Частное учреждение образования
«Колледж бизнеса и права»

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕССЕНДЖЕРА «NEAR MESSAGE»

по учебному предмету «Конструирование программ
и языки программирования»

КП Т.096020.401

Руководитель проекта (В.Ю.Михалевич)

Обучающийся (В.А.Лукашонок)

2023

СОДЕРЖАНИЕ

Введение

Мессенджер – это программа, мобильное приложение или веб-сервис для мгновенного обмена сообщениями.

Нужно сказать, что понятие мессенджера уже давно не связывают только с обменом текстовыми сообщениями. Современные мессенджеры уже стали полноценными коммуникационными центрами, которые помимо обмена сообщениями реализуют голосовую и видеосвязь, обмен файлами, веб-конференции.

С помощью мессенджеров бизнес отправляет рассылки, создаёт каналы для привлечения клиентов или распространяет рекламу. Бизнес использует мессенджеры, чтобы распространять контент, общаться с покупателями, уведомлять о заказах и доставке, рассылать информацию об акциях и скидках, привлекать клиентов с помощью розыгрышей и викторин, продавать товары и услуги.

Целью курсового проекта на тему «Программная реализация мессенджера «Near Message» является разработка программы, с помощью которой пользователи могли бы вести переписку текстовыми сообщениями и смайликами. Для того, чтобы пользователь мог вести переписку, ему надо завести аккаунт. Для этого необходимо зарегистрироваться, указав уникальное имя пользователя и пароль. После успешной регистрации, пользователям будут доступны функции ведения переписки, поиска, создания групп изменения фотографии профиля, информации о себе и переключение темы приложения.

Решение поставленных задач отражено в пояснительной записке, которая состоит из четырех разделов и содержит необходимую и достаточную информацию по использованию данного программного средства.

В первом разделе «Описание задачи» рассматривается сущность и актуальность поставленной задачи, описание существующих аналогов, описание основных функций для разработки программного средства.

Второй раздел «Проектирование системы» содержит проектирование модели на основе выбранных диаграмм, требования к аппаратным и операционным ресурсам, средства защиты, организация данных и описание концептуального прототипа.

Третий раздел «Описание реализации программного средства» включает описание избранной технологии для реализации программы и избранные инструменты разработки, порядок аутентификации и авторизации пользователей, физическая организация данных, способы реализации функции пользователя, формат входных и выходных данных, функциональное тестирование, описание справочной системы.

Четвертый раздел «Применение» содержит информацию, необходимую в процессе эксплуатации программного средства: его назначение и условия применения.

В заключении описывается выполнение поставленной задачи, степень соответствия проектных решений задания, причины несоответствия, если таковые имеются.

В приложении А представлен текст программных модулей, в приложении Б – формы выходных документов, в приложении В – результаты работы программы.

Графическая часть представлена диаграммами вариантов использования, классов, деятельности.

1 Описание задачи

1.1 Анализ предметной области

Требуется разработать программное средство для переписки пользователей – мессенджер «Near Message».

Предметной областью решаемой задачи является мессенджер.

Мессенджер — это программное средство, которое предоставляет возможность обмена сообщениями между пользователями. Обычно мессенджеры представляют собой приложения или программы, которые устанавливаются на устройства пользователей и позволяют отправлять и принимать текстовые, голосовые, видео или другие форматы сообщений.

Однако, мессенджер не ограничивается только самим приложением. За каждым мессенджером обычно стоит сеть обмена сообщениями, которая обеспечивает передачу сообщений между пользователями. Эта сеть может быть внутри компании, где мессенджер используется для внутренней коммуникации, или глобальной сетью, такой как Jabber, которая предоставляет возможность общения между пользователями в разных частях мира. Таким образом, мессенджер включает в себя как само программное средство, так и связанную с ним сеть обмена сообщениями.

К основным функциям мессенджеров относятся:

−Обмен текстовыми сообщениями: Мессенджеры позволяют пользователям обмениваться текстовыми сообщениями, подобно простому отправлению СМС;

−Прикрепление различных файлов: Пользователи могут прикреплять к сообщениям смайлики, фотографии, картинки, документы, видео и другие файлы, похожие на электронную почту, но работающие в режиме онлайн;

−Создание групповых чатов: Мессенджеры позволяют создавать группы пользователей для общения по интересам, работе или учебе. Это способствует коллективному общению и дистанционному обучению;

−Голосовые сообщения и набор голосом: Некоторые мессенджеры позволяют записывать и отправлять голосовые сообщения или использовать голосовой набор, что удобно для тех, кто предпочитает говорить, а не писать;

−Звонки на мобильные и городские телефоны: В некоторых мессенджерах есть функция совершения звонков на мобильные и городские телефоны за дополнительную плату.

