);

FilteredPinnedChats.Remove(v);

//update list

OnPropertyChanged("Chats");

OnPropertyChanged("FilteredChats");

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredPinnedChats");

OnPropertyChanged("ArchiveChats");

}

}

});

На рис. 3.2.3 представлен результат работы функции архивирования чата.



Рисунок 3.2.3. Результат работы функции архивирования чата

**4. ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

* 1. **История изменений**

Таблица 4.1.1

|  |  |  |  |
| --- | --- | --- | --- |
| Дата | Версия | Автор | Описание изменений |
| 15.10.2023 | 1.0 | Малакаускайте Анна | Создание сервера |
| 25.10.2023 | 1.1 | Преснухина Лидия | Добавление главного окна |
| 4.11.2023 | 1.2 | Преснухина Лидия | Добавление функции отправки сообщений, закрепления, архивирования и поиска чата |
| 17.11.2023 | 1.3 | Преснухина Лидия | Привязка БД к проекту, добавление файлов в БД |

* 1. **Терминология тестирования программного обеспечения**

Список терминов и определений, используемых далее в документе и тестировании в целом:

* «Клиент — сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Фактически клиент и сервер — это программное обеспечение.
* Стратегия тестирования — это план проведения работ по тестированию системы или ее модуля, учитывающий специфику функциональности и зависимости с другими компонентами системы и платформы.
* Позитивное тестирование – это тестирование с применением сценариев, которые соответствуют нормальному (штатному, ожидаемому) поведению системы.
* Негативное или отрицательное тестирование — это тип тестирования программного обеспечения (ПО), направленный на проверку того, что система или приложение ведут себя должным образом в негативных ситуациях, то есть, когда они получают недопустимые или неожиданные входные данные.
* Тестирование методом «белого ящика – метод тестирования программного обеспечения, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику.
  1. **Стратегия тестирования**

Стратегия тестирования ПО заключается в использовании метода тестирования «белого ящика» для анализа и оценки работоспособности программы.

* 1. **Определение объектов тестирования**

Определение объектов тестирования

Таблица 4.4.1

|  |  |
| --- | --- |
| Тесты | Объекты тестирования |
| **Виды тестирования** | |
| Функциональное тестирование (functional testing) | * Работа системы регистрации пользователе; * Работа системы авторизации пользователей. |
| Тестирование производительности (performance testing) | * Скорость сохранения данных в БД; * Скорость отображения данных из БД. |
| Нагрузочное тестирование (load testing) | * Нахождение 100 авторизованных пользователей в системе. |
| Тестирование совместимости (Compatibility testing) | * корректное отображение программы в разных версиях ОС Windows (8, 10); * корректное отображение программы на различных соотношениях сторон экрана; * корректное отображение программы на различных разрешениях экрана; |
| **Различные виды тестов** | |
| Позитивные тесты | * регистрация пользователей, ввод данных в форме регистрации корректных значений; * авторизация пользователей, ввод данных в форме авторизации корректных значений. |
| Негативные тесты | * регистрация пользователей, ввод данных в форме регистрации некорректных значений; * авторизация пользователей, ввод данных в форме авторизации некорректных значений. |
| Исследовательские тесты |  |
| **Уровни тестирования** | |
| Модульное тестирование | * тестирование функций пользователя: регистрация, авторизация, отправка сообщений |
| Интеграционное тестирование | * переход с стартового модуля на модуль авторизации; * переход с модуля авторизации на модуль регистрации. |
| Системное тестирование | * попытка обойти средства защиты программы; * скорость отклика программы; * способность системы эффективно использовать оперативную память. |

* 1. **Архитектура тестируемой системы**

Правило функционирования архитектуры клиент-серверного приложения заключается в обработке запросов, происходящих на другом компьютере.

Достоинством реализации данной модели является то, что программный код клиентской и серверной части отделен. Также к достоинству можно отнести то, что к компьютерам, на которых установлен клиент, низкие требования, поскольку все основные вычисления происходят на сервере.

К недостаткам можно отнести высокую стоимость серверного оборудования, поскольку на сервере происходят основные вычисления.

Наконец выделим то, что архитектура позволяет равномерно разбить нагрузку и поделить функционал между клиентской и серверной частью системы.

Схема промышленного стенда будет реализована в офисной обстановке.

Схема тестового стенда будет реализована на личном компьютере.

