МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий
Кафедра программной инженерии
Специальность 1-40 01 01 Программное обеспечение информационных технологий
Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет приложений)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
КУРСОВОГО ПРОЕКТ:

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»
Тема Программное средство «Мессенджер X-Messaging»

Исполнитель
студент (ка) 2 курса группы 5 Гайков Дмитрий Викторович
(Ф.И.О.)

Руководитель работы преп.-стажер Север А.С.
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Председатель Пацей Н.В.
(подпись)

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»
Факультет информационных технологий
Кафедра программной инженерии

Утверждаю
Заведующий кафедрой ПИ
_____ __Н.В. Пацей
подпись инициалы и фамилия
“___”_____2023г.

ЗАДАНИЕ

к курсовому проектированию

по дисциплине "Объектно-ориентированные технологии программирования и стандарты проектирования"

Специальность: 1-40 01 01 Программное обеспечение Группа: __5__ информационных технологий

Студент: Гайков Дмитрий Викторович

Тема: Программное средство «X-Messenger»

1. Срок сдачи студентом законченной работы: "20 мая 2023 г."

2. Исходные данные к проекту:

2.1. Функционально ПС поддерживает:

- Функции администратора:
  - Просмотр данных всех пользователей;
  - Удаление аккаунтов;
  - Добавление и удаление стикеров;
  - Просмотр статистики приложения;
  - Рассылка сообщений всем пользователям.
- Функции клиента:
  - Выполнять регистрацию и авторизацию;
  - Редактирование профиля;
  - Просмотр информации о пользователях;
  - Отправка текстовых, голосовых или в виде стикеров сообщений;
  - Удаление или редактирование сообщений.

2.2. При выполнении курсового проекта необходимо использовать принципы проектирования ООП. Приложение разрабатывается под ОС Windows и представляет собой настольное приложение (desktop). Отображение, бизнес логика должны быть максимально независимы друг от друга для возможности расширения. Диаграммы вариантов использования, классов реализации задачи, взаимодействия разработать на основе UML. Язык разработки проекта – C#. Управление программой должно быть интуитивно понятным и удобным. Использовать архитектурные и шаблоны проектирования.

3. Содержание расчетно-пояснительной записки

(перечень вопросов подлежащих разработке)

- Введение

- Постановка задачи и обзор литературы (алгоритмы решения, обзор прототипов, актуальность задачи)
- Проектирование архитектуры проекта (структура модулей, классов).
- Разработка функциональной модели и модели данных ПС (выполняемые функции)
- Руководство пользователя
- Тестирование
- Заключение
- Список используемых источников
- Приложения

4. Форма представления выполненной курсовой работы:

- Теоретическая часть курсового проекта должны быть представлены в формате docx. Оформление записки должно быть согласно выданным правилам.
- Листинги программы представляются в приложении.
- Пояснительную записку, листинги, проект (инсталляцию проекта) необходимо загрузить диск, указанный преподавателем.

Календарный план

№ п/п	Наименование этапов курсового проекта	Срок выполнения этапов проекта	Примечание
1	Введение	19.02.2023	
2	Аналитический обзор литературы по теме проекта. Изучение требований, определение вариантов использования	12.03.2023	
3	Анализ и проектирование архитектуры приложения (построение диаграмм, проектирование бизнес-слоя, представления и данных)	26.03.2023	
4	Проектирование структуры базы данных. Разработка дизайна пользовательского интерфейса	02.04.2023	
5	Кодирование программного средства	23.04.2023	
6	Тестирования и отладка программного средства	30.04.2023	
7	Оформление пояснительной записки	07.05.2023	
8	Защита проекта	20.05.2023	

5. Дата выдачи задания 12.02.2023

Руководитель_____ *Север А.С.*
(подпись)

Задание принял к исполнению 12.02.2023_____ *Гайков Д.В.*
(дата и подпись студента)

Содержание

ВВЕДЕНИЕ 6
1 Аналитический обзор литературы и формирование требований 7
1.1 Анализ прототипов 7
1.2 Требования к проекту 9
2 Анализ требований к программному средству и разработка функциональных требований 10
2.1 Описание средств разработки 10
2.1.1 Microsoft Visual Studio 2022 10
2.1.2 Программная платформа .NET 7 10
2.1.3 Язык программирования C# 10
2.1.4 Технология WPF 10
2.1.5 Расширяемый язык разметки XAML 11
2.1.6 Технология ADO.NET 11
2.1.7 MS SQL Server 11
2.1.8 Библиотека NAudio 11
2.1.9 Паттерн MVVM 12
2.2 Спецификация функциональных требований к программному средству 12
2.3 Спецификация функциональных требований 13
3 Проектирование программного средства 14
3.1 Общая структура 14
3.2 Взаимоотношение между классами 19
3.3 Модель базы данных 19
3.4 Проектирование архитектуры приложения 21
3.5 Проектирование последовательностей проекта 22
4 Реализация программного средства 23
4.1 Основные классы программного средства 23
4.2 Авторизация 23
4.3 Регистрация 24
4.4 Редактирование профиля 25
4.5 Отправка сообщений 25
4.6 Панель администратора 25
5 Тестирование, проверка работоспособности и анализ полученных результатов 26

5.1 Тестирование регистрации 26
5.2 Тестирование отправки сообщения и работы с сообщениями 27
5.3 Тестирование панели администратора 28
6 Руководство по установке и использованию 30
ЗАКЛЮЧЕНИЕ 33
Список использованных источников 34
ПРИЛОЖЕНИЕ А 35
ПРИЛОЖЕНИЕ Б 36
ПРИЛОЖЕНИЕ В 37

ВВЕДЕНИЕ

Данный курсовой проект посвящён разработке программного средства «X-Messenger», основной целью которого является позволить людям общаться, находившись в разных точках мира. Данное приложение позволит расширить бизнес-сеть за счет того, что заказчики легко смогут связаться с поставщиками и обсудить все детали заказа.

Пользователи такого приложения смогут заняться улучшение своего аккаунта за счет выполнения внутренних заданий, за которые можно получить внутреннею валюту, за которую можно будет купить стикеры, улучшения для сообщений таких, как особая подсветка сообщения, смайлики в имени.

Главная задача данного курсового проектирования – это разработка программного средства, которое реализует все вышеперечисленные функции и решает поставленные задачи. Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП, база данных MS SQL Server, и технология Windows Presentation Foundation (WPF).

Кроме того, при разработке программного средства "X-Messenger" будут использоваться следующие принципы и методологии:

1. SOLID — это набор принципов объектно-ориентированного программирования, которые направлены на улучшение качества кода, уменьшение его сложности и повышение гибкости приложения. Принципы SOLID помогают создавать более легко сопровождаемый и расширяемый код.

2. Agile — это методология разработки программного обеспечения, которая направлена на ускорение процесса разработки и увеличение качества продукта. Agile предполагает частые итерации разработки, постоянное взаимодействие с заказчиком и быструю адаптацию к изменяющимся требованиям.

1 Аналитический обзор литературы и формирование требований

1.1 Анализ прототипов

Были проанализированы цели и задачи, поставленные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

Первый аналог — WhatsApp.

WhatsApp обладает широким спектром функций, включая отправку текстовых сообщений, голосовых сообщений, видеозвонки, групповые чаты, обмен файлами и стикерами, а также возможность создания и управления каналами. Этот мессенджер также позволяет пользователям отправлять местоположение и документы.

WhatsApp имеет простой и удобный интерфейс, который легко использовать. Дизайн этого мессенджера состоит из основных функций в левой части экрана и списка чатов в правой части. WhatsApp также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид мессенджера.

Дизайн WhatsApp - довольно минималистичный и простой в использовании. Цветовая схема состоит в основном из белого и зеленого, что создает ощущение легкости и чистоты.

Блок сообщения сделан закругленным. Текст для имени пользователя подсвечивается другим цветом. Текст черного цвета на белом фоне. На данное сообщение довольно-таки приятно смотреть.

Интерфейс приложения представлен на рисунке 1.1.

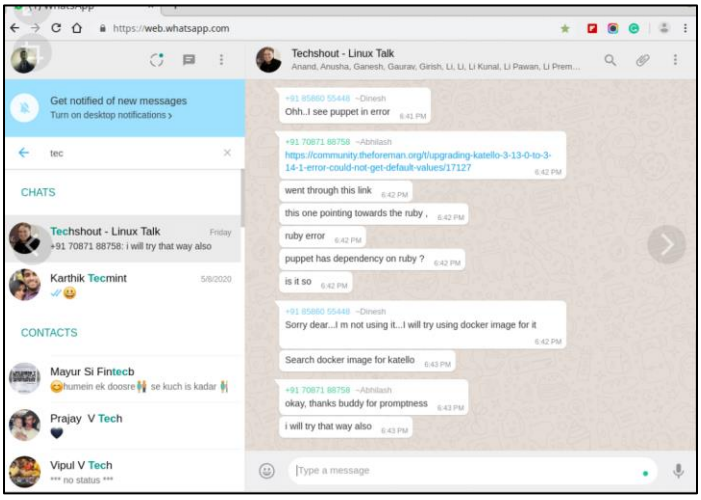


Рисунок 1.1 – Приложение «WhatsApp»

Второй аналог — это приложение Viber.

Viber позволяет отправлять текстовые сообщения, стикеры, фотографии, видео и аудиофайлы. Вы также можете создавать групповые чаты для общения с несколькими людьми одновременно. Viber позволяет совершать бесплатные голосовые звонки и видеозвонки с другими пользователями Viber по всему миру.

Для этого необходимо, чтобы оба пользователя были подключены к интернету. имеет светлый интерфейс с фиолетовыми элементами дизайна.

Главный экран отображает список ваших чатов, а также кнопки для совершения голосовых и видеозвонков. В верхней части экрана расположены кнопки для доступа к контактам, магазину стикеров и настройкам. В целом, интерфейс Viber интуитивно понятен и удобен в использовании.

Дизайн Viber представляет собой современный и лаконичный интерфейс с использованием ярких цветов и плоских иконок. Он имеет минималистичный дизайн, который делает приложение легким и простым в использовании.

Страница пользователя имеет картинку человека, на которого мы зашли. Кнопки в нем подсвечены и выделяются на общем фоне. Сразу видно, куда жать.

Интерфейс приложения представлен на рисунке 1.2.

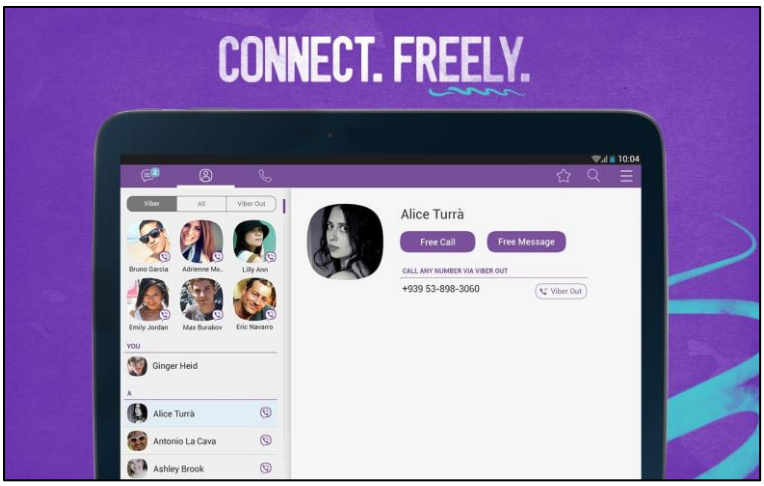


Рисунок 1.2 – Приложение «Viber»

Третий аналог — это Telegram.

Telegram обладает широким спектром функций, включая отправку текстовых сообщений, голосовых сообщений, видеозвонки, групповые чаты, демонстрация экрана, обмен файлами и стикерами, а также возможность создания каналов и ботов. Этот мессенджер также предлагает возможность создания секретных чатов с функцией автоматического удаления сообщений.

Telegram имеет простой и удобный интерфейс, который легко использовать. Дизайн этого мессенджера состоит из основных функций в нижней части экрана и списка чатов в верхней части. Telegram также предлагает различные темы, которые пользователи могут выбрать, чтобы изменить внешний вид мессенджера.

Дизайн Telegram отличается от других мессенджеров своей минималистичностью и простотой в использовании. Он имеет темно-синий цветовой фон, который выделяет контент на экране. В верхней части экрана расположено главное меню, которое позволяет быстро переключаться между разделами мессенджера. В центре экрана находится список чатов и диалогов. Каждый чат представлен в виде миниатюрного значка с изображением фото профиля пользователя или группы, а также отображается последнее сообщение. При

нажатии на любой из чатов, открывается окно переписки, которое также имеет темно-синий фон и белый шрифт. Дизайн приятный и легко пользоваться им.

Интерфейс приложения представлен на рисунке 1.3.