−Непрерывное развитие и новые функции: Разработчики мессенджеров постоянно работают над улучшением своих продуктов. Новые функции могут включать облачные серверы, видеоконференции и другие инновации.

Мессенджеры также стремятся предоставить возможность архивирования чатов, чтобы пользователи могли восстановить переписку и переговоры в случае необходимости, например, при смене номера телефона.

Мессенджер состоит из двух частей: клиента и сервера. Пользователи работают с клиентским приложением, где реализованы все основные функции.

Сервер обрабатывает запросы от клиентов и содержит базу данных с информацией о пользователях, сообщениях, запросах и друзьях. Для оптимизации работы сервера каждый клиентское программное средство также имеет локальную базу данных для хранения некоторых данных и избежания повторных запросов к серверу. Например, сообщения между пользователями могут быть сохранены в локальной базе данных клиента.

1.2 Постановка задачи

Исходя из анализа предметной области можно выделить следующие задачи:

- ведение локального хранилища сообщений, данных о пользователи на клиенте;
- ведение базы данных на сервере, в которой будет храниться вся информация;
- ведение хранилища на сервере, в которой будет храниться сообщения и данные о пользователях;
- авторизация пользователей;
- регистрация пользователей;
- переписка пользователей, с использованием текстовых сообщений и файлов;
- поиск других пользователей.

Существует довольно много аналогов для данного приложения, такие как Телеграмм, Вайбер, ВКонтакте и другие. Разрабатываемый мессенджер направлен только на пользователей Windows 10. Программное средство сильно упрощено и не имеет больших массивов данных.

2 Проектирование системы

2.1 Требования к приложению

Разрабатываемое программное средство должно иметь понятный и удобный в использовании интерфейс, чтобы взаимодействие между программой и пользователем было максимально упрощено.

Согласно общим требованиям, графический интерфейс разрабатываемого программного средства должен:

- ориентироваться на пользователя, который общается с программой на внешнем уровне взаимодействия;
- сохранять стандартизированное назначение и местоположение на экране графических объектов, работающих в среде Windows.

Интерфейс программного средства будет разрабатываться с учетом общих требований к пользовательскому интерфейсу. Пользователь может изменять настройки по своему усмотрению при работе с программным средством.

Существуют общие требования, которые предъявляются к программным приложениям:

- соответствие стандартам организации интерфейса: использование многооконного подхода, реализация управления работой программного приложения с помощью элементов управления;
- требования к выбранной цветовой схеме рабочего экрана;
- выполнение одной функции с помощью разных элементов управления.

Для обучения пользователя необходимо разработать справочную систему, в которой должны быть раскрыты все аспекты работы с программой, возможные трудности, возникшие во время работы и пути их решения.

Для удобства работы пользователя с программным средством необходимо при разработке форм придерживаться единого стиля оформления. Формы не должны быть перегружены излишней информацией или содержать информацию, не относящуюся к данной форме. Также необходимо предусмотреть защиту данных от удаления и изменения, а также от ввода некорректных данных. В случае ввода некорректных данных или попытке совершить запрещенные действия, пользователь должен быть проинформирован о своих действиях с помощью диалоговых окон.

2.2 Проектирование модели

Цель моделирования данных состоит в обеспечении разработчика информационной системы концептуальной схемой базы данных в форме одной или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных.

Наиболее распространенным средством моделирования данных является диаграмма «Сущность-связь» (ERD). С их помощью определяется важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). ERD непосредственно используются для проектирования реляционных баз данных.

Диаграмма «Сущность-связь» в нотации Чена представлена на рисунке 2.1.