Конфигурация промышленного стенда

Таблица 4.5.1

|  |  |
| --- | --- |
| **Промышленное оборудование** | |
| **Ресурсы** | **Детали** |
| **Сервер 1** | |
| CPU | Intel core i7 |
| RAM | 8 ГБ |
| HDD | 512 ГБ |
| ОС | Windows 10 |
| Дополнительно | Разрешение 1920x1080 |
| **Клиент 1** | |
| CPU | Intel core i7 |
| RAM | 8 ГБ |
| HDD | 512 ГБ |
| ОС | Windows 10 |
| Дополнительно | Разрешение 1920x1080 |

Конфигурация тестового стенда

Таблица 4.5.2

|  |  |
| --- | --- |
| **Промышленное оборудование** | |
| **Ресурсы** | **Детали** |
| **Сервер 1** | |
| CPU | Intel core i7 |
| RAM | 8 ГБ |
| HDD | 512 ГБ |
| ОС | Windows 10 |
| Дополнительно | Разрешение 1920x1080 |
| **Клиент 1** | |
| CPU | Intel core i7 |
| RAM | 8 ГБ |
| HDD | 512 ГБ |
| ОС | Windows 10 |
| Дополнительно | Разрешение 1920x1080 |

Требования к тестируемому окружению:

* Оборудование среднего сегмента рынка;
* Версия ОС Windows 10;
* Используемая БД MS SQL Server.
  1. **Описание процесса тестирования**

Тестовые сценарии с указанием конкретных тестовых наборов

Таблица 4.6.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название тестового сценария** | **Тестовый набор** | **Ожидаемый результат** | **Актуальный результат** | **Результат тестирования** |
| Проверка отправки сообщения | Графический интерфейс ПО | Сообщение отправлено | Функция ПО работает корректно | Тестирование проведено успешно |
| Проверка функции закрепления чата | Графический интерфейс ПО | Чат закреплен | Функция ПО работает корректно | Тестирование проведено успешно |
| Проверка функции архивирования чата | Графический интерфейс ПО | Чат архивирован | Функция ПО работает корректно | Тестирование проведено успешно |

**ЗАКЛЮЧЕНИЕ**

В ходе исследовательской работы был разработан корпоративный мессенджер.

В первой главе были сформулированы бизнес требования, пользовательские требования, функциональные и нефункциональные требования, требования к интерфейсу и требования к данным.

Во второй главе была рассмотрена архитектура программного продукта, были смоделированы диаграммы деятельности и последовательности.

В третьей главе были рассмотрены и описаны функции продукта.

В четвертой главе было проведено тестирование системы.

В дальнейшем программный продукт будет доработан в рамках дипломного проекта; будут добавлены общая папка, шифрование данных, а также будут соединены серверная и клиентская части.

В процессе написания работы научились работать в команде, распределять обязанности и разрабатывать программный продукт. Полученные нами знания, умения, навыки позволят нам избежать ошибок при дальнейшей профессиональной карьере

**СПИСОК ИСТОЧНИКОВ ИНФОРМАЦИИ**

1. С. С. Качаева. Внутренние коммуникации в период пандемии. 2021. – 7 с.
2. Л.Г. Гагарина. Разработка и эксплуатация автоматизированных информационных систем: Учебное пособие. 2021. -384 с.;
3. В.В. Лисяк. Разработка информационных систем: Учебное пособие. 2019. – 96 с.;
4. SignalR Chat: Сервер и его клиенты. // youtube.com: Sergei Calabonga. Канал для программистов и разработчиков. 2021. URL: <https://youtu.be/TIG2oMgkbcg?si=qzmFIQYKHUB4vTTp> (дата обращения: 24.11.2023)
5. Е. Кучерявый. Пишем мессенджер на С#. Часть 1. Верстка. // skillbox.ru. Онлайн платформа обучения. 2020. URL: <https://skillbox.ru/media/code/pishem_messendzher_na_c_chast_1_vyerstka/> (дата обращения: 24.11.2023)
6. Сборник курсов по языку программирования С#. // codeacademy.com. Онлайн школа для программистов. 2023. URL: <https://www.codecademy.com/catalog/language/c-sharp> (дата обращения: 24.11.2023)
7. Как создать свой мессенджер. // mdm.ooo. Портал для мобильной разработки. 2022. <https://mdm.ooo/ru/how-to-create-a-messaging-app/amp/> (дата обращения: 24.11.2023)
8. Разработка интерфейса мессенджера на С#. youtube.com: JD’s Code Lab. Обучение UI Design. 2020. URL: [Chat App UI Design in Wpf C# - YouTube](https://www.youtube.com/playlist?list=PLe_Naf3_gfQl8ZDxkIQpB3-aZJ7aEPFUC)
9. Создание клиент-серверного приложения C# // youtube.com: Simple Code. Подробные уроки программирования. 2018. URL: <https://www.youtube.com/watch?v=QohqDyTjclw&t=1645s&ab_channel=%23SimpleCode> (дата обращения: 24.11.2023)
10. Miao Yu, Huan Yang, The impact of effective communication on corporate performance, 2023. – 3 с.