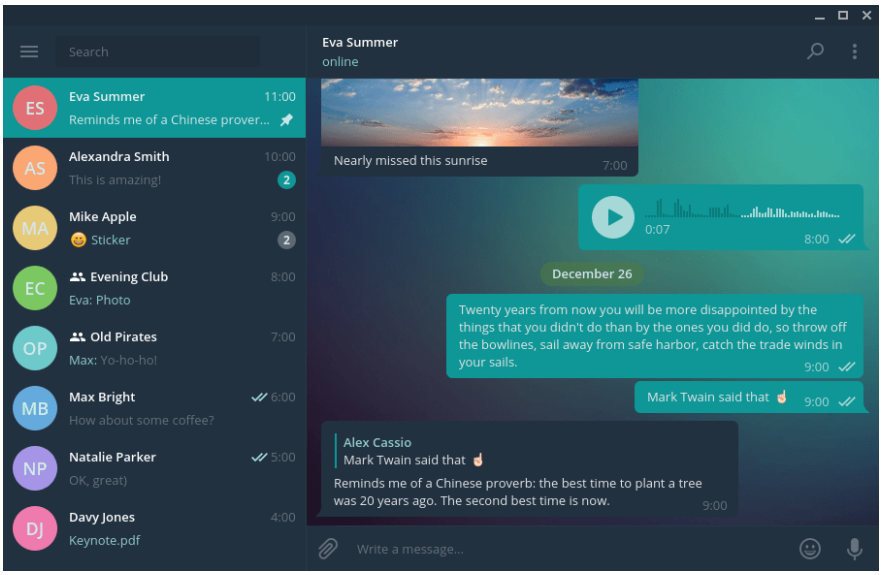


Рисунок 1.3 – Приложение «Telegram»

Анализируя аналоги WhatsApp, Viber и Telegram, можно сделать вывод, что все три приложения предлагают широкий спектр функций, включая обмен сообщениями, голосовые и видеозвонки, групповые чаты и возможность отправки файлов. Каждое из приложений имеет свою уникальную цветовую схему и дизайн интерфейса, но общий тренд — простота и удобство использования. Все они предлагают настраиваемый интерфейс с возможностью выбора тем оформления

1.2 Требования к проекту

Обзор вышеперечисленных известных аналогов позволяет проанализировать все преимущества и недостатки альтернативных возможностей и позволяет сформулировать список требований, предъявляемых к программному средству, разрабатываемому в данном курсовом проекте.

Программное средство должно обеспечивать возможность выполнения перечисленных ниже функций:

– управление администратором базой данных;

– возможность пользователю зарегистрироваться или войти в существующую учетную запись;

– авторизированным пользователям общаться с другими авторизированными пользователями;

– возможность загрузки изображений или других файлов.

2 Анализ требований к программному средству и разработка функциональных требований

2.1 Описание средств разработки

При разработке приложения были использованы:

- интегрированная среда разработки Microsoft Visual Studio 2022;
- программная платформа .NET 7.0;
- язык программирования C#;
- расширяемый язык разметки XAML;
- технология WPF;
- технология ADO.NET;
- MS SQL Server.

2.1.1 Microsoft Visual Studio 2022

Microsoft Visual Studio 2022 — это интегрированная среда разработки для написания, отладки и сборки кода, а также последующей публикации приложений. Данный продукт позволяет разрабатывать не только консольные, но и десктопные приложения, с использованием таких технологий, как WinForms или WPF.

2.1.2 Программная платформа .NET 7

Платформа .NET - это программный фреймворк, разработанный компанией Microsoft, который в основном работает на операционной системе Microsoft Windows. Он предоставляет среду выполнения для запуска приложений и набор библиотек, которые используются для создания программных приложений. Фреймворк .NET состоит из нескольких компонентов, включая общую языковую среду (Common Language Runtime - CLR), библиотеку классов .NET Framework и различные инструменты разработки.

2.1.3 Язык программирования C#

В качестве языка программирования используется C# – основной язык разработки в .NET. Язык объектно-ориентированный, имеет строгую статическую типизацию, поддерживает перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения. Используется как основной язык в технологии WPF.

2.1.4 Технология WPF

Для предоставления пользовательского интерфейса и разграничения дизайна и бизнес-логики используется технология Microsoft WPF – аналог WinForms, система для построения клиентских приложений Windows с возможностями

взаимодействия с пользователем и графическая подсистема в составе .NET, использующая язык разметки XAML.

2.1.5 Расширяемый язык разметки XAML

WPF предоставляет средства для создания визуального интерфейса, включая язык XAML (eXtensible Application Markup Language элементы управления, привязку данных, макеты, двухмерную и трёхмерную графику, анимацию, стили, шаблоны, документы, текст, мультимедиа и оформление. XAML представляет собой язык декларативного описания интерфейса, основанный на XML.

2.1.6 Технология ADO.NET

ADO.NET - это технология доступа к данным в .NET Framework. Она предоставляет программистам возможность работать с различными источниками данных, такими как реляционные базы данных, XML-документы, объекты и т.д.

ADO.NET использует набор классов, которые предоставляют доступ к данным и возможность их изменения. Он включает в себя классы для подключения к источникам данных, чтения и записи данных, выполнения команд SQL и многое другое.

ADO.NET предоставляет возможность работать с данными как в синхронном, так и в асинхронном режиме. Это позволяет создавать приложения, которые могут эффективно обрабатывать большие объемы данных и выполнять запросы к базам данных без блокировки пользовательского интерфейса.

2.1.7 MS SQL Server

Для организации баз данных MS SQL Server использует реляционную модель, которая предполагает хранение данных в виде таблиц, каждая из которых состоит из строк и столбцов. Каждая строка хранит отдельный объект, а в столбцах размещаются атрибуты этого объекта. Для взаимодействия с базой данных применяется язык SQL (Structured Query Language). Клиент (например, внешняя программа) отправляет запрос 10 на языке SQL должным образом интерпретирует и выполняет запрос, а затем посылает клиенту результат выполнения. Основной используемый язык запросов — Transact-SQL — реализован на структурированном языке запросов (SQL) с расширениями.

2.1.8 Библиотека NAudio

NAudio - это библиотека для работы со звуком в языке программирования C#. Она предоставляет разработчикам возможность осуществлять запись, воспроизведение и обработку звуковых файлов, а также работать с аудио-устройствами, такими как микрофоны, звуковые карты и динамики.

Библиотека NAudio поддерживает следующие форматы звуковых файлов: WAV, MP3, AIFF, AU, FLAC, OGG, AAC, WMA и многие другие. Она также

поддерживает использование аудио-кодеков для обработки и конвертации звуковых файлов.

Кроме того, библиотека NAudio имеет открытый исходный код и документацию, что делает ее удобной для использования и расширения. Она также поддерживает работу в Windows Forms и WPF, что позволяет создавать приложения с графическим интерфейсом для работы со звуком.

2.1.9 Паттерн MVVM

MVVM (Model-View-ViewModel) - это паттерн архитектуры, используемый в разработке программного обеспечения для разделения пользовательского интерфейса (View) от бизнес-логики (Model) и связывания их через промежуточный слой ViewModel.

В MVVM модель представляет данные и бизнес-логику приложения, которая может включать в себя операции чтения/записи данных из и в источники данных, такие как база данных или веб-сервисы.

Представление (View) отвечает за отображение данных и взаимодействие с пользователем. Оно не содержит бизнес-логику и должно быть максимально независимым от модели.

ViewModel представляет промежуточный слой между View и Model. Он содержит логику, необходимую для обработки пользовательских действий, обновления данных в модели и уведомления View об изменениях. ViewModel предоставляет свойства и команды, которые привязываются к элементам пользовательского интерфейса во View, позволяя им отображать данные и реагировать на действия пользователя.

MVVM позволяет достичь разделения ответственности между компонентами приложения, облегчает тестирование и повышает переиспользуемость кода. Он также способствует лучшей поддержке параллельной разработки пользовательского интерфейса и бизнес-логики, так как разработчики могут работать независимо над своими частями приложения.

2.2 Спецификация функциональных требований к программному средству

Программное средство должно предоставлять следующие функциональные возможности:

Для пользователя:

- регистрация;
- авторизация;
- редактирование профиля;
- отправка голосовых, текстовых и в виде стикеров сообщений;
- отправка изображений;
- персонализация приложения;
- просмотр информации о других пользователях;

– поиск пользователей;
– редактирование и удаление сообщений.

Для администратора:

– авторизация;
– ответ пользователям;
– рассылка сообщений;
– просмотр статистики приложения;
– просмотр всех аккаунтов;
– удаление аккаунтов;
– добавление и удаление стикеров.

2.3 Спецификация функциональных требований

Для функциональности ПС необходимо создание базы данных для хранения информации приложения. Подробно база данных описано в следующем разделе.

В программном средстве необходимо реализовать регистрацию и авторизацию пользователей для доступа ко всем возможностям приложения. Для авторизации входными параметрами являются логин и пароль пользователя, которые содержатся в базе данных. Для регистрации входными данными являются имя, фамилия, логин, пароль, e-mail. Введенные данные, успешно прошедшие валидацию, заносятся в базу данных.

Пользователь может отправлять сообщения другим людям, поэтому надо хранить информацию о всех сообщениях.

Администратор обладает теми же возможностями, что и пользователь, а также обладает возможностями добавления, удаления стикеров. Также он может ограничивать возможность пользоваться данным приложением.

Описание функциональности программного средства «X-Messenger» представлено на UML-схеме, изображенной на рисунке 2.1.

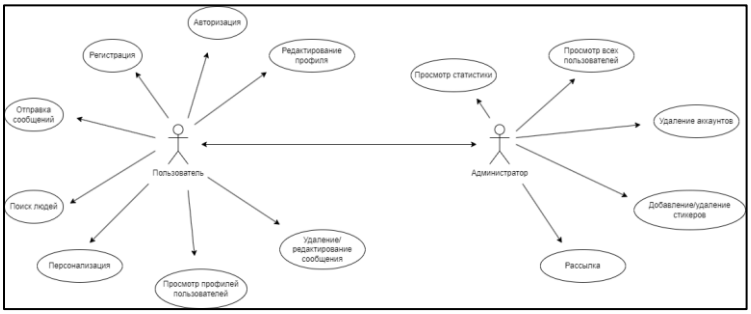


Рисунок 2.1 — UML-схема

Разграничение прав пользователей, включая администратора, позволяет эффективно управлять доступом и ограничивать возможности использования приложения в соответствии с требованиями и политиками.

3 Проектирование программного средства

3.1 Общая структура

Программное средство «X-Messenger» имеет следующею структуру, представленную на рисунке 3.1.

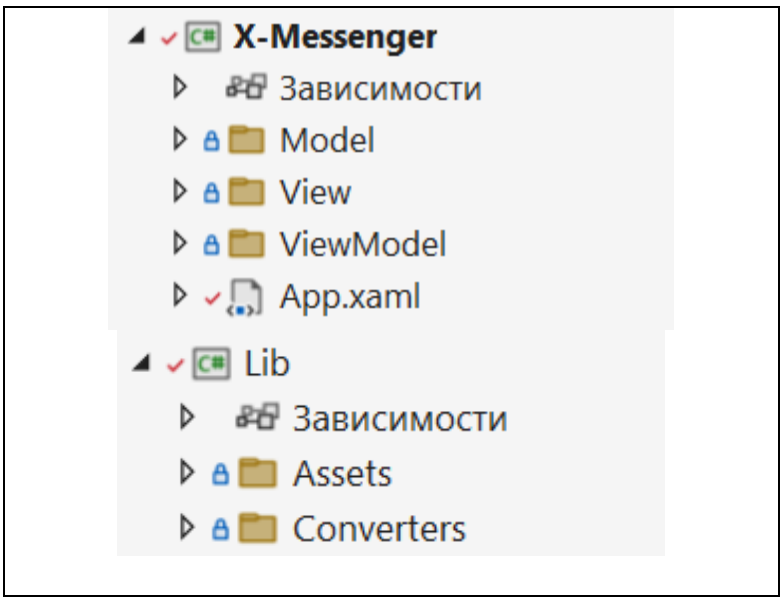


Рисунок 3.1 – Структура проекта

Описание структуры основных папок и файлов проекта и библиотеки классов представлено в таблице 3.1.

Таблица 3.1 – Описание структуры папок и файлов проекта

Имя пакета	Содержание
Папка Model	Здесь описаны модели, с которыми происходит вся работа в приложении: – Клиент; – Пользователь; – База данных; – Объекты для работы с БД; – Всякие классы преобразователи.
Папка View	В этой папке храняться файлы с xaml-разметкой, то есть окна, страницы, для их словари ресурсов с стилями.
Папка ViewModel	В данной папке хранятся все классы для взаимодействия графической составляющей и бизнес логики приложения.

Продолжение таблицы 3.1

Имя пакета	Содержание
Папка Assets	Содержит вспомогательные классы и интерфейсы для работы в приложении. Классы и интерфейсы используются для реализации разных паттернов.
Папка Converters	Содержит класс для преобразования данных из одного вида в другой.
App.xaml	Файл, который определяет ресурсы, которые являются общие для приложения

Более подробная структура содержимого папки «Model» программного средства показана на рисунке 3.2.

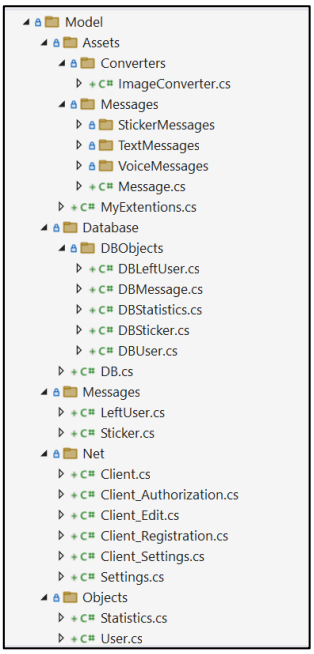


Рисунок 3.2 — Подробная структура папки «Model»

Описание файлов и внутренних папок папки представлено в таблице 3.2.