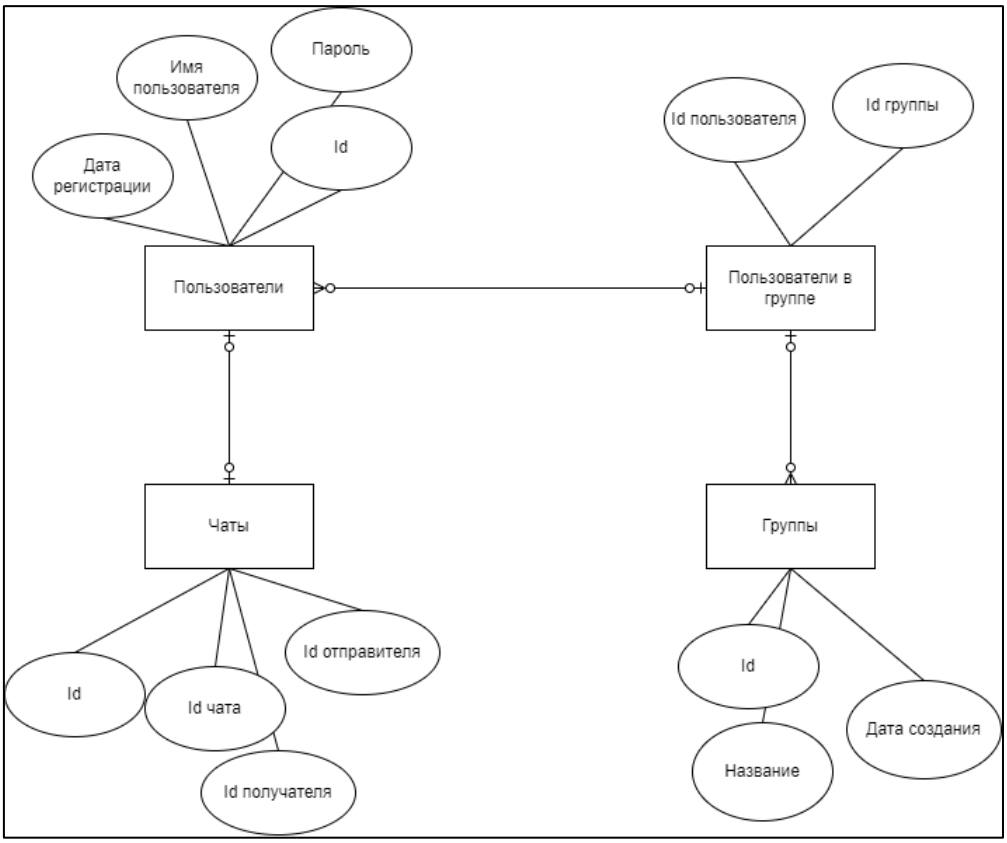

Рисунок 2.1

Исходя из предметной области можно выделить следующие сущности разработки: «Пользователи», «Чаты», «Группы» и «Пользователи в группе».

Для сущности «Пользователи» атрибутами будут являться:

- id;
- имя;
- пароль;
- дата регистрации.

Для сущности «Чаты» атрибутами будут являться:

- id;
- id чата;
- id отправителя;
- id получателя.

Для сущности «Группы» атрибутами будут являться:

- id;
- название;
- дата создания.

Для сущности «Пользователи в группе» атрибутами будут являться:

- id пользователя;
- id группы.

Суть диаграммы вариантов использования заключается в том, что проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых вариантов использования.

К «extend» относится информация о неудаче:

- вывод сообщения об ошибке;
- отклонения запроса.

Также к «extend» относится подключение к базе данных и вызов справочной системы.

К «include» относится:

- авторизация;
- регистрация;
- отправка сообщения;
- отправка файла;
- изменение темы;
- настройка профиля;
- открытие чата или группы
- создание группы.

Диаграмма вариантов использования представлена на листе 1 графической части.

Диаграмма классов служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывать их внутреннюю структуру и типы отношений.

При реализации программного средства были реализованы следующие основные классы:

- форма приветствия;
- форма авторизации;
- форма регистрации;
- главная форма;
- форма чата;
- форма создания группы;
- форма настроек.

Класс «форма приветствия» отвечает за включение форм авторизации, регистрации и главной и логику их появления. Класс «форма авторизации» отвечает за включение форм регистрации и главной и логику авторизации нового пользователя. Класс «форма регистрации» отвечает за включение форм регистрации и главной и логику регистрации пользователя. Класс «главная форма» отвечает за включение форм чата, создания группы и настроек. Класс «форма чата» отвечает за отображение сообщений в окне и переписку между пользователями. Класс «форма создания группы» отвечает за создания новой

группы. Класс «форма настроек» позволяет редактировать информацию о пользователи и его фотографию пользователя.

Диаграмма классов представлена на листе 2 графической части.

При моделировании поведения проектируемой или анализируемой системы возникает необходимость детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности.