**ПРИЛОЖЕНИЕ**

**ЛИСТИНГ ПРОГРАММНЫХ МОДУЛЕЙ**

1. ViewModel.cs

using app\_interface.Commands;

using app\_interface.CustomControls;

using app\_interface.Models;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.ComponentModel;

using System.Data.SqlClient;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Input;

using System.Windows.Media;

namespace app\_interface.ViewModels

{

public class ViewModel : INotifyPropertyChanged

{

//initializing resource dictionary file

// private readonly ICommand \_command;

private readonly ResourceDictionary dictionary =

Application.LoadComponent(new Uri("app\_interface;component/Assets/icons.xaml",

UriKind.RelativeOrAbsolute)) as ResourceDictionary;

#region MainWindow

#region Properties

public string ContactName { get; set; }

public Uri ContactPhoto { get; set; }

public string LastSeen { get; set; }

#region Search Chats

protected bool \_isSearchBoxOpen { get; set; }

public bool IsSearchBoxOpen

{

get => \_isSearchBoxOpen;

set

{

if (\_isSearchBoxOpen == value) return;

\_isSearchBoxOpen = value;

if (\_isSearchBoxOpen != false) SearchText = string.Empty;

OnPropertyChanged("IsSearchBoxOpen");

OnPropertyChanged("SearchText");

}

}

protected string LastSearchText { get; set; }

protected string mSearchText { get; set;}

public string SearchText

{

get => mSearchText;

set

{

//checked if value is different

if (mSearchText == value) { return; }

//update view

mSearchText = value;

//if search text is empty restore message

if (string.IsNullOrEmpty(SearchText))

Search();

}

}

//this is our list containing the window options...

private ObservableCollection<MoreOptionsMenu> \_windowMoreOptionsMenuList;

public ObservableCollection<MoreOptionsMenu> WindowMoreOptionsMenuList

{

get {

return \_windowMoreOptionsMenuList;

}

set

{

\_windowMoreOptionsMenuList = value;

}

}

private ObservableCollection<MoreOptionsMenu> \_attachmentOptionsMenuList;

public ObservableCollection<MoreOptionsMenu>AttachmentOptionsMenuList

{

get

{

return \_attachmentOptionsMenuList;

}

set

{

\_attachmentOptionsMenuList = value;

}

}

#endregion

#endregion

#region Logics

#region Window: More Options Popup

void WindowMoreOptionsMenu()

{

WindowMoreOptionsMenuList = new ObservableCollection<MoreOptionsMenu>()

{

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["newgroup"],

MenuText = "New Group"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["newbroadcast"],

MenuText = "New Broadcast"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["starredmessages"],

MenuText = "Starred Messages"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["settings"],

MenuText = "Settings"

},

};

OnPropertyChanged("WindowMoreOptionsMenuList");

}

void ConversationScreenMoreOptionsMenu()

{

//to populate menu items for conversation screen options list..

WindowMoreOptionsMenuList = new ObservableCollection<MoreOptionsMenu>()

{

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["all media"],

MenuText = "All Media"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["wallpaper"],

MenuText = "Change Wallpaper"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["report"],

MenuText = "Report"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["block"],

MenuText = "Block"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["clearchat"],

MenuText = "Clear Chat"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["exportchat"],

MenuText = "Export Chat"

},

};

OnPropertyChanged("WindowMoreOptionsMenuList");

}

void AttachmentOptionsMenu()

{

//to populate menu items for attachment menu options list..

AttachmentOptionsMenuList = new ObservableCollection<MoreOptionsMenu>()

{

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["docs"],

MenuText = "Docs",

BorderStroke="#3F3990",

Fill="#CFCEEC"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["gallery"],

MenuText = "Gallery",

BorderStroke="#3F3990",

Fill="#CFCEEC"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["audio"],

MenuText = "Audio",

BorderStroke="#3F3990",

Fill="#CFCEEC"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["location"],

MenuText = "Location",

BorderStroke="#3F3990",

Fill="#CFCEEC"

},

new MoreOptionsMenu()

{

Icon = (PathGeometry)dictionary["contact"],

MenuText = "Contact",

BorderStroke="#3F3990",

Fill="#CFCEEC"

},

};

OnPropertyChanged("AttachmentOptionsMenuList");

}

#endregion

public void OpenSearchBox()

{

IsSearchBoxOpen = true;

}

public void ClearSearchBox()

{

if (!string.IsNullOrEmpty(SearchText)) SearchText = string.Empty;

else CloseSearchBox();

}

public void CloseSearchBox() => IsSearchBoxOpen = false;

public void Search()

{

//to avoid searching same text again

if((string.IsNullOrEmpty(LastSearchText) && string.IsNullOrEmpty(SearchText)) || string.Equals(LastSearchText,SearchText))

{ return; }

//if searchbox is empty or chat in null pr char cound less then 0

if (string.IsNullOrEmpty(SearchText) || Chats == null || Chats.Count <= 0)

{

FilteredChats = new ObservableCollection<ChatListData>(Chats ?? Enumerable.Empty<ChatListData>());

OnPropertyChanged("FilteredChats");

FilteredPinnedChats = new ObservableCollection<ChatListData>(PinnedChats ?? Enumerable.Empty<ChatListData>());

OnPropertyChanged("FilteredPinnedChats");

//update last search text

LastSearchText= SearchText;

return;