Таблица 3.2 – Описание файлов и внутренних папок папки «Model»

Имя файла/папки	Содержание
Папка Assets	Папка хранит в себе дополнительне классы для работы в приложении. Одни из классов — это преобразователь фотографий в байты и наоборот.
Папка Messages в папке Assets	Папка предназначена для хранения иерархии видов сообщений, доступных в ПС.

Продолжение таблицы 3.2

Имя файла/папки	Содержание
Файл Message.cs	Файл хранит в себе класс, который является базовым для всех видов сообщений.
Файл MyExtentions.cs	Файл содержащий статический класс для расширения разных типов.
Папка Database	Данная папка создана для взаимодействия приложения с базой данных.
Файл DB.cs	Данный файл содержит класс DB для настройки связи с БД. Данный класс работает с теми типами, которые реализовывают интерфейс IDBTable.
Папка DBObjects	Эта папка содержит классы или же адаптеры под те типы, которые будут взаимодействовать с БД. Все эти классы реализуют интерфейс IDBTable, который предоставляет методы и свойства для извлевлечения, добавления и изменения данных.
Папка Messages	Данная папка содержит классы, которые будут использоваться для того, что показать на View пользователю.
Файл LeftUser.cs	Содержит класс LeftUser, предназначенный для показа, какие люди в приложении есть, какое последнее сообщение от них, их аватар.
Файл Sticker.cs	Содержит класс Sticker. Он используется для того, чтобы хранить информацию о стикере и показывать его на View.
Папка Net	Данная папка содержит классы для взаимодействия пользователя и приложения, настройки его.
Файлы Client.cs и другие, начинающиеся на Client	Отвечают за работу с человеком, который пользуется компьютером, и за работу с БД для получения информацию о пользователе, который авторизируется.
Файлы Client.cs и другие, начинающиеся на Client	Отвечают за работу с человеком, который пользуется компьютером, и за работу с БД для получения информацию о пользователе, который авторизируется.

Продолжение таблицы 3.2

Имя файла	Содержание
Файл Settings.cs	Файл отвечает за настройку приложения. Он работает с файлом settings.json, в котором можно настроить начальную тему или строку соединения с базой данных.
Папка Objects	Данная папка содержит основые объект, с которыми будет работать пользователь.
Файл User.cs	Содержит класс User, который отвечает за пользователя подключенного к приложению.

Рассмотрим папку «View». Их подробная структура показана на рисунке 3.3.

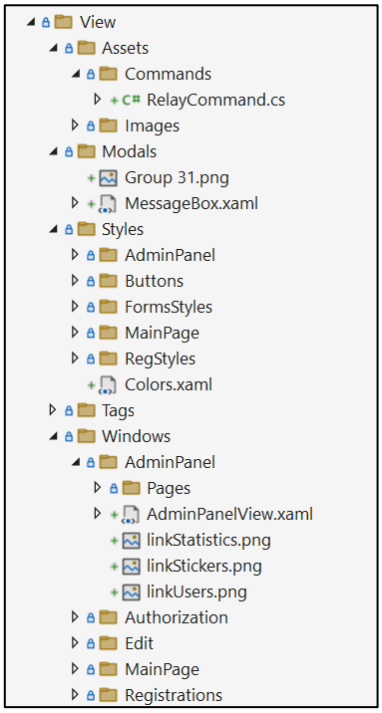


Рисунок 3.3 — Подробная структура папки «View»

Описание файлов и внутренних папок папки представлено в таблице 3.3.

Таблица 3.3 – Описание файлов и внутренних папок папки «View»

Имя файла	Содержание
Папка Assets	Папка содержит в себе комманду для взаимодействия View и ViewModel части приложения. Так же тут хранятся картинки, используемые в приложении.

Продолжение таблицы 3.3

Имя файла	Содержание
Папка Modals	В этой папке хранятся модальные окна. Одно из них это, оповещательное окно MessageBox.
Папка Styles	Папка, где хранятся все стили для каждого окна и каждой страницы.
Файл Colors.xaml	Словарь, в котором хранятся все цвета, которые используются в приложении.
Папка Tags	Папка, где хранятся пользовательские элементы.
Папка Window	Папка, в которой хранятся папки, в которых написана разметка для окон.
Папка AdminPanel	Окно админ панели. Внутренняя папка Pages хранит страницы для данного окна.
Папка Authorization	Хранит в себе окно авторизации.
Папка Edit	Хранит в себе окно редактирования.
Папка MainPage	Хранит в себе окно главной страницы для переписок и страницы для этого окна.
Папка Registration	Хранит в себе окно для регистрации.

Рассмотрим папку «ViewModel». Её подробная структура показана на рисунке 3.4.

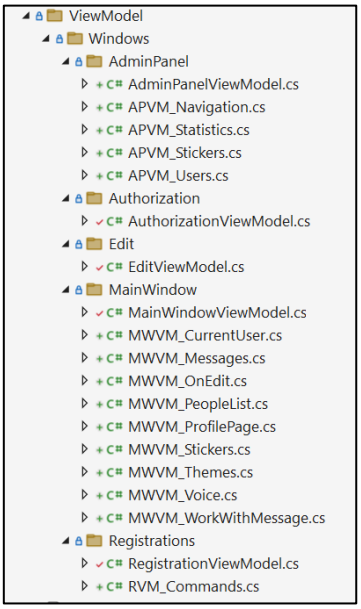


Рисунок 3.4 — Подробная структура папки «ViewModel»

Описание файлов и внутренних папок представлено в таблице 3.4.

Таблица 3.4 – Описание файлов и внутренних папок папки «ViewModel»

Имя файла	Содержание
Папка AdminPanel	Содержит класс AdminPanelViewModel. Данный класс описывает взаимодействия пользователя с графической частью приложения и бизнес логикой.
Папка Authorization	Содержит класс AuthorizationViewModel. Данный класс отвечает за функционал авторизации пользователя.
Папка Edit	Содержит класс EditViewModel. Данный класс отвечает за логику произведения редактирования профиля.
Папка MainWindow	Содержит класс MainWindowViewModel. Этот класс отвечают за все взаимодействия визуальной части главного окна приложения с бизнес логикой.
Папка Registrations	Содержит класс RegistrationViewModel данный класс отвечает за то, чтобы связать форму регистрации и бизнес логику.

В целом, описание структуры проекта позволяет лучше понимать, как устроено программное средство и какие компоненты в нем присутствуют.

3.2 Взаимоотношение между классами

Для визуализации взаимосвязей между классами используется диаграмма UML – графическое представление набора элементов, изображаемое чаще всего в виде связанного графа с вершинами (сущностями) и ребрами (отношениями).

Для представления внутренней структуры программы в виде классов и связей между ними используется диаграмма классов. Приложение спроектировано таким образом, что каждый класс выполняет свои функции и практически не зависит от других. Диаграмма классов представлена в приложении А.

3.3 Модель базы данных

Для реализации поставленной задачи была создана база данных X_Messenger. Для ее создания использовалась система управления реляционными базами данных MS SQL Server. База данных состоит из таблиц, представленных на рисунке 3.3. Скрипт для создания базы данных представлен в приложении В.

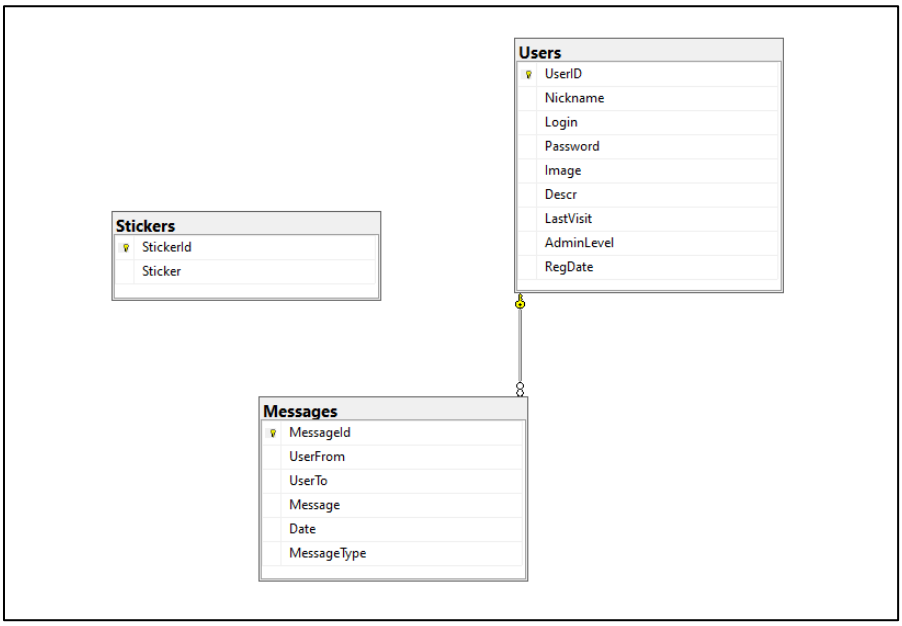


Рисунок 3.3 — База данных X_Messenger

На рисунке 3.4 проиллюстрирована структура таблицы «Users», которая содержит информацию о пользователях. В данной таблице есть поле «UserID», которое является первичным ключом. Оно отвечает за уникальность пользователя в приложении. Следующее поле — это «Nickname». Это поле отвечает за имя пользователя, его позывной. Следующее поле — это «Login». Оно отвечает за уникальное поле при регистрации или же авторизации. Следующее поле — это «Password». Оно отвечает за пароль от аккаунта. Еще одно поле — это «Image». Оно хранит изображение в байтах. Поле «Descr» отвечает за описание, краткую информацию о пользователе. Следующее поле — «LastVisit». Это время, когда пользователь был последний раз в сети. Поле «AdminLevel» отвечает за уровень доступности в приложении. Также есть поле «RegDate», которое отвечает за дату регистрации аккаунта.

	Column Name	Data Type	Allow Nulls
🔑	UserID	int	☐
	Nickname	nvarchar(50)	☐
	Login	nvarchar(30)	☐
	Password	nvarchar(256)	☐
	Image	varbinary(MAX)	☑
	Descr	nvarchar(300)	☑
	LastVisit	datetime	☐
	AdminLevel	int	☐
	RegDate	datetime	☐

Рисунок 3.4 — Структура таблицы «Users»

На рисунке 3.5 изображена структура таблицы «Messages». В данной таблице хранится информация об отправленных сообщениях. Первое поле — «MessageId». Уникальный идентификатор сообщения. Поле «UserFrom» отвечает за то, от кого

сообщение идет, а поле «UserTo» отвечает за то, кому придет сообщение. Поле «Message» хранит в себе байты сообщения. А вот поле «MessageType» определяет, чем являются данные байты: звуком, текстом или же картинкой. Поле «Date» — это дата и время, когда было отправленно сообщение.

Column Name	Data Type	Allow Nulls
MessageId	nvarchar(50)	☐
UserFrom	int	☐
UserTo	int	☐
Message	varbinary(MAX)	☐
Date	datetime	☐
MessageType	nvarchar(10)	☐

Рисунок 3.5 — Структура таблицы «Messages»

На рисунке 3.6 изображена структура таблицы «Stickers». Данная таблица для хранения стикеров. Одно из полей — это идентификатор стикера «StickerId». А Второе поле — это сама картинка стикера «Sticker».

Column Name	Data Type	Allow Nulls
StickerId	int	☐
Sticker	varbinary(MAX)	☐

Рисунок 3.6 — Структура таблицы «Stickers»

В целом, структура базы данных "X_Messenger" предоставляет необходимую основу для хранения и управления информацией о пользователях, сообщениях и стикерах в приложении "X-Messenger".

3.4 Проектирование архитектуры приложения

Для общего представления функционального назначения системы часто используется диаграмма использования. Она позволяет описать, какой функционал доступен каждой группе пользователей в разрабатываемой программной системе. В диаграмме использования выделяются два основных типа сущностей: варианты использования и актёры.

Актёры представляют различные группы пользователей, которые взаимодействуют с системой. Они могут быть представлены любыми сущностями, использующими систему. Варианты использования, с другой стороны, описывают функционал системы. Каждый вариант использования определяет набор действий, которые актёр может выполнить для взаимодействия с системой, а также действия, которые сама система выполняет в ответ.

Диаграмма использования представлена в приложении Б.

3.5 Проектирование последовательностей проекта

Для наглядного отображения взаимодействия между объектами системы в различные моменты времени в рамках одного сценария использования применяется диаграмма последовательностей в UML. Эта диаграмма позволяет иллюстрировать, как объекты системы взаимодействуют друг с другом для выполнения определенной функции, а также отображает порядок их взаимодействия в рамках конкретного случая использования.

Для отображения времени на диаграмме используется линия жизни объекта, которая представляет период его существования в системе и изображается вертикальной штриховой линией, направленной вниз. Объекты представлены прямоугольниками, а сообщения, которыми они обмениваются, обозначаются линиями с стрелками.

Диаграмма последовательностей позволяет наглядно представить взаимодействие между объектами и порядок выполнения действий во времени. Она является полезным инструментом для анализа и проектирования системы, а также может использоваться для коммуникации между разработчиками и заинтересованными сторонами.

Пример диаграммы последовательности представлен на рисунке 3.7.