Диаграммы деятельности – частный случай диаграмм состояний. Основная цель использования таких диаграмм – визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. Диаграмма последовательности представлена на листе 3 графической части.

2.3 Концептуальный прототип

Концептуальный прототип состоит из описания внешнего пользовательского интерфейса, а именно, элементов управления.

При создании данного приложения важную роль играют формы, так как они являются основным средством работы с пользователем. Разрабатываемое приложение будет содержать несколько форм.

Все формы будут содержать стандартные пользовательские элементы управления.

В рабочем режиме программы, пользователю, для удобной навигации, будет предоставлено меню.

Первая форма, которую увидит пользователь будет форма приветствия, которая будет приветствовать пользователя в приложении. Данная форма будет содержать кнопку «Next». Кнопка «Next» будет открывать окно регистрации авторизации или главное окно, в зависимости от сохраненных в приложении данных пользователя. Макет формы представлен на рисунке 2.2..

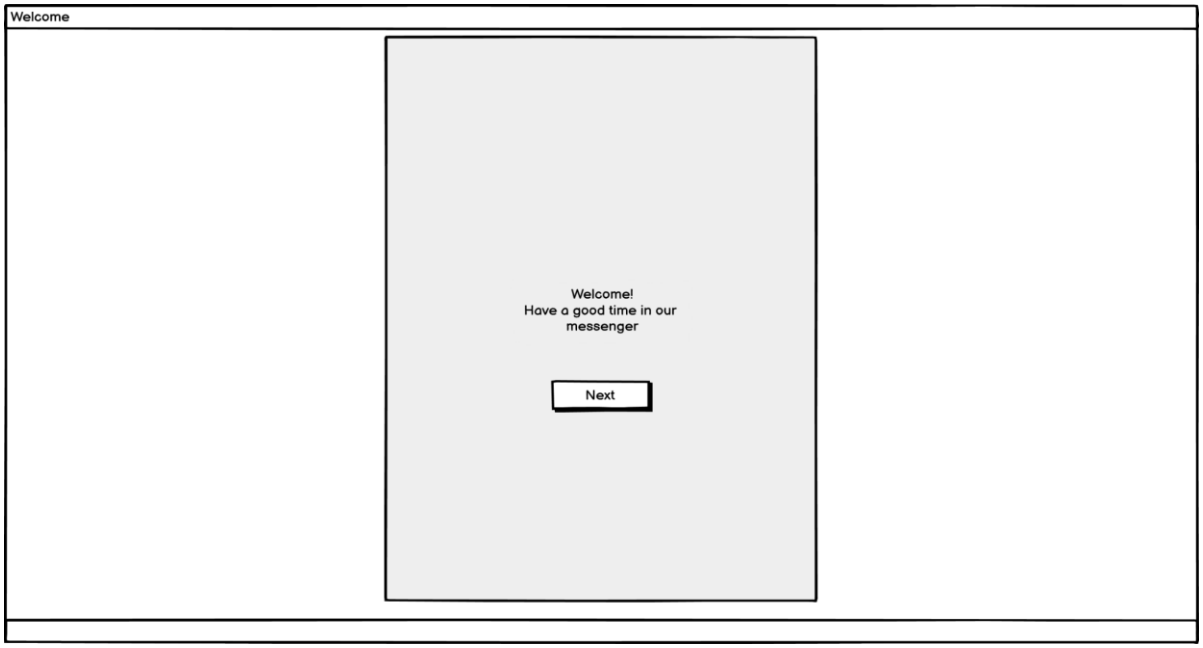

Рисунок 2.2

При нажатии на кнопку «Next» откроется форма. Форма «Регистрация» аналогична форме «Авторизация». Данная форма будет содержать кнопку «Sing in» и «Sing up». Кнопка «Sing in» будет авторизовать пользователя в приложении. Кнопка «Sing up» будет открывать окно регистрации. Макет формы представлен на рисунке 2.3..