}

//now, to find all chats that contain the text in our search box

FilteredChats = new ObservableCollection<ChatListData>(

Chats.Where(

chat => chat.ContactName.ToLower().Contains(SearchText)

||

chat.Message!= null && chat.Message.ToLower().Contains(SearchText)

));

OnPropertyChanged("FilteredChats");

FilteredPinnedChats = new ObservableCollection<ChatListData>(

PinnedChats.Where(

pinnedchat => pinnedchat.ContactName.ToLower().Contains(SearchText)

||

pinnedchat.Message != null && pinnedchat.Message.ToLower().Contains(SearchText)

));

OnPropertyChanged("FilteredPinnedChats");

//update last search text

LastSearchText = SearchText;

}

#endregion

#region Commands

protected ICommand \_windowsMoreOptionsCommand;

public ICommand WindowsMoreOptionsCommand

{

get

{

if (\_windowsMoreOptionsCommand == null)

\_windowsMoreOptionsCommand = new CommandViewModel(WindowMoreOptionsMenu);

return \_windowsMoreOptionsCommand;

}

set

{

\_windowsMoreOptionsCommand = value;

}

}

protected ICommand \_conversationScreenMoreOptionsCommand;

public ICommand ConversationScreenMoreOptionsCommand

{

get

{

if (\_conversationScreenMoreOptionsCommand == null)

\_conversationScreenMoreOptionsCommand = new CommandViewModel(ConversationScreenMoreOptionsMenu);

return \_conversationScreenMoreOptionsCommand;

}

set

{

\_conversationScreenMoreOptionsCommand = value;

}

}

protected ICommand \_attachmentOptionsCommand;

public ICommand AttachmentOptionsCommand

{

get

{

if (\_attachmentOptionsCommand == null)

\_attachmentOptionsCommand = new CommandViewModel(AttachmentOptionsMenu);

return \_attachmentOptionsCommand;

}

set

{

\_attachmentOptionsCommand = value;

}

}

protected ICommand \_openSearchCommand;

public ICommand OpenSearchCommand

{

get

{

if (\_openSearchCommand == null)

\_openSearchCommand = new CommandViewModel(OpenSearchBox);

return \_openSearchCommand;

}

set

{

\_openSearchCommand = value;

}

}

protected ICommand \_closeSearchCommand;

public ICommand CloseSearchCommand

{

get

{

if (\_closeSearchCommand == null)

\_closeSearchCommand = new CommandViewModel(CloseSearchBox);

return \_closeSearchCommand;

}

set

{

\_closeSearchCommand = value;

}

}

protected ICommand \_searchCommand;

public ICommand SearchCommand

{

get

{

if (\_searchCommand == null)

\_searchCommand = new CommandViewModel(Search);

return \_searchCommand;

}

set

{

\_searchCommand = value;

}

}

protected ICommand \_clearSearchCommand;

public ICommand ClearSearchCommand

{

get

{

if (\_clearSearchCommand == null)

\_clearSearchCommand = new CommandViewModel(ClearSearchBox);

return \_clearSearchCommand;

}

set

{

\_clearSearchCommand = value;

}

}

#endregion

#endregion

#region Status Thumbs

#region Properties

#region Logics

public ObservableCollection<StatusDataModel> StatusThumbsCollection { get; set; }

#endregion

void LoadStatusThumbs()

{

StatusThumbsCollection = new ObservableCollection<StatusDataModel>()

{

new StatusDataModel{

IsMeAddStatus = true

},

new StatusDataModel{

ContactName="Майк",

ContactPhoto=new Uri("/Assets/1.png",UriKind.RelativeOrAbsolute),

StatusImage=new Uri("/Assets/5.jpg",UriKind.RelativeOrAbsolute),

IsMeAddStatus=false

},

new StatusDataModel{

ContactName="Стив",

ContactPhoto=new Uri("/Assets/2.jpg",UriKind.RelativeOrAbsolute),

StatusImage=new Uri("/Assets/8.jpg",UriKind.RelativeOrAbsolute),

IsMeAddStatus=false

},

new StatusDataModel{

ContactName="Вилл",

ContactPhoto=new Uri("/Assets/3.png",UriKind.RelativeOrAbsolute),

StatusImage=new Uri("/Assets/5.jpg",UriKind.RelativeOrAbsolute),

IsMeAddStatus=false

},

new StatusDataModel{

ContactName="Джон",

ContactPhoto=new Uri("/Assets/4.png",UriKind.RelativeOrAbsolute),

StatusImage=new Uri("/Assets/3.jpg",UriKind.RelativeOrAbsolute),

IsMeAddStatus=false

},

};

OnPropertyChanged("StatusThumbsCollection");

}

#endregion

#endregion

#region Chats List

#region Properties

public ObservableCollection<ChatListData> mChats;

public ObservableCollection<ChatListData> mPinnedChats;

public ObservableCollection<ChatListData> Chats

{

get => mChats;

set {

//to change the list

if (mChats == value) return;

//to update the list

mChats = value;