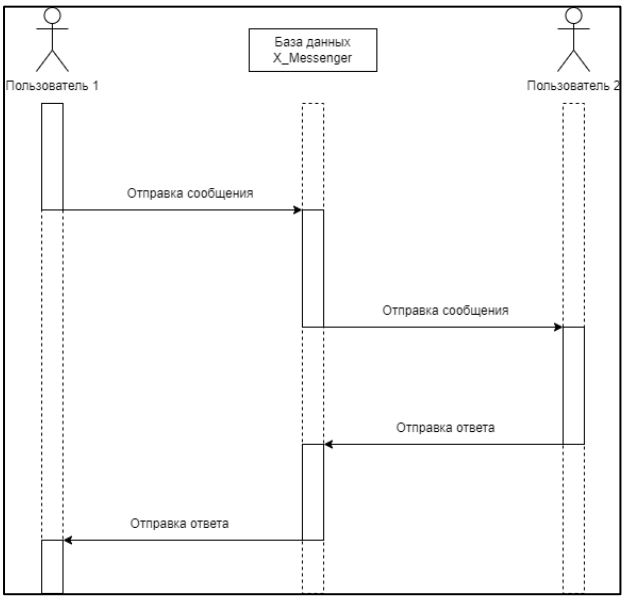


Рисунок 3.7 — Диаграмма последовательности

Пример диаграммы последовательности отображает процесс отправки сообщения одним пользователем другому пользователю и после чего ожидание получения ответа.

4 Реализация программного средства

4.1 Основные классы программного средства

Для выполнения технических задач программного средства «X-Messenger» должны быть реализованы следующие функции и соответствующие им классы и методы:

- авторизация;
- регистрация;
- редактирование профиля;
- отправка сообщений;
- панель администратора.

4.2 Авторизация

Графическая часть формы авторизации описана в классе Autorization. После нажатия кнопки «Войти» сработает команда «OnSubmitCommand, функционал которой описан в классе AuthorizationViewModel.

В классе AuthorizationViewModel команда OnSubmitCommand вызывает метод OnSubmitAsync, который асинхронно проверяет введенные пользователем данные и, если нужно, подготавливает список ошибок и по итогу проверки показывает его. Если данные корректны, вызывается асинхронная функция Authorize. Код метода Authorize представлен в листинге 4.1.

```
public async Task Authorize(User? user)
    {
        var res = await client.DoAuthorizationAsync(user);

        if (res) {
            using DB db = new();
            user = (await
db.GetDatasAsync<DBUser>(new(user))).First();

            Client.GetClient().CurrentUser = user;
            Client.NavigateTo<MainPageView>();
        }
        else
        {
            View.Modals.MessageBox.Show("Пользователя с такими
данными не существует");
        }
    }
```

Листинг 4.1 — Метод Authorize

Функция Authorize уже обращается к классу DB, то есть к самой базе данных. В базе данных вызывается процедура Authorization, в которой сразу проверяется,

есть ли вообще аккаунт с введенным логином и если он есть, то соответствует ли ему введенный пароль.

Если аккаунт есть, то процедура вернет ID данного пользователя, если аккаунт не найден по таким параметрам, возвращается -1. И в зависимости от ответа от БД будет в случае успешной авторизации перемещение на главную страницу под пользователем, ID которого вернула процедура, а в случае неуспешной авторизации будет выведена ошибка.

Листинги реализаций классов Autorization, AuthorizationViewModel, метода сравнения класса DB и процедуры Authorization в базе данных представлены в приложении В.

4.3 Регистрация

Для графической части процесса регистрации создан класс RegistrationView. В нем описана разметка для формы регистрации.

После нажатия на кнопку зарегистрироваться, срабатывает команда ClickOnSubmitCommand. Ее функциональность описана в классе RegistrationViewModel.

Функционал регистрации описывает метод ClickOnSubmitAsync, который асинхронно проверяет форму на корректность введенных данных и, если данные введены корректно, вызывает метод DoRegisterAsync, которая уже обращается к БД, вызывая там процедуру Registration, которая генерирует для нового пользователя собственный уникальный идентификатор, проверяет, чтобы логин не был занят и возвращает идентификатор нового пользователя. Описание процедуры Registration представлено в листинге 4.2.

```
ALTER procedure [dbo].[Registration] @NICKNAME nvarchar(50), @LOGIN
nvarchar(30), @PASS nvarchar(64)
as
begin
     begin try
          DECLARE @ID INT = (isnull((select max(Users.UserID) from
Users),1) + 1);
          insert into Users values (@ID, @NICKNAME, @LOGIN, @PASS,
(select Image from users where UserID = 0), NULL, GETDATE(), 0,
GETDATE())
          exec sendStartMessage @idTo = @ID, @name = @NICKNAME;
          return @ID;
     end try
     begin catch; return -1;
     end catch; end
```

Листинг 4.2 — Описание процедуры Registration

Листинг классов и всех методов представлен в приложении В.

4.4 Редактирование профиля

Графическая часть формы редактирования содержится в классе EditView. Для связи ее с бизнес логикой используется класс EditViewModel.

После заполнения нужных полей, пользователь должен нажать на кнопку «Редактировать» и после этого срабатывает команда OnSubmitCommand. Функционал которой реализован в методе OnSubmit. В данном методе идет проверка на корректность введенных данных. Если данные введены корректно, вызывается метод GetAnswerAsync. Он делает запрос на редактирование в БД. В базе данных отрабатывает процедура editUser. В зависимости от введенных данных, она будет возвращать разные значения, для примера, если введен логин, который уже занят, процедура вернет цифру «-1».

После возвращения значения процедурой метод GetAnswerAsync дает ответ пользователю об успешном или неуспешном изменении профиля.

Листинги реализаций классов и процедуры БД представлены в приложении В.

4.5 Отправка сообщений

Отправка сообщений начинается с того, что пользователь должен ввести текст в поле для сообщения, после чего нажать на кнопку «Отправить сообщение». Данные действия происходят на главной странице.

После нажатия кнопки отправки сообщения срабатывает команда SendMessageCommand. Функционал данной команды заключается в методе SendTextMessage. Данная функция формирует новое сообщение типа Message. И с помощью метода SendMessageAsync, асинхронно отправляет сообщение в БД. В БД отрабатывает процедура sendMessage, которая записывает данные в нужные таблицы. Ее листинг будет представлен в приложении В.

Листинги реализаций методов и процедур для процесса отправки сообщений представлены в приложении В.

4.6 Панель администратора

Графическая часть панели администратора представлена классом AdminPanelView. Администратор имеет доступ к трем страницам: странице, где видны все пользователи; странице статистики и страницы стикеров.

За связь с бизнес логикой отвечает класс AdminPanelViewModel. В нем описаны команды для удаления аккаунтов, добавления и удаления стикеров. Также данный класс отвечает за загрузку данных на видимую часть.

Листинги реализаций этих классов представлены в приложении В.

5 Тестирование, проверка работоспособности и анализ полученных результатов

5.1 Тестирование регистрации

При регистрации может быть попытка создать аккаунт с логином, который уже занят, поэтому было принято решение проверить это.

Предусловием будет то, что уже создан аккаунт с логином 123123123, но я буду пытать создать еще один аккаунт с таким логином. Результат данного теста представлен на рисунке 5.1.

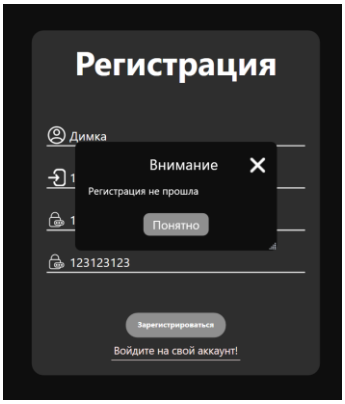


Рисунок 5.1 — Результат первого тестирования

Дальше бы я хотел попробовать ввести некорректные данные. Например, ввести пароль, состоящий из 3 символов. Результат данного тестирования представлен на рисунке 5.2.

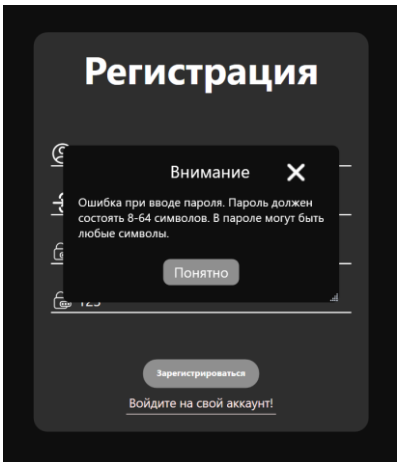


Рисунок 5.2 — Результат второго тестирования

Результаты тестирования регистрации говорят о том, что она работает корректно.

5.2 Тестирование отправки сообщения и работы с сообщениями

Самая главная задача мессенджера — это отправка. Поэтому протестировать данную функцию должно быть первой задачей.

Для начала нужно проверить, сработает ли запись голосового сообщения, а именно при нажатии на кнопку микрофона, она должна подсветиться, и после повторного нажатия, подсветка должна пропасть и должно отправить голосовое сообщение. Результат данного тестирования представлен на рисунке 5.3.

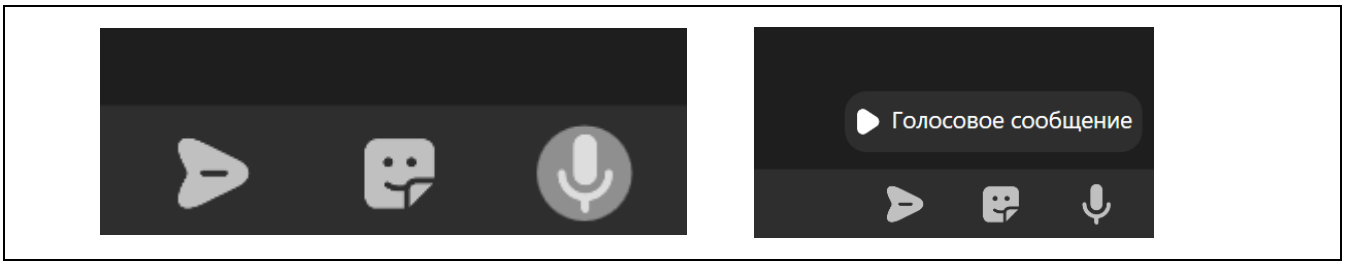


Рисунок 5.3 — После первого и второго нажатия на кнопку микрофона

Также пользователь может редактировать текстовые сообщения. Для этого ему нужно дважды нажать на сообщение, после чего должно появится минименю с кнопками удалить и редактировать. После нажатия на кнопку редактировать в поле ввода сообщения должен появится текст, написанный в сообщении. Для изменения нужно просто написать новый текст и отправить. Результат после двойного нажатия на сообщения представлено на рисунке 5.4. Результат после нажатия на кнопку редактирования представлен на рисунке 5.5. Результат после отправки измененного сообщения представлен на рисунке 5.6.

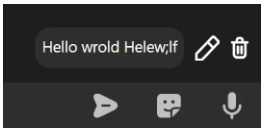


Рисунок 5.4

Рисунок 5.5

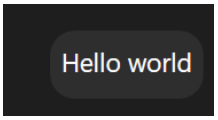


Рисунок 5.6

Возможность записывать голосовые сообщения облегчает возможность общения между людьми. Когда руки заняты или просто хочется высказаться кому-то, голосовые сообщения будут как некстати удобны.

Проверка будет заключаться в том, что я буду слушать большое голосовое сообщение и, когда я перейду в переписку с другим человек, оно должно остановиться проигрывать звук.

В результате данного тестирования был найден баг. Во время прослушивания сообщения, когда я перешел в переписку к другому человеку, сообщение продолжало проигрывания. Баг был исправлен.

5.3 Тестирование панели администратора

Администраторы должны следить за порядком в приложении. Чтобы пользователи не создавали аккаунты с непристойными именами или же неприемлемыми аватарами. Поэтому у администратора должна быть возможность удаления аккаунта пользователя.

Первое тестирование — это попытка удаления аккаунта другого пользователя. В разделе «Пользователи» в таблице состоящей из аккаунтов пользователей есть кнопку «Удалить аккаунт». После нажатие на нее, аккаунт пользователя и все переписки с ним должны удалиться.

По результату данного тестирования, видно, что аккаунт и все переписки с удаленным человеком были удалены. Результат представлен на рисунке 5.7.

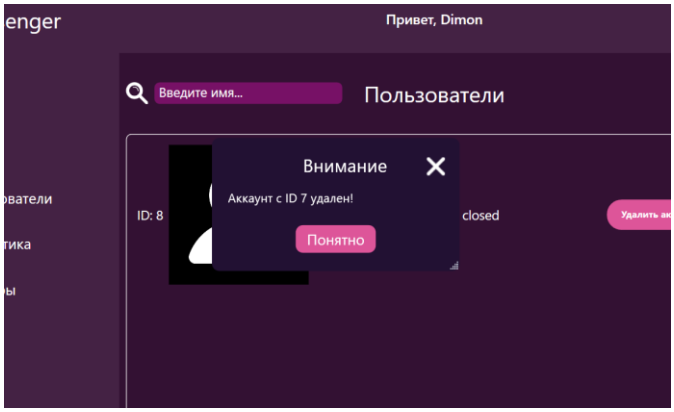


Рисунок 5.7 — Результат первой проверки панели администратора

Также если администратор по ошибке попытается удалить свой аккаунт. Должна быть предупреждение, что свой аккаунт он удалить не сможет. Результат данного тестирования показан на рисунке 5.8.