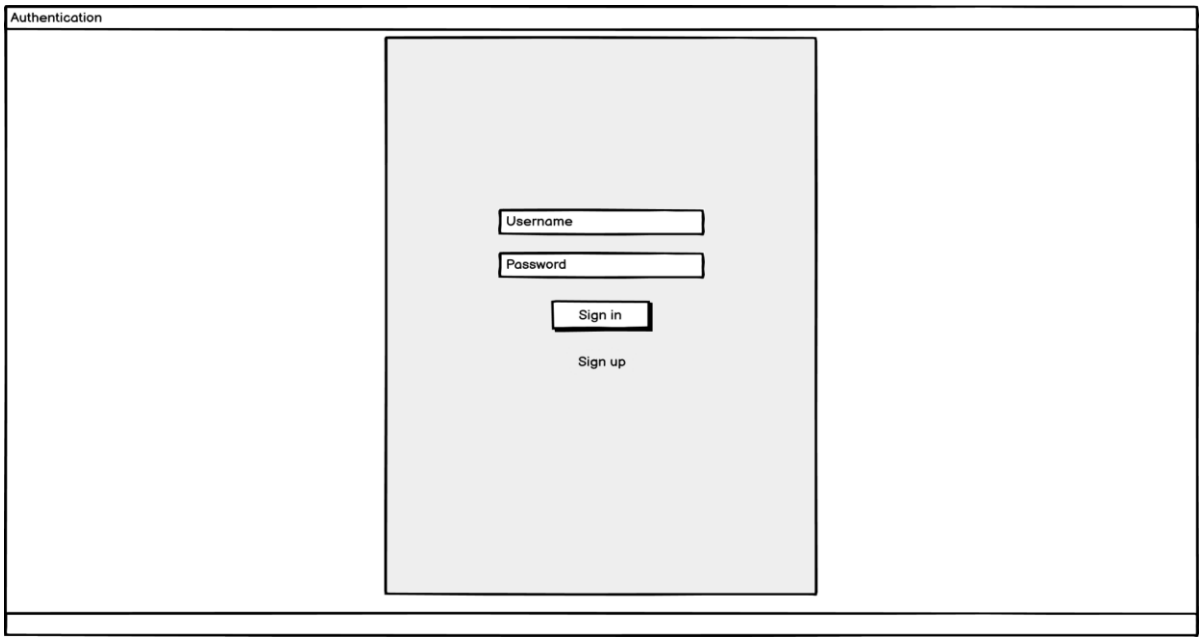

Рисунок 2.3

После успешного входа в систему открывается главная форма. Данная форма будет иметь главное меню, форму чата и всплывающее меню. Главное меню будет содержать список чатов и групп, кнопку поиска, кнопку создания группы и кнопку открытия всплывающего меню. Форма чата будет содержать сообщения в выбранном чате. Макет формы представлен на рисунке 2.4..