//updating filtered chats to match

FilteredChats = new ObservableCollection<ChatListData>(mChats);

OnPropertyChanged("Chats");

OnPropertyChanged("FilteredChats");

}

}

public ObservableCollection<ChatListData> PinnedChats

{

get => mPinnedChats;

set

{

//to change the list

if (mPinnedChats == value) return;

//to update the list

mPinnedChats = value;

//updating filtered chats to match

FilteredPinnedChats = new ObservableCollection<ChatListData>(mPinnedChats);

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredPinnedChats");

}

}

protected ObservableCollection<ChatListData> \_ArchiveChats;

public ObservableCollection<ChatListData> ArchiveChats

{ get => \_ArchiveChats;

set {

\_ArchiveChats = value;

OnPropertyChanged();

} }

//Filtering Chats, Pinned chats

public ObservableCollection<ChatListData> FilteredChats { get; set; }

public ObservableCollection<ChatListData> FilteredPinnedChats { get; set; }

protected int ChatPosition { get; set; }

#endregion

#region Logics

void LoadChats()

{

Chats = new ObservableCollection<ChatListData>()

{

new ChatListData

{

ContactName="Билл",

ContactPhoto=new Uri("/Assets/6.jpg",UriKind.RelativeOrAbsolute),

Message="Привет, как дела?",

LastMessageTime="вт, 15:28",

ChatIsSelected=true

},

new ChatListData

{

ContactName="Майк",

ContactPhoto=new Uri("/Assets/1.png",UriKind.RelativeOrAbsolute),

Message="Проверь почту",

LastMessageTime="пн, 12:01"

},

new ChatListData

{

ContactName="Стив",

ContactPhoto=new Uri("/Assets/7.png",UriKind.RelativeOrAbsolute),

Message="Да, было весело)",

LastMessageTime="вт, 08:10"

},

new ChatListData

{

ContactName="Джон",

ContactPhoto=new Uri("/Assets/8.jpg",UriKind.RelativeOrAbsolute),

Message="Как ты?",

LastMessageTime="вт, 01:00"

}

};

OnPropertyChanged();

}

#endregion

#region Commands

//to get the contactnaem of selected chat

protected ICommand \_getSelectedChatCommand;

public ICommand GetSelectedChatCommand => \_getSelectedChatCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatListData v)

{

//getting contactname from selected chat

ContactName = v.ContactName;

OnPropertyChanged("ContactName");

//getting contactphoto from selected chat

ContactPhoto = v.ContactPhoto;

OnPropertyChanged("ContactPhoto");

LoadChatConversation(v);

}

});

//to pin chat on pin button click

protected ICommand \_pinChatCommand;

public ICommand PinChatCommand => \_pinChatCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatListData v)

{

if (!FilteredPinnedChats.Contains(v))

{

//add selected chat to pin chat

PinnedChats.Add(v);

FilteredPinnedChats.Add(v);

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredPinnedChats");

v.ChatIsPinned = true;

//remove selected chat from all chats / unpinned chats

//store position of chat befor pinning sp that when we unoin or unarchive we get it on original position...

ChatPosition = Chats.IndexOf(v);

Chats.Remove(v);

FilteredChats.Remove(v);

OnPropertyChanged("Chats");

OnPropertyChanged("FilteredChats");

if(ArchiveChats!=null)

{

if (ArchiveChats.Contains(v))

{

ArchiveChats.Remove(v);

v.ChatIsArchive = false;

}

}

}

}

});

//to pin chat on pin button click

protected ICommand \_unPinChatCommand;

public ICommand UnPinChatCommand => \_unPinChatCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatListData v)

{

if (!FilteredPinnedChats.Contains(v))

{

//restore position of chat befor pinning sp that when we unoin or unarchive we get it on original position...

//remove selected chat from all chats / unpinned chats

Chats.Add(v);

FilteredChats.Add(v);

//restore position of chat befor pinning sp that when we unoin or unarchive we get it on original position...

Chats.Move(Chats.Count-1, ChatPosition);

FilteredChats.Move(Chats.Count-1, ChatPosition);

//update

OnPropertyChanged("Chats");

OnPropertyChanged("FilteredChats");

//add selected chat to pin chat

PinnedChats.Remove(v);

FilteredPinnedChats.Remove(v);

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredPinnedChats");

v.ChatIsPinned = false;

}

}

});

protected ICommand \_archiveChatCommand;

public ICommand ArchiveChatCommand => \_archiveChatCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatListData v)

{

if (!ArchiveChats.Contains(v))

{

//add chat in archive list

ArchiveChats.Add(v);

v.ChatIsArchive = true;

v.ChatIsPinned= false;

//remove chat from pinned and unpinned chat list

Chats.Remove(v);

FilteredChats.Remove(v);

PinnedChats.Remove(v);

FilteredPinnedChats.Remove(v);

//update list

OnPropertyChanged("Chats");

OnPropertyChanged("FilteredChats");

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredPinnedChats");