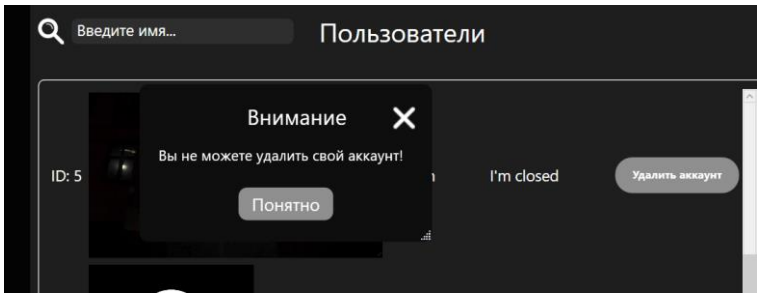


Рисунок 5.8 — Результат второго тестирования панели администратора

Администраторы обладают возможностями добавлять стикеры и удалять их. После нажатия на кнопку добавления стикера должно появиться окно для загрузки стикера. После выбора фотографии, она должна добавиться в приложение как стикер. Результат нажатия на кнопку добавить стикер представлен на рисунке 5.9.

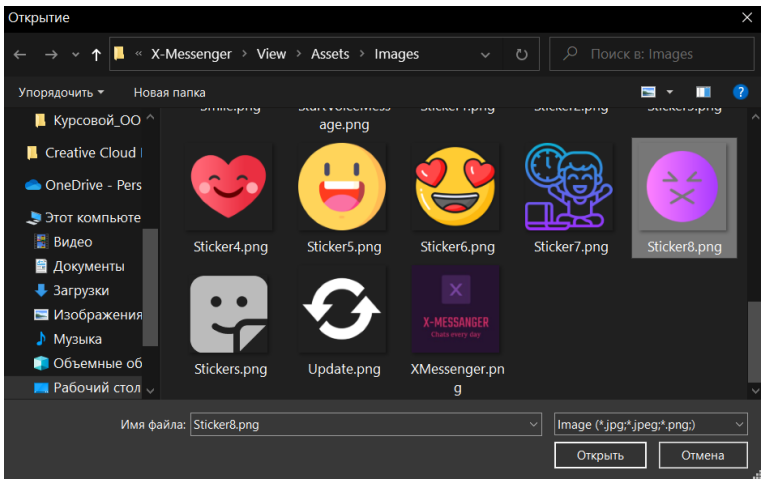


Рисунок 5.9 — Результат нажатия на кнопку «Добавить стикер»

После выбора картинки. Она добавилась в коллекцию стикеров и стала доступна в приложение. Результат показан на рисунке 5.10.

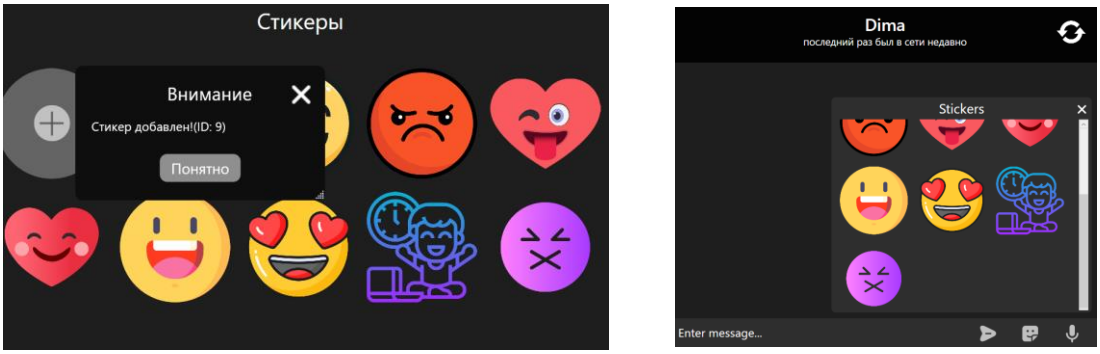


Рисунок 5.10 — Результат после добавления стикера

По результатам тестирования можно сказать, что панель администратора работает корректно.

6 Руководство по установке и использованию

При первом запуске приложения будет открыто окно авторизации, которое показано на рисунке 6.1.

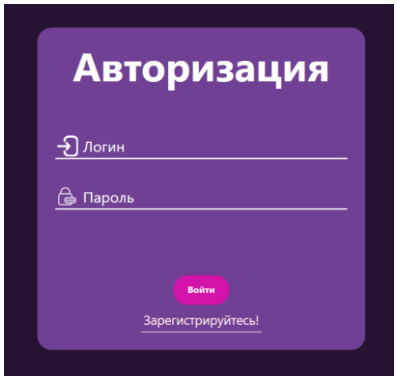


Рисунок 6.1 — Окно авторизации

Если у клиента не аккаунта, он может нажать на надпись «Зарегистрируйтесь!» для того, чтобы создать свой аккаунт.

После нажатия на кнопку регистрации откроется окно «Регистрация». Это окно представлено на рисунке 6.2. Поле имени должно быть заполнено либо русскими, либо латинскими буквами и первая буква должна быть большой. Логин должен состоять от 6 до 30 символов цифр или же букв. Пароль должен состоять от 8 до 64 разных символов. Поле с повторением пароля должно содержать такие же данные, как и поле с паролем.

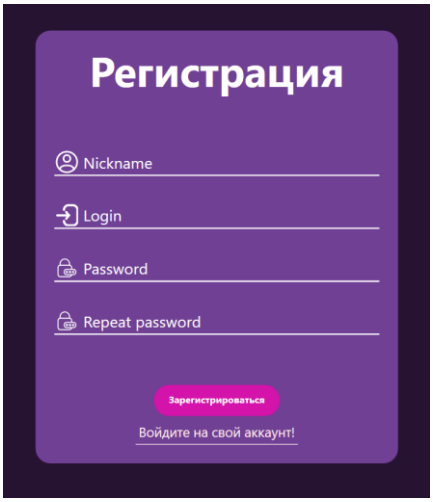


Рисунок 6.2 — Окно регистрации

После успешной регистрации пользователя перекинет на главную страницу. Нового пользователя встречает приветственное сообщение от администратора приложения, где указан минимальный напутствия на использование приложения «X-Messenger». На главной странице пользователю также будут доступны переписки со всеми другими пользователями. Для удобства разработан поиск

пользователей. Также тут есть кнопка, которая откроет меню с настройками приложения. Главная страница приложения представлена на рисунке 6.3.

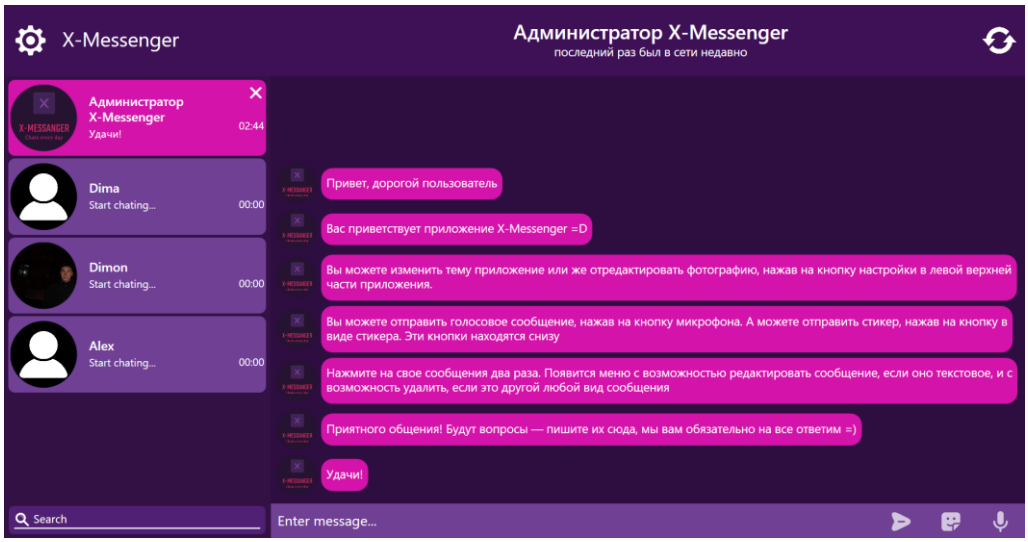


Рисунок 6.3 — Главная страница при первой загрузке

Для редактирования профиля, нужно открыть настройки и нажать на кнопку «Редактировать профиль». После нажатия на кнопку появится новое окно «Редактирование». Меню настроек показано на рисунке 6.4.

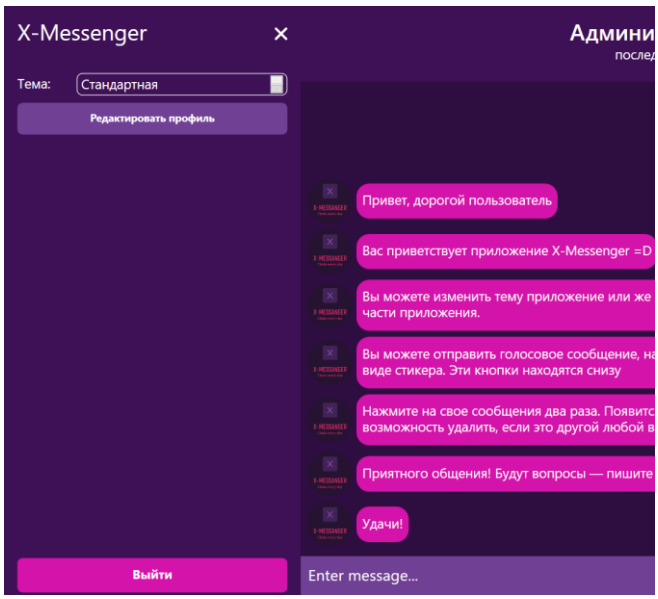


Рисунок 6.4 — Меню настроек

В меню редактирование вам предложено сменить фотографию пользователя, поменять имя, сменить описание, поменять логин, поменять пароль. Если пользователь не хочет ничего менять, то он может нажать на кнопку «Отмена». Но введенные новые данные будут все ровно проверены на корректность, будет проверено, чтобы новый логин не был занят. Окно регистрации представлено на рисунке 6.5.

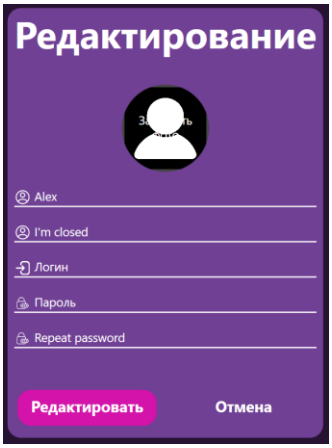


Рисунок 6.5 — Форма редактирования

Для смены темы приложения нужно выбрать нужную тему в меню настроек и нажать на нее и данная тема сохранится даже после выхода из приложения. Пример выбора новой темы представлен на рисунке 6.6.

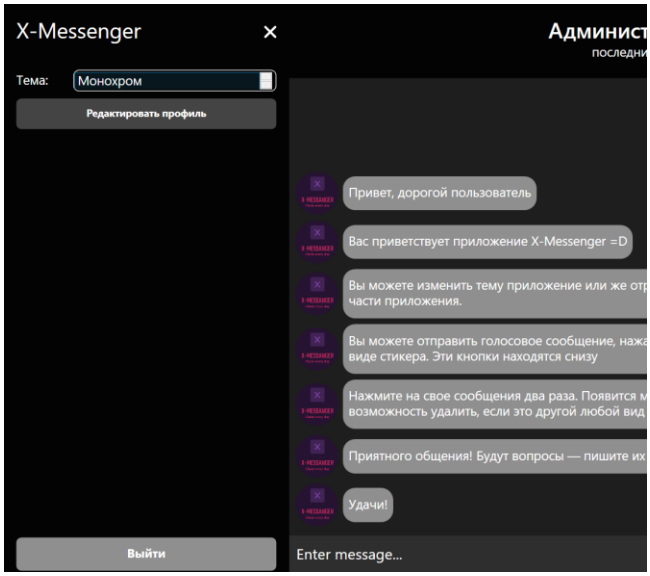


Рисунок 6.6 — Смена темы приложения

В целом, приложение «X-Messenger» обеспечивает удобный пользовательский опыт с возможностью настройки профиля и интерфейса приложения, повышая удобство и персонализацию для пользователей.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта было разработано программное средство "X-Messenger", предназначенное для обеспечения коммуникации между людьми находящимися в разных точках мира. Основной целью приложения является облегчение общения и расширение бизнес-сети пользователей.

Программное средство предоставляет возможности регистрации и авторизации пользователей, редактирования профилей, отправки голосовых, текстовых и стикерных сообщений, а также изображений. Пользователи имеют возможность персонализации приложения, просмотра информации о других пользователях, поиска пользователей, а также редактирования и удаления сообщений.

Для администраторов предусмотрены дополнительные функции, такие как авторизация, ответ пользователям, рассылка сообщений, просмотр статистики приложения, просмотр всех аккаунтов, удаление аккаунтов, добавление и удаление стикеров.

В процессе разработки были использованы принципы SOLID, которые способствуют созданию высококачественного и гибкого кода, а также методология Agile, обеспечивающая быструю адаптацию к изменяющимся требованиям и улучшение процесса разработки.

Для реализации приложения был выбран язык программирования C# и технология Windows Presentation Foundation (WPF), обеспечивающая разработку удобного и современного пользовательского интерфейса. В качестве базы данных была использована MS SQL Server.

В процессе разработки программного средства "X-Messenger" был активно применен паттерн проектирования MVVM (Model-View-ViewModel). MVVM является одним из наиболее популярных паттернов для разработки пользовательских интерфейсов в технологии WPF.