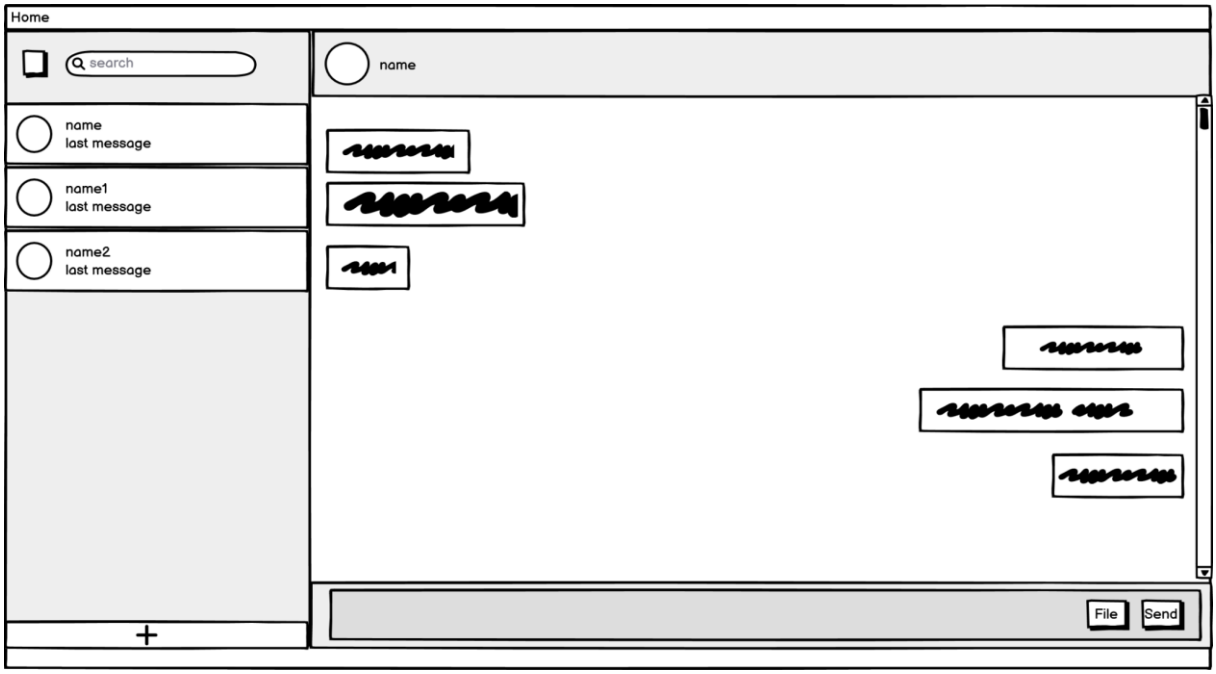

Рисунок 2.4

После открытия всплывающего меню становятся доступны кнопки «Settings» и «Logout». Кнопка «Settings» открывает форму настроек. Кнопка «Logout» открывает окно авторизации и выходит из учетной записи пользователя. Макет формы представлен на рисунке 2.5..

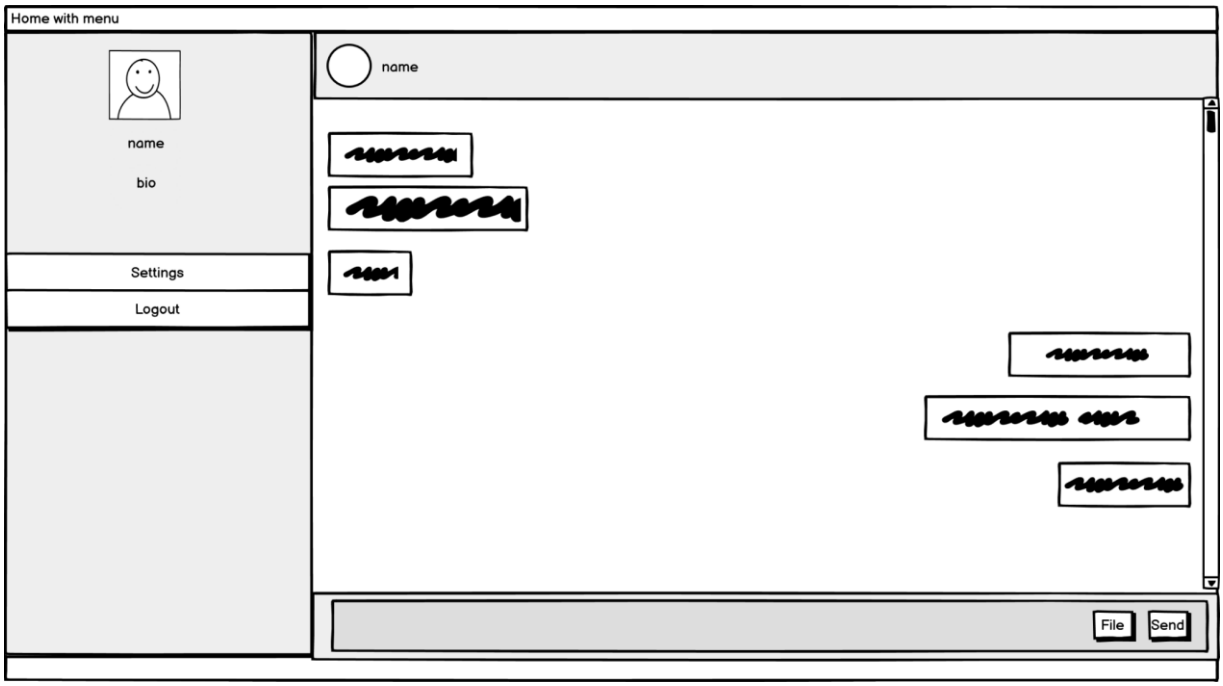

Рисунок 2.5

3 Описание реализации программного средства

3.1 Инструменты разработки и применяемые технологии

Инструментами для разработки данного программного средства будут являться:

- операционная система Windows 10;
- среда программирования Visual Studio;
- язык программирования C#.
- база данных MS SQL Server ;
- менеджер базами данных SQL Server Management Studio 19;
- система контроля версий GitHub;
- сайта для разработки диаграмм Draw.io.

Для разработки программного средства выбрана операционная система Windows 10.