OnPropertyChanged("ArchiveChats");

}

}

});

protected ICommand \_unArchiveChatCommand;

public ICommand UnArchiveChatCommand => \_unArchiveChatCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatListData v)

{

if (!FilteredChats.Contains(v) && !Chats.Contains(v))

{

//remove chat from pinned and unpinned chat list

Chats.Add(v);

FilteredChats.Add(v);

}

//add chat in archive list

ArchiveChats.Remove(v);

v.ChatIsArchive = false;

v.ChatIsPinned = false;

OnPropertyChanged("Chats");

OnPropertyChanged("FilteredChats");

OnPropertyChanged("ArchiveChats");

}

});

#endregion

#endregion

#region Conversations

#region Properties

protected ObservableCollection<ChatConversation> mConversations;

public ObservableCollection<ChatConversation> Conversations

{

get => mConversations;

set

{

//to change the list

if (mConversations == value) return;

//to update the list

mConversations = value;

//updating filtered chats to match

FilteredConversations = new ObservableCollection<ChatConversation>(mConversations);

OnPropertyChanged("Conversations");

OnPropertyChanged("FilteredConversations");

}

}

public ObservableCollection<ChatConversation> FilteredConversations { get; set; }

protected string messageText;

public string MessageText

{

get => messageText;

set

{

messageText = value;

OnPropertyChanged("MessageText");

}

}

protected string LastSearchConversationText;

protected string mSearchConversationText;

public string SearchConversationText

{

get => mSearchConversationText;

set

{

//checked if value is different

if (mSearchConversationText == value) { return; }

//update view

mSearchConversationText = value;

//if search text is empty restore message

if (string.IsNullOrEmpty(SearchConversationText)) SearchInConversation();

}

}

public bool FocusMessageBox { get; set; }

public bool IsThisAReplyMessage { get; set; }

public string MessageToReplyText { get; set; }

#endregion

#region Logics

protected bool \_isConversationSearchBoxOpen;

public bool IsConversationSearchBoxOpen

{

get => \_isConversationSearchBoxOpen;

set

{

if (\_isConversationSearchBoxOpen == value) return;

\_isConversationSearchBoxOpen = value;

if (\_isConversationSearchBoxOpen == false) SearchConversationText = string.Empty;

OnPropertyChanged("IsConversationSearchBoxOpen");

OnPropertyChanged("SearchConversationText");

}

}

public void OpenConversationSearchBox()

{

IsConversationSearchBoxOpen = true;

}

public void ClearConversationSearchBox()

{

if (!string.IsNullOrEmpty(SearchConversationText))

SearchConversationText = string.Empty;

else CloseConversationSearchBox();

}

public void CloseConversationSearchBox() => IsConversationSearchBoxOpen = false;

private void LoadChatConversation( ChatListData chat)

{

if (connection.State == System.Data.ConnectionState.Closed)

connection.Open();

if(Conversations == null)

Conversations = new ObservableCollection<ChatConversation>();

Conversations.Clear();

FilteredConversations.Clear();

using (SqlCommand com = new SqlCommand("select \* from conversations where ContactName=@ContactName", connection))

{

com.Parameters.AddWithValue("@ContactName",chat.ContactName);

using (SqlDataReader reader = com.ExecuteReader())

{

while (reader.Read())

{

string MsgReceivedOn = !string.IsNullOrEmpty(reader["MsgReceivedOn"].ToString()) ?

Convert.ToDateTime(reader["MsgReceivedOn"].ToString()).ToString("MMM dd, hh:mm tt") : "";

string MsgSentOn = !string.IsNullOrEmpty(reader["MsgSentOn"].ToString()) ?

Convert.ToDateTime(reader["MsgSentOn"].ToString()).ToString("MMM dd, hh:mm tt") : "";

var conversation = new ChatConversation()

{

ContactName = reader["ContactName"].ToString(),

ReceivedMessage = reader["ReceivedMsgs"].ToString(),

MsgReceivedOn = MsgReceivedOn,

MsgSentOn = MsgSentOn,

SentMessage = reader["SentMsgs"].ToString(),

IsMessageReceived = !string.IsNullOrEmpty(reader["ReceivedMsgs"].ToString())

};

Conversations.Add(conversation);

OnPropertyChanged("Conversations");

FilteredConversations.Add(conversation);

OnPropertyChanged("FilteredConversations");

chat.Message = !string.IsNullOrEmpty(reader["ReceivedMsgs"].ToString())? reader["ReceivedMsgs"].ToString():

reader["SentMsgs"].ToString();

}

}

}

//reset reply message text when the new chat is fetched

MessageToReplyText=string.Empty;

OnPropertyChanged("MessageToReplyText");

}

void SearchInConversation()

{

//to avoid searching same text again

if ((string.IsNullOrEmpty(LastSearchConversationText) && string.IsNullOrEmpty(SearchConversationText)) || string.Equals(LastSearchConversationText, SearchConversationText))