В процессе разработки программного средства "X-Messenger" ViewModel была активно использована для связи пользовательского интерфейса с бизнес-логикой и данными. Благодаря MVVM, разработка стала более структурированной и управляемой, а код стал более поддерживаемым и расширяемым.

Таким образом, применение паттерна MVVM в программном средстве "X-Messenger" обеспечило эффективное управление данными и логикой приложения, улучшенную архитектуру и обеспечило высокую отзывчивость и удобство использования пользовательского интерфейса.

В результате выполнения данного курсового проекта было создано функциональное и удобное программное средство "X-Messenger", которое позволяет пользователям легко общаться. Программа имеет потенциал для дальнейшего развития и расширения функциональности в соответствии с потребностями пользователей.

Список использованных источников

1. Microsoft Visual Studio [Электронный ресурс] – https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio – Дата доступа 23.04.2023

2. Полное руководство по языку программирования C# 7.0 и платформе .NET 4.7. Режим доступа: https://metanit.com/sharp/tutorial/ – Дата доступа: 23.04.2023

3. Пацей, Н.В. Курс лекций по языку программирования C# / Н. В. Пацей. – Минск: БГТУ, 2018. – 175 с.

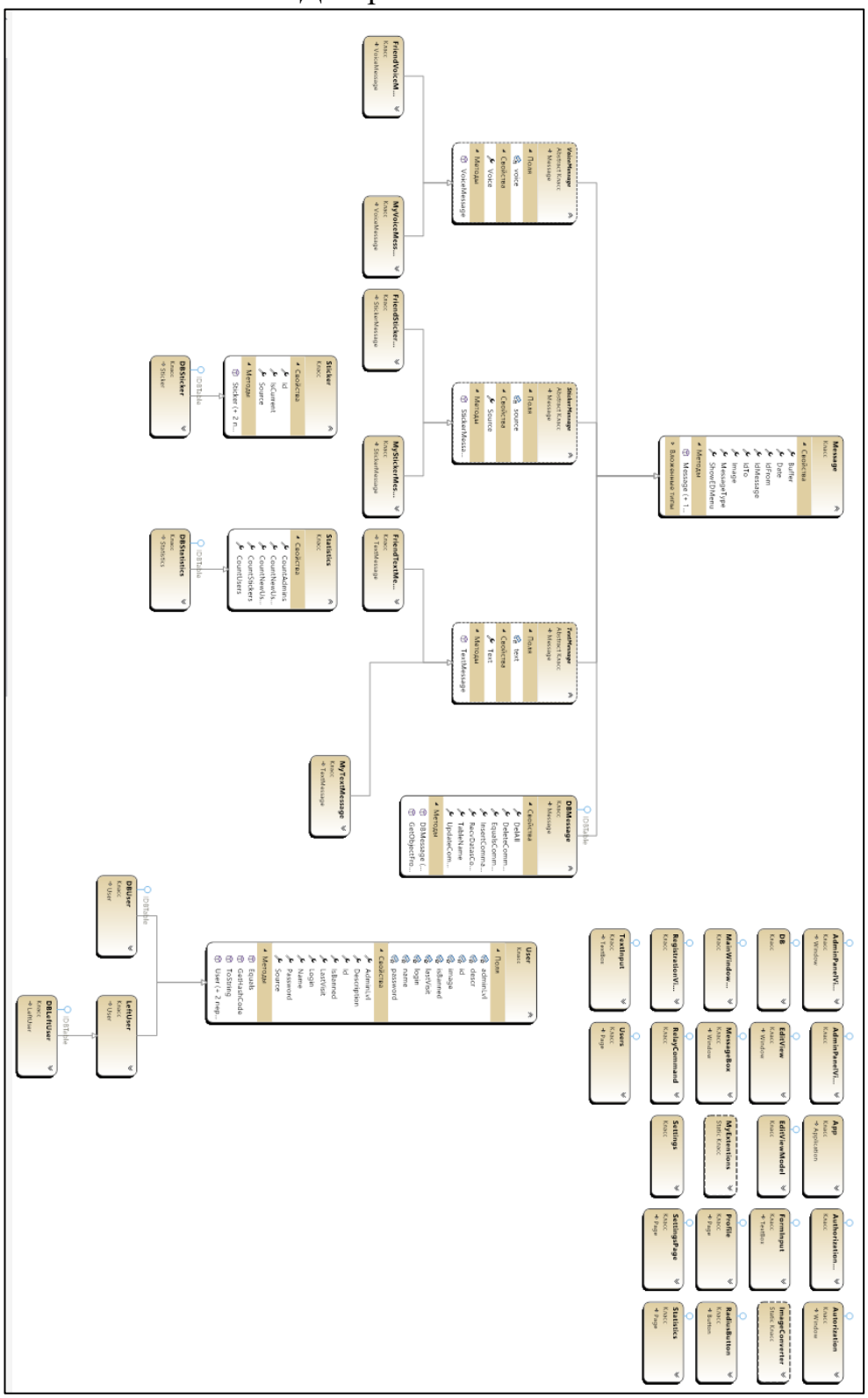
4. Руководство по WPF // [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/wpf/ – Дата доступа: 25.04.2023

5. Руководство по XAML // [Электронный ресурс]. – Режим доступа: https://www.tutorialspoint.com/xaml/index.htm– Дата доступа: 25.04.2023

6. Блинова, Е.А. Курс лекций по Базам данным / Е.А. Блинова. – Минск: БГТУ, 2019. – 175 с.

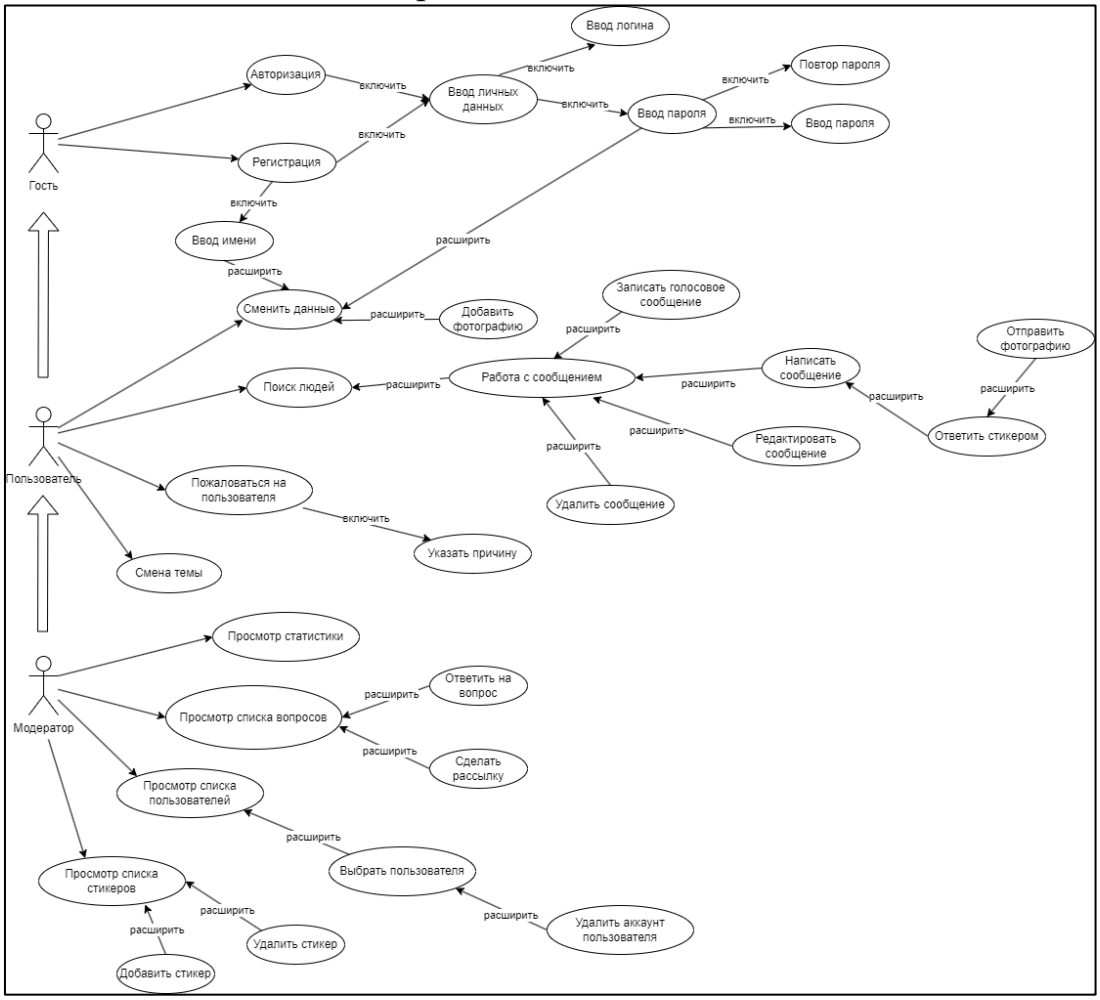
ПРИЛОЖЕНИЕ А

Диаграмма классов



ПРИЛОЖЕНИЕ Б

Диаграмма использования



ПРИЛОЖЕНИЕ В

Листинг формы авторизации в xaml

```
  <StackPanel x:Name="formInputs" Margin="0,0,0,57">

                    <tb:FormInput
                        Margin="0,0,0,35"
                        FontSize="21"

ImageSource="/View/Windows/Registrations/Login.png"
                        InputName="NameInput"
                        InputText="Логин"
                        Text="{Binding Login, Mode=TwoWay}" />

                    <tb:FormInput
                        Margin="0,0,0,35"
                        FontSize="21"

ImageSource="/View/Windows/Registrations/Password.png"
                        InputName="PasswordInput"
                        InputText="Пароль"
                        Text="{Binding Password, Mode=TwoWay}" />

                </StackPanel>

                <StackPanel>
                    <b:RadiusButton
                        Margin="0,0,0,8"
                        HorizontalAlignment="Center"
                        Command="{Binding OnSubmitCommand}"
                        Content="Войти"
                        Radius="20"
                        From="{ DynamicResource ColorThirdMain }"
                        To="{DynamicResource ColorThirdMain }" />

                    <Button
                        Name="LinkTo"
                        Command="{Binding OnLinkCommand}"
                        Content="Зарегистрируйтесь!"
                        Style="{StaticResource LinkButtonStyle}" />
</StackPanel>
```

Листинг функционала авторизации

```
using Lib.Assets.Checkers;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Input;
using X_Messenger.Model.Database;
using X_Messenger.Model.Database.DBObjects;
```

```
using X_Messenger.Model.Net;
using X_Messenger.Model.Objects;
using X_Messenger.View.Assets.Commands;
using X_Messenger.View.Windows.MainPage;
using X_Messenger.View.Windows.Registrations;

namespace X_Messenger.ViewModel.Windows.Authorisation;

internal partial class AuthorizationViewModel :
INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;
    private readonly FormDataChecker checker = new();
    private readonly Client client = Client.GetClient();

    public ICommand OnSubmitCommand { get; init; }
    public ICommand OnLinkCommand { get; init; }

    public AuthorizationViewModel()
    {
        OnSubmitCommand = new RelayCommand((o) => {
OnSubmitAsync(o); }, o => true);
        OnLinkCommand = new RelayCommand(OnLink, o => true);
    }

    public void OnPropertyChanged(string pName)
    {
        if (PropertyChanged is not null)
        {
            PropertyChanged(this, new
PropertyChangedEventArgs(pName));
        }
    }

    private string login;
    private string password;

    public string Login
    {
        get => login;
        set
        {
            login = value ?? string.Empty;
            OnPropertyChanged("Login");
        }
    }

    public string Password
    {
        get => password;
        set
        {
```

```
            password = value ?? string.Empty;
            OnPropertyChanged("Password");
        }
    }

    private readonly IList<string> errors = new List<string>();

    public async Task OnSubmitAsync(object param)
    {
        bool loginCorrect = checker.CheckLogin(Login);
        bool passCorrect = checker.CheckPassword(Password);

        errors.Clear();

        if (!loginCorrect)
        {
            errors.Add("Логин не корректен. Должно быть от 6 до 30
символов");
        }
        if (!passCorrect)
        {
            errors.Add("Пароль не корректен. Должно быть от 8 до
64 символов");
        }
        if (errors.Count > 0)
        {
            View.Modals.MessageBox.Show(string.Join("\n\n",
errors));
        }
        else
        {
            User user = new()
            {
                Login = Login.Trim(),
                Password = Password.Trim()
            };

            await Authorize(user);
        }
    }

    public async Task Authorize(User? user)
    {
        var res = await client.DoAuthorizationAsync(user);

        if (res)
        {
            using DB db = new();
            user = (await
db.GetDatasAsync<DBUser>(new(user))).First();

            Client.GetClient().CurrentUser = user;
```

```
            Client.NavigateTo<MainPageView>();
        }
        else
        {
            View.Modals.MessageBox.Show("Пользователя с такими
данными не существует");
        }
    }

    public void OnLink(object param)
    {
        Window reg = new RegistrationView();
        Client.NavigateTo<RegistrationView>();
    }
}
```

Листинг метода сравнения объектов в классе DB

```
public int EqualsObject<T>(T obj) where T : IDBTable
    {
        var sqlCommand = obj.EqualsCommand;
        sqlCommand.Connection = connection;

        SqlParameter returnParam =
sqlCommand.Parameters.Add("returnVal", SqlDbType.Int);
        returnParam.Direction = ParameterDirection.ReturnValue;

        sqlCommand.ExecuteNonQuery();

        return (int)returnParam.Value;
    }
```