Интегрированная среда разработки Visual Studio – это стартовая площадка для написания, отладки и сборки кода, а также последующей публикации приложений. Интегрированная среда разработки (IDE) представляет собой многофункциональную программу, которую можно использовать для различных аспектов разработки программного обеспечения. Помимо стандартного редактора и отладчика, которые существуют в большинстве сред IDE, Visual Studio включает в себя компиляторы, средства автозавершения кода, графические конструкторы и многие другие функции для упрощения процесса разработки [4].

Язык программирования C# – это объектно- и компонентно-ориентированный язык программирования. Он предоставляет языковые конструкции для непосредственной поддержки такой концепции работы. Благодаря этому C# подходит для создания и применения программных компонентов. С момента создания язык C# обогатился функциями для поддержки новых рабочих нагрузок и современными рекомендациями по разработке ПО [5].

SQL Server — это реляционная база данных, разработанная корпорацией Microsoft. Она является клиент-серверной системой, в которой клиентские приложения могут подключаться к серверу, чтобы получить доступ к базе данных. SQL Server поддерживает не только язык SQL, но и многие другие языки программирования, такие как C#, Java и Python. SQL Server не является встраиваемой базой данных, как SQLite. Вместо этого SQL Server использует парадигму клиент-сервер. Это означает, что сервер является отдельным процессом, который работает непрерывно и обрабатывает запросы от клиентов. SQL Server может хранить базу данных на разных компьютерах, а не только на том компьютере, на котором исполняется программа, как в случае с SQLite. Это означает, что многие пользователи могут одновременно работать с базой данных, если имеют соответствующие права доступа.

Для управления базой данных SQL Server можно использовать различные менеджеры баз данных, такие как SQL Server Management Studio (SSMS). Он предоставляет множество функций для работы с базой данных, включая создание таблиц, индексов и запросов. Он может быть загружен с официального сайта Microsoft и установлен на компьютер. [6].

GitHub – крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git и разработан на Ruby on Rails и Erlang компанией GitHub, Inc (ранее Logical Awesome). Сервис бесплатен для проектов с открытым исходным кодом и (с 2019 года) небольших частных проектов, предоставляя им все возможности (включая SSL), а для крупных корпоративных проектов предлагаются различные платные тарифные планы [7].

Draw.io – это бесплатный инструмент для создания диаграмм и схем онлайн. Он предлагает широкий выбор инструментов для рисования, таких как блоки, стрелки, линии, графики и многое другое. [8].

3.2 Организация данных

Организация данных подразумевает создание модели данных, главными элементами которой являются сущности и их связи.

Реляционная модель основана на математическом понятии отношения, представлением которого является таблица. В реляционной модели отношения используются для хранения информации об объектах, представленных в базе данных. Отношение имеет вид двумерной таблицы, в которой строки соответствуют записям, а столбцы – атрибутам. Для этого используют первичные и внешние ключи. Достоинством реляционной модели является простота и удобство физической реализации. Для этого используют первичные и внешние ключи.

Структура базы данных разрабатываемого программного средства включает шесть таблиц. Описание таблиц приводится в таблицах 1-5.

Таблица «Users» хранит информацию о пользователях, структура которой представлена в таблице 3.1.

Таблица 3.1 – Структура таблицы «Users»

Имя поля	Тип поля	Размер поля, байт	Описание поля
Id	uniqueid entifier	16	Идентификатор пользователе
Username	varchar	128	Имя пользователя3
Password	nvarchar	128	Пароль пользователя

Продолжение таблицы 3.1

Имя поля	Тип поля	Размер поля, байт	Описание поля
CreationTime	datetime	2	Дата создания

Таблица «Chats» хранит информацию о чатах, структура которой представлена в таблице 3.2.

Таблица 3.2 – Структура таблицы «Chats»

Имя поля	Тип поля	Размер поля, байт	Описание поля
Id	int	4	Идентификатор чата
ChatId	uniqueid entifier	16	Id чата
SenderId	uniqueid entifier	16	Id отправителя
ReceiverId	uniqueid entifier	16	Id получателя

Таблица «UserGroups» хранит информацию о пользователях которые вступили в группу, структура таблицы представлена в таблице 3.3.

Таблица 3.3 – Структура таблицы «UserGroups»

Имя поля	Тип поля	Размер поля, байт	Описание поля
UserId	uniqueid entifier	16	Идентификатор пользователя
GroupId	uniqueid entifier	16	Идентификатор группы

Таблица «Groups» хранит информацию о группах. Структура таблицы представлена в