{ return; }

//if searchbox is empty or chat in null pr char cound less then 0

if (string.IsNullOrEmpty(SearchConversationText) || Conversations == null || Conversations.Count <= 0)

{

FilteredConversations = new ObservableCollection<ChatConversation>(Conversations ?? Enumerable.Empty<ChatConversation>());

OnPropertyChanged("FilteredConversations");

//update last search text

LastSearchConversationText = SearchConversationText;

return;

}

//now? to find all chats that contain the text in our search box

FilteredConversations = new ObservableCollection<ChatConversation>(

Conversations.Where(

chat => chat.ReceivedMessage.ToLower().Contains(SearchConversationText)

||

chat.SentMessage.ToLower().Contains(SearchConversationText)

));

OnPropertyChanged("FilteredConversations");

//update last search text

LastSearchConversationText = SearchConversationText;

}

public void CancelReply()

{

IsThisAReplyMessage = false;

//reset reply msg text

MessageToReplyText = string.Empty;

OnPropertyChanged("MessageToReplyText");

}

public void SendMessage()

{

//send message only when the textbox is not empty

if(!string.IsNullOrEmpty(MessageText))

{

var conversation = new ChatConversation()

{

ReceivedMessage = MessageToReplyText,

SentMessage = MessageText,

MsgSentOn = DateTime.Now.ToString("MMM dd, hh:mm tt"),

MessageContainsReply = IsThisAReplyMessage

};

//add msg to conversation list

FilteredConversations.Add(conversation);

Conversations.Add(conversation);

//clear message properties and textbox when message is sent

MessageText=string.Empty;

IsThisAReplyMessage = false;

MessageToReplyText= string.Empty;

UpdateChatAndMoveUp(Chats, conversation);

UpdateChatAndMoveUp(PinnedChats, conversation);

UpdateChatAndMoveUp(FilteredChats, conversation);

UpdateChatAndMoveUp(FilteredPinnedChats, conversation);

UpdateChatAndMoveUp(ArchiveChats, conversation);

//update

OnPropertyChanged("FilteredConversations");

OnPropertyChanged("Conversations");

OnPropertyChanged("MessageText");

OnPropertyChanged("IsThisAReplyMessage");

OnPropertyChanged("MessageToReplyText");

}

}

//move the chat contact on top when new message is sent or recieved

protected void UpdateChatAndMoveUp(ObservableCollection<ChatListData> chatList, ChatConversation conversation)

{

//check if the message sent is to selected contact or not

var chat = chatList.FirstOrDefault(x=>x.ContactName == ContactName);

//if found.. then

if(chat != null)

{

//Update Contact Chat Last Message and Messahe Time..

chat.Message = MessageText;

chat.LastMessageTime = conversation.MsgSentOn;

//move chat on top when new msg is recieved/sent...

chatList.Move(chatList.IndexOf(chat), 0);

//update collection

OnPropertyChanged("Chats");

OnPropertyChanged("PinnedChats");

OnPropertyChanged("FilteredChats");

OnPropertyChanged("FilteredPinnedChats");

OnPropertyChanged("ArchiveChats");

}

}

#endregion

#region Commands

protected ICommand \_openConversationSearchCommand;

public ICommand OpenConversationSearchCommand

{

get

{

if (\_openConversationSearchCommand == null)

\_openConversationSearchCommand = new CommandViewModel(OpenConversationSearchBox);

return \_openConversationSearchCommand;

}

set

{

\_openConversationSearchCommand = value;

}

}

protected ICommand \_clearConversationSearchCommand;

public ICommand ClearConversationSearchCommand

{

get

{

if (\_clearConversationSearchCommand == null)

\_clearConversationSearchCommand = new CommandViewModel(ClearConversationSearchBox);

return \_clearConversationSearchCommand;

}

set

{

\_searchConversationCommand = value;

}

}

protected ICommand \_closeConversationSearchCommand;

public ICommand CloseConversationSearchCommand

{

get

{

if (\_closeConversationSearchCommand == null)

\_closeConversationSearchCommand = new CommandViewModel(CloseConversationSearchBox);

return \_closeConversationSearchCommand;

}

set

{

\_searchConversationCommand = value;

}

}

protected ICommand \_searchConversationCommand;

public ICommand SearchConversationCommand

{

get

{

if (\_searchConversationCommand == null)

\_searchConversationCommand = new CommandViewModel(SearchInConversation);

return \_searchConversationCommand;

}

set

{

\_searchConversationCommand = value;

}

}

protected ICommand \_replyCommand;

public ICommand ReplyCommand => \_replyCommand ?? new RelayCommand(parameter =>

{

if (parameter is ChatConversation v)

{

//if replying sender's message

if (v.IsMessageReceived)

MessageToReplyText = v.ReceivedMessage;

//if replying own message

else

MessageToReplyText = v.SentMessage;

//update

OnPropertyChanged("MessageToReplyText");

//set focus on message box when user clicks reply button

FocusMessageBox = true;