Листинг процедуры для организации процесса авторизации в БД

```
ALTER procedure [dbo].[Authorization]
     @LOGIN nvarchar(30),
     @PASS nvarchar(64)
as
begin
     declare @id int = -1;
     declare users cursor local for select UserID from Users where
@LOGIN = [Login] and @PASS = [Password];

     open users
          fetch users into @id;

          if @@FETCH_STATUS < 0
               set @id = -1;
     close users

     return @id;
end
```

Листинг формы регистрации в XAML

```
<StackPanel x:Name="formInputs" Margin="0,0,0,67">

                    <tb:FormInput
                        Margin="0,0,0,35"
                        FontSize="21"

ImageSource="/View/Windows/Registrations/Nickname.png"
                        InputName="NameInput"
                        InputText="{ Binding Nickname, Mode=OneTime
}"
                        Text="{ Binding Nickname, Mode=TwoWay}"
                        TabIndex="0"/>

                    <tb:FormInput
                        Margin="0,0,0,35"
                        FontSize="21"

ImageSource="/View/Windows/Registrations/Login.png"
                        InputName="LoginInput"
                        InputText="{ Binding Login, Mode=OneTime }"
                        Text="{ Binding Login, Mode=TwoWay}"
                        TabIndex="1"
                        />

                    <tb:FormInput
                        Margin="0,0,0,35"
                        FontSize="21"

ImageSource="/View/Windows/Registrations/Password.png"
                        InputName="PasswordInput"
                        InputText="{ Binding Password, Mode=OneTime
}"
                        Text="{ Binding Password, Mode=TwoWay}"
                        TabIndex="2"/>

                    <tb:FormInput
                        FontSize="21"

ImageSource="/View/Windows/Registrations/Password.png"
                        InputName="RepassInput"
                        InputText="{ Binding RepeatPassword,
Mode=OneTime }"
                        Text="{ Binding RepeatPassword,
Mode=TwoWay}"
                        TabIndex="3"/>
                </StackPanel>

                <StackPanel>
                    <b:RadiusButton
                        Margin="0,0,0,8"
```

```
                        HorizontalAlignment="Center"
                        Content="Зарегистрироваться"
                        Radius="20"
                        From="{ DynamicResource ColorThirdMain }"
                        To="{ DynamicResource ColorThirdMain }"
                        Command="{ Binding ClickOnSubmitCommand }"
                        TabIndex="4"/>

                    <Button
                        Name="LinkTo"
                        Content="Войдите на свой аккаунт!"
                        Style="{StaticResource LinkButtonStyle}"
                        Command="{ Binding OnLinkCommand }"
                        />
                </StackPanel>
```

Листинг функционала класса RegistrationViewModel

```
private async Task ClickOnSubmitAsync(object? param)
    {
        bool isNameCorrect = checker.CheckName(Nickname);
        bool isLoginCorrect = checker.CheckLogin(Login);
        bool isPassCorrect = checker.CheckPassword(Password);
        bool isRepeatPasswordCorrect =
RepeatPassword.Equals(Password) && Password is not null;

        errors.Clear();

        if (!isNameCorrect)
        {
            errors.Add(
                """
                Ошибка при вводе имени. Первая буква имени должна
быть большой,
                а остальные маленькие. Имя может быть лишь на одном
языке(русский/английский
                """);
        }

        if (!isLoginCorrect)
        {
            errors.Add(
                """
                Ошибка при вводе логина. Логин должен состоять от 6
до 30 числовых и буквенных символов
                """
                );
        }

        if (!isPassCorrect)
        {
            errors.Add(
                """
```

```
                Ошибка при вводе пароля. Пароль должен состоять 8-64
символов. В пароле могут быть любые символы.
                """
                );
        }

        if (!isRepeatPasswordCorrect)
        {
            errors.Add(
                """
                Ошибка при подтверждении пароля. Поля пароля и
повторения пароля не совпадают
                """
                );
        }

        if (errors.Count() > 0)
        {
            View.Modals.MessageBox.Show(errors);
        }
        else
        {
            User user = new()
            {
                Name = Nickname?.Trim(),
                Password = Password?.Trim(),
                Login = Login?.Trim()
            };

            await DoRegisterAsync(user);
        }
    }

    private async Task DoRegisterAsync(User? user)
    {
        bool res = await client.DoRegistrationAsync(user);

        if (res)
        {
            client.AppSettings.Save();

            using DB db = new();
            user = (await
db.GetDatasAsync<DBUser>(new(user))).First();

            client.CurrentUser = user;

            Client.NavigateTo<MainPageView>();
        }
        else
        {
```

```
            View.Modals.MessageBox.Show(new[] { "Регистрация не
прошла" });
        }
    }
```

Листинг процесса вставки нового пользователя в классе DB

```
public int InsertObject<T>(T obj) where T : IDBTable
    {
        var sqlCommand = obj.InsertCommand;
        sqlCommand.Connection = connection;

        SqlParameter returnParam =
sqlCommand.Parameters.Add("returnVal", SqlDbType.Int);
        returnParam.Direction = ParameterDirection.ReturnValue;

        sqlCommand.ExecuteNonQuery();

        return (int)returnParam.Value;
    }
```

Листинг класса EditViewModel

```
using Lib.Assets.Checkers;
using Microsoft.Win32;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Windows;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using X_Messenger.Model.Assets.Converters;
using X_Messenger.Model.Database;
using X_Messenger.Model.Database.DBObjects;
using X_Messenger.Model.Net;
using X_Messenger.Model.Objects;
using X_Messenger.View.Assets.Commands;
using X_Messenger.View.Windows.MainPage;

namespace X_Messenger.ViewModel.Windows.Edit;

internal class EditViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler? PropertyChanged;

    private readonly FormDataChecker checker = new();

    public ICommand OnSubmitCommand { get; set; }
    public ICommand OnLoadImageCommand { get; set; }
    public ICommand OnCancelCommand { get; set; }

    public User CurrentUser { get; set; }
```

```
    private BitmapImage source;
    public BitmapImage Source
    {
        get => source;
        set
        {
            Application.Current.Dispatcher.Invoke(() =>
            {
                source = value;
                OnPropertyChanged(nameof(Source));
            });
        }
    }

    private string name;
    public string Name
    {
        get => name;
        set
        {
            name = value;
            OnPropertyChanged("Name");
        }
    }

    private string description;
    public string Description
    {
        get => description;
        set
        {
            description = value;
            OnPropertyChanged("Description");
        }
    }

    private string login = "Логин";
    public string Login
    {
        get => login;
        set
        {
            login = value;
            OnPropertyChanged("Login");
        }
    }

    private string password = "Пароль";
    public string Password
    {
        get => password;
```

```
        set
        {
            password = value;
            OnPropertyChanged("Password");
        }
    }

    private string rePass = "Повторите пароль";
    public string RePass
    {
        get => rePass;
        set
        {
            rePass = value;
            OnPropertyChanged("RePass");
        }
    }


    public EditViewModel()
    {
        Application.Current.Dispatcher.Invoke(() =>
        {
            OnSubmitCommand = new RelayCommand(OnSubmit, o => true);
            OnLoadImageCommand = new RelayCommand(OnLoadImage, o =>
true);
            OnCancelCommand = new RelayCommand(OnCancel, o => true);

            CurrentUser = Client.GetClient().CurrentUser;
            Name = CurrentUser.Name;
            Description = CurrentUser.Description;
            Source = CurrentUser.Source;
        });
    }

    public void OnLoadImage(object param)
    {
        Source = ImageConverter.OnLoadImage() ?? Source;
    }

    public void OnPropertyChanged(string propertyName)
    {
        if (PropertyChanged is not null)
        {
            PropertyChanged(this, new
PropertyChangedEventArgs(propertyName));
        }
    }

    public void OnCancel(object param)
    {
        Client.NavigateTo<MainPageView>();
```

```
    }

    public void OnSubmit(object param)
    {
        bool nameCorrect = checker.CheckName(Name) || Name ==
CurrentUser.Name;
        bool loginCorrect = checker.CheckLogin(Login) || Login ==
"Login";
        bool passCorrect = checker.CheckPassword(Password) ||
Password == "Password";
        bool repassCorrect = RePass.Equals(Password) || RePass ==
"Repeat password";

        Description = (string.IsNullOrWhiteSpace(Description) ||
Description == CurrentUser?.Description) ? CurrentUser?.Description
: Description;

        List<string> errors = new();

        if (!nameCorrect)
        {
            errors.Add("Имя не корректно!");
        }
        if (!loginCorrect)
        {
            errors.Add("Логин не корректен!");
        }
        if (!passCorrect)
        {
            errors.Add("Пароль не корректен!");
        }
        if (!repassCorrect)
        {
            errors.Add("Пароли не совпадают!");
        }
        if(Description.Length > 300)
        {
            errors.Add("Описание не может быть больше 300
символов!");
        }

        if (errors.Count > 0)
        {

            View.Modals.MessageBox.Show(string.Join("\n", errors));
        }
        else
        {
            User user = new User();
            user.Id = CurrentUser.Id;
            user.Name = Name == CurrentUser.Name ? null : Name;
            user.Login = Login == "Login" ? null : Login;
```

```
            user.Password = Password == "Password" ? null :
Password;
            user.Description = Description ==
CurrentUser.Description ? null : Description;
            user.Source = Source;

            GetAnswerAsync(user);
        }
    }

    public async void GetAnswerAsync(User? user)
    {
        if (user is null) return;

        var client = Client.GetClient();
        int code = await client.EditAsync(user);

        switch (code)
        {
            case 1:
                View.Modals.MessageBox.Show("Успешно!");

                using (DB db = new())
                    client.CurrentUser = (await
db.GetDatasAsync<DBUser>(new DBUser(user))).First();

                Client.NavigateTo<MainPageView>();
                break;
            case -1:
                View.Modals.MessageBox.Show("Не успешно. Логин
занят");
                break;
            case -2:
                View.Modals.MessageBox.Show("Не успешно. Данные не
загрузились");
                break;
        }
    }


}
```

Листинг части класса Client, отвечающая за редактирование профиля

```
internal partial class Client
{
    public async Task<int> EditAsync(User? user) => await
Task.Run(() => Edit(user));
    public int Edit(User? user)
    {
        if(user is null)
        {
```

```
            return -3;
        }

        using DB db = new();
        return db.UpdateObject<DBUser>(new(user));
    }
}
```

Листинг метода для измения данных в БД

```
public async Task<int> UpdateObjectAsync<T>(T obj) where T :
IDBTable
    {
        return await Task.Run(() => UpdateObject(obj));
    }

    public int UpdateObject<T>(T obj) where T : IDBTable
    {
        var sqlCommand = obj.UpdateCommand;
        sqlCommand.Connection = connection;

        SqlParameter returnParam =
sqlCommand.Parameters.Add("returnVal", SqlDbType.Int);
        returnParam.Direction = ParameterDirection.ReturnValue;

        sqlCommand.ExecuteNonQuery();

        return (int)returnParam.Value;
    }
```

Листинг подготовки SQL команды для изменения записи в БД

```
public SqlCommand UpdateCommand
    {
        get
        {
            SqlCommand command = new("editUser");
            command.CommandType =
System.Data.CommandType.StoredProcedure;

            command.Parameters.AddWithValue("@id", id);
            command.Parameters.AddWithValue("@name", name);
            command.Parameters.AddWithValue("@login", login ??
null);
            command.Parameters.AddWithValue("@pass", password ??
null);
            command.Parameters.AddWithValue("@image", image is null
? null : ImageConverter.ImageToBytes(image));
            command.Parameters.AddWithValue("@descr", descr ??
null);

            return command;
        }
    }
```

Листинг процедуры editUser в базе данных

```
ALTER procedure [dbo].[editUser]
     @id int,
     @name nvarchar(50) = null,
     @login nvarchar(30) = null,
     @pass nvarchar(64) = null,
     @image varbinary(max) = null,
     @descr nvarchar(300) = null
as
begin
     begin tran

          select * from users where UserID != @id and users.Login =
@login

          if @@ROWCOUNT > 0
          begin
               rollback tran;
               return -1;
          end

          if @name is not null
          begin
               update users set Nickname = @name where users.UserID
= @id;


          end

          begin try

               if @login is not null
                    update users set Login = @login where
users.UserID = @id;
               if @pass is not null
                    update users set Password = @pass where
users.UserID = @id;
               if @image is not null
                    update users set Image = @image where
users.UserID = @id;
               if @descr is not null
                    update users set Descr = @descr where
users.UserID = @id;

          end try
          begin catch

               rollback tran;
               return -2;

          end catch
     commit tran
```

```
     return 1;
end
```

Листинг методов для отправки сообщения

```
public void SendTextMessage(object param)
    {
        if (!string.IsNullOrWhiteSpace(SendedMessage) &&
!Edit(SendedMessage))
        {
            Message msg = new();

            if(SendedMessage.Length > 200)
            {
                SendedMessage = SendedMessage.Substring(0, 200);
                View.Modals.MessageBox.Show("Сообщение обрезано.
Максимальная длина сообщения 200 символов");
            }

            msg.IdFrom = Client.GetClient().CurrentUser?.Id.Value ??
0;
            msg.IdTo = CurrentUser?.Id.Value ?? 0;
            msg.Buffer = Encoding.UTF8.GetBytes(SendedMessage);
            msg.MessageType = Message.TypeOfMessage.Text;

            SendMessageAsync(msg);

            msg = new MyTextMessage(msg);
            AddToMessageList(msg, false);
        }

        SendedMessage = "Enter message...";
        OnPropertyChanged("SendedMessage");
    }

    public void SendMessage(Message msg)
    {
        using DB db = new();
        db.InsertObject(new DBMessage(msg));
    }

    public async Task SendMessageAsync(Message msg) => await
Task.Run(() => SendMessage(msg));
```

Листинг реализации класса DBMessage

```
internal class DBMessage : Message, IDBTable
{
    public string TableName => throw new NotImplementedException();
    public bool DelAll { get; set; } = true;

    public SqlCommand InsertCommand
    {
```

```
        get
        {
            SqlCommand command = new("sendMessage");
            command.CommandType =
System.Data.CommandType.StoredProcedure;

            command.Parameters.AddWithValue("@messageId",
this.IdMessage);
            command.Parameters.AddWithValue("@idFrom", this.IdFrom);
            command.Parameters.AddWithValue("@idTo", this.IdTo);
            command.Parameters.AddWithValue("@buffer", this.Buffer);
            command.Parameters.AddWithValue("@type",
this.MessageType switch
            {
                TypeOfMessage.Text => "TEXT",
                TypeOfMessage.Sticker => "STICKER",
                TypeOfMessage.Voice => "VOICE"
            });

            return command;
        }
    }

    public SqlCommand EqualsCommand => throw new
NotImplementedException();

    public SqlCommand RecvDatasCommand => throw new
NotImplementedException();

    public SqlCommand UpdateCommand
    {
        get
        {
            SqlCommand command = new("editMsg");
            command.CommandType =
System.Data.CommandType.StoredProcedure;

            command.Parameters.AddWithValue("@id", this.IdMessage);
            command.Parameters.AddWithValue("@buffer", this.Buffer);

            return command;
        }
    }

    public SqlCommand DeleteCommand
    {
        get
        {
            if (DelAll)
            {
                SqlCommand command = new("deleteAllMessages");
```

```
                command.CommandType =
System.Data.CommandType.StoredProcedure;
                command.Parameters.AddWithValue("idFrom",
this.IdFrom);
                command.Parameters.AddWithValue("idTo", this.IdTo);
                return command;
            }
            else
            {
                SqlCommand command = new("delMsg");

                command.CommandType =
System.Data.CommandType.StoredProcedure;
                command.Parameters.AddWithValue("@id",
this.IdMessage);

                return command;
            }
        }
    }

    public object GetObjectFrom(SqlDataReader set)
    {
        Message msg = new();

        msg.IdMessage = new(set.GetString(0));
        msg.IdFrom = set.GetInt32(1);
        msg.IdTo = set.GetInt32(2);
        msg.Buffer = set.GetSqlBinary(3).Value;

        string msgType = set.GetString(4);

        msg.MessageType = msgType switch
        {
            "STICKER" => Message.TypeOfMessage.Sticker,
            "VOICE" => Message.TypeOfMessage.Voice,
            "TEXT" => Message.TypeOfMessage.Text,
            _ => Message.TypeOfMessage.Text
        };

        msg.Date = set.GetDateTime(5);

        var image = set.GetValue(6) is DBNull ? null :
set.GetSqlBinary(6).Value;

        if (image is null)
        {
            Application.Current.Dispatcher.Invoke(() => msg.Image =
null);
        }
        else
        {
```

```
            Application.Current.Dispatcher.Invoke(() => msg.Image =
ImageConverter.ToImageSource(image));
        }

        return msg;
    }

    public DBMessage()
    {

    }

    public DBMessage(Message msg)
    {
        this.IdMessage = msg.IdMessage;
        this.IdFrom = msg.IdFrom;
        this.IdTo = msg.IdTo;
        this.Buffer = msg.Buffer;
        this.Date = msg.Date;
        this.MessageType = msg.MessageType;
    }
}
```

Листинг процедуры БД sendMessage

```
ALTER procedure [dbo].[sendMessage]
     @messageId nvarchar(50),
     @idFrom int,
     @idTo int,
     @buffer varbinary(max),
     @type nvarchar(10)
as
begin
     if @idFrom = @idTo and @idFrom = 1
     begin
          declare users cursor local for select UserID from users
where UserID > 0;
          declare @id int;
          open users

               fetch users into @id;

               while @@FETCH_STATUS = 0
               begin
                    insert into Messages values((SELECT
SUBSTRING(CONVERT(varchar(255), NEWID()), 0, 50) AS RandomString),
@idFrom, @id, @buffer, GETDATE(), @type);
                    fetch users into @id;
               end
          close users
     end
     else
```

```
          insert into Messages values(@messageId, @idFrom, @idTo,
@buffer, GETDATE(), @type);
end
```

Листинг части класса AdminPanelViewModel, отвечающая за инициализацию команд

```
internal partial class AdminPanelViewModel : INotifyPropertyChanged
{
    private User currentAdmin;

    public event PropertyChangedEventHandler? PropertyChanged;
    public User CurrentAdmin
    {
        get => currentAdmin;
        set
        {
            currentAdmin = value;
            OnPropertyChanged("CurrentAdmin");
        }
    }

    public AdminPanelViewModel(Frame navigator)
    {
        CurrentAdmin = Client.GetClient().CurrentUser;
        this.navigator = navigator;
        ToUsers(new object());

        ToUsersCommand = new RelayCommand(ToUsers, o => true);
        ToStatisticsCommand = new RelayCommand(ToStatistics, o =>
true);
        ToStickersCommand = new RelayCommand(ToStickers, o => true);

        Users = new ObservableCollection<User>();

        OnSearchCommand = new RelayCommand(OnSearchAsync, o =>
true);
        OnRemoveAccountCommand = new
RelayCommand(OnRemoveAccountAsync, o => true);
        OnStickerCommand = new RelayCommand(OnSticker, o => true);
        OnRemoveStickerCommand = new
RelayCommand(OnRemoveStickerAsync, o => true);
    }



    private void OnPropertyChanged(string pName)
    {
        if (PropertyChanged is not null)
        {
            PropertyChanged(this, new
PropertyChangedEventArgs(pName));
        }
```

```
    }
}
```

Листинг части класса AdminPanelViewModel, отвечающая за навигацию

```
internal partial class AdminPanelViewModel
{
    private readonly Frame navigator;

    public ICommand ToUsersCommand { get; private init; }
    public ICommand ToStatisticsCommand { get; private init; }
    public ICommand ToStickersCommand { get; private init; }

    private void ToUsers(object param)
    {
        Navigate(new Users(this));
        GetUsersAsync();
    }
    private void ToStatistics(object param)
    {
        Navigate(new Statistics(this));
        GetStatsAsync();
    }
    private void ToStickers(object param)
    {
        Navigate(new Stickers(this));
        GetStickersAsync();
    }

    private void Navigate(Page page)
    {
        navigator.Navigate(page);
    }
}
```

Листинг части класса AdminPanelViewModel, отвечающая за статистику

```
internal partial class AdminPanelViewModel
{
    private Model.Objects.Statistics stats;

    public Model.Objects.Statistics Stats
    {
        get => stats;
        set
        {
            stats = value;
            OnPropertyChanged("Stats");
        }
    }
```

```
    public async void GetStatsAsync()
    {
        using DB db = new DB();
        var stats = await db.GetDatasAsync(new DBStatistics());

        await Application.Current.Dispatcher.InvokeAsync(new
Action(() =>
        {
            Stats = stats.First();
        }));
    }
}
```

Листинг части класса AdminPanelViewModel, отвечающая за страницу пользователей

```
internal partial class AdminPanelViewModel
{
    public ICommand OnSearchCommand { get; private init; }
    public ICommand OnRemoveAccountCommand { get; private init; }

    private ObservableCollection<User> users;
    private string searchText = "Введите имя...";

    public ObservableCollection<User> Users
    {
        get => users;
        set
        {
            users = value;
            OnPropertyChanged("Users");
        }
    }
    public string SearchText
    {
        get => searchText;
        set
        {
            searchText = value;
            OnPropertyChanged("SearchText");
        }
    }


    public async void GetUsersAsync()
    {
        await Task.Run(() =>
        {
            using DB db = new();
            DBUser dbUser = new();
            User user;
```

```
            var set = db.GetDataSetByQuery("select * from
getUsers()");

            Application.Current.Dispatcher.Invoke(() =>
            {
                Users.Clear();
            });

            while (set.Read())
            {
                user = (User)dbUser.GetObjectFrom(set);

                Application.Current.Dispatcher.Invoke(() =>
                {
                    Users.Add(user);
                });
            }
        });
    }
    public async void OnSearchAsync(object param) => await
Task.Run(() => OnSearch(param));
    public void OnSearch(object param)
    {
        var sortedList = Users.OrderBy(el =>
el.Name.LevenshteinDistance(SearchText));

        Application.Current.Dispatcher.Invoke(() =>
        {
            Users = new(sortedList);
        });
    }
    public async void OnRemoveAccountAsync(object param)
    {
        User? user = param as User;

        if(user.Id == CurrentAdmin.Id)
        {
            View.Modals.MessageBox.Show("Вы не можете удалить свой
аккаунт!");
            return;
        }

        using DB db = new();
        await db.DeleteObjectAsync(new DBUser(user));

        await Application.Current.Dispatcher.BeginInvoke(() =>
        {
            Users.Remove(user);
        });

        View.Modals.MessageBox.Show($"Аккаунт с ID {user.Id}
удален!");
```

```
    }
}
```

Листинг части AdminPanelViewModel, отвечающая за страницу стикеров

```
internal partial class AdminPanelViewModel
{
    private ObservableCollection<Sticker> stickers = new();
    private TextMessage editingMsg = null;
    private DateTime last = DateTime.Now;
    public ICommand OnStickerCommand { get; private init; }
    public ICommand OnRemoveStickerCommand { get; private init; }
    public ObservableCollection<Sticker> Stickers
    {
        get => stickers;
        set
        {
            stickers = value;
            OnPropertyChanged("Stickers");
        }
    }
    private async void GetStickersAsync()
    {
        Stickers.Clear();
        using DB db = new();
        var set = await db.GetDataSetFromAsync<DBSticker>();
        DBSticker dbSticker = new();
        Sticker sticker;
        while (await set.ReadAsync())
        {
            sticker = (Sticker)dbSticker.GetObjectFrom(set);

            await
System.Windows.Application.Current.Dispatcher.InvokeAsync(() =>
            {
                Stickers.Add(sticker);
            });
        }
    }
    private void OnSticker(object param)
    {
        Sticker sticker = param as Sticker;

        if (sticker is null) return;

        if(sticker.Id == 0)
        {
            AddSticker();
        }
        else
        {
            var dur = DateTime.Now - last;
```

```
            last = DateTime.Now;
            if (dur.TotalMilliseconds > 500)
            {
                return;
            }

            sticker.IsCurrent = !sticker.IsCurrent;
            Stickers = new(Stickers.Select(el =>
            {
                if(sticker.Id != el.Id)
                {
                    el.IsCurrent = false;
                }

                return el;
            }));
        }
    }

    private async void AddSticker()
    {
        Sticker sticker = new();
        sticker.Source = ImageConverter.OnLoadImage();

        if(sticker.Source is null)
        {
            return;
        }

        DBSticker dbSticker = new(sticker);
        sticker.Id = await AddStickerToDBAsync(dbSticker);

        await Application.Current.Dispatcher.InvokeAsync(() =>
        {
            Stickers.Add(sticker);
        });

        View.Modals.MessageBox.Show($"Стикер добавлен!(ID:
{sticker.Id})");
    }
    private async Task<int> AddStickerToDBAsync(DBSticker sticker)
    {
        using DB db = new();
        return await Task.Run(() => db.InsertObject(sticker));
    }
    private async void OnRemoveStickerAsync(object param)
    {
        Sticker? sticker = param as Sticker;

        if (sticker is null) return;

        DBSticker dbSticker = new(sticker);
```

```
        using DB db = new();
        await db.DeleteObjectAsync(dbSticker);
        await Application.Current.Dispatcher.InvokeAsync(() =>
        {
            Stickers.Remove(sticker);
        });

        View.Modals.MessageBox.Show($"Стикер под ID {sticker.Id}
удален!");
    }
}
```

Листинг скрипта создания базы данных и таблиц для нее

```
use X_Messenger;

create database X_Messenger;

create table Users(
     UserID int primary key,
     Nickname nvarchar(50),
     [Login] nvarchar(30) unique,
     [Password] nvarchar(64),
     [Image] varbinary(MAX) null,
     [Descr] nvarchar(300) null,
     LastVisit datetime,
     AdminLevel int,
     RegDate datetime
)

create table Messages
( UserFrom int,UserTo int,[Message] varbinary(max),[Date] date,
constraint UserFrom_FK foreign key(UserFrom) references
Users(UserID),
     constraint UserTo_FK foreign key(UserTo) references
Users(UserID)
)

create table Stickers
(
     StickerId int primary key,
     Sticker varbinary(max)
)

CREATE INDEX IX_Messages_Date ON Messages (Date ASC);
CREATE INDEX idx_Messages_UserFrom ON Messages(UserFrom);
CREATE INDEX idx_Messages_UserTo ON Messages(UserTo);
CREATE INDEX idx_Users_UserID ON Users (UserID);
CREATE INDEX idx_Users_Login ON Users (Login);
```