OnPropertyChanged("FocusMessageBox");

//flag this msg as reply msg

IsThisAReplyMessage = true;

OnPropertyChanged("IsThisAReplyMessage");

}

});

protected ICommand \_cancelReplyCommand;

public ICommand CancelReplyCommand

{

get

{

if (\_cancelReplyCommand == null)

\_cancelReplyCommand = new CommandViewModel(CancelReply);

return \_cancelReplyCommand;

}

set

{

\_cancelReplyCommand = value;

}

}

protected ICommand \_sendMessageCommand;

public ICommand SendMessageCommand

{

get

{

if (\_sendMessageCommand == null)

\_sendMessageCommand = new CommandViewModel(SendMessage);

return \_sendMessageCommand;

}

set

{

\_sendMessageCommand = value;

}

}

#endregion

#endregion

#region ContactInfo

#region Properties

protected bool \_IsContactInfoOpen;

public bool IsContactInfoOpen

{

get=> \_IsContactInfoOpen;

set { \_IsContactInfoOpen = value;

OnPropertyChanged("IsContactInfoOpen");

}

}

#endregion

#region Logics

public void OpenContactInfo()=>IsContactInfoOpen=true;

public void CloseContactInfo() => IsContactInfoOpen = false;

#endregion

#region Commands

protected ICommand \_openContactInfoCommand;

public ICommand OpenContactInfoCommand

{

get

{

if (\_openContactInfoCommand == null)

\_openContactInfoCommand = new CommandViewModel(OpenContactInfo);

return \_openContactInfoCommand;

}

set

{

\_openContactInfoCommand = value;

}

}

protected ICommand \_closeContactInfoCommand;

public ICommand CloseContactInfoCommand

{

get

{

if (\_closeContactInfoCommand == null)

\_closeContactInfoCommand = new CommandViewModel(CloseContactInfo);

return \_closeContactInfoCommand;

}

set

{

\_closeContactInfoCommand = value;

}

}

#endregion

#endregion

SqlConnection connection = new SqlConnection(@"Data Source=LAPTOP-S3L918JB;Initial Catalog=Database;Integrated Security=True;Connect Timeout=30;Encrypt=False;TrustServerCertificate=False;ApplicationIntent=ReadWrite;MultiSubnetFailover=False");

public ViewModel()

{

LoadStatusThumbs();

LoadChats();

PinnedChats = new ObservableCollection<ChatListData>();

ArchiveChats = new ObservableCollection<ChatListData>();

}

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)

{

PropertyChanged?.Invoke(this,new PropertyChangedEventArgs(propertyName));

}

}

}

2. Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.ServiceModel;

namespace ChatHost

{

class Program

{

static void Main(string[] args)

{

using (var host = new ServiceHost(typeof(wcf\_chat.ServiceChat)))

{

host.Open();

Console.WriteLine("host is started!");

Console.ReadLine();

}

}

}

}

3. IServiceChat.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

namespace wcf\_chat

{

// ПРИМЕЧАНИЕ. Команду "Переименовать" в меню "Рефакторинг" можно использовать для одновременного изменения имени интерфейса "IServiceChat" в коде и файле конфигурации.

[ServiceContract(CallbackContract = typeof(IServerChatCallback))]

public interface IServiceChat

{

[OperationContract]

int Connect(string name);

[OperationContract]

void Disconnect(int id);

[OperationContract(IsOneWay = true)]

void SendMsg(string msg, int id);

}

public interface IServerChatCallback

{

[OperationContract(IsOneWay = true)]

void MsgCallback(string msg);

}

}

4. ServerUser.cs

using System.ServiceModel;

namespace wcf\_chat

{

public class ServerUser

{

public int ID { get; set; }

public string Name { get; set; }

public OperationContext operationContext { get; set; }

}

}

5. ServiceChat.cs using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.Serialization;

using System.ServiceModel;

using System.Text;

namespace wcf\_chat

{

[ServiceBehavior(InstanceContextMode = InstanceContextMode.Single)]

public class ServiceChat : IServiceChat

{

List<ServerUser> users = new List<ServerUser>();

int nextId = 1;

public int Connect(string name)

{

ServerUser user = new ServerUser()

{

ID = nextId,

Name = name,

operationContext = OperationContext.Current

};

nextId++;

SendMsg(": " + user.Name + " was added", 0);

users.Add(user);

return user.ID;

}

public void Disconnect(int id)

{

var user = users.FirstOrDefault(i => i.ID == id);

if (user != null)

{

users.Remove(user);

SendMsg(": " + user.Name + " left", 0);

}

}

public void SendMsg(string msg, int id)

{

foreach (var item in users)

{

string answer = DateTime.Now.ToShortTimeString();

var user = users.FirstOrDefault(i => i.ID == id);

if (user != null)

{

answer += ": " + user.Name + " ";

}

answer += msg;

item.operationContext.GetCallbackChannel<IServerChatCallback>().MsgCallback(answer);

}

}

}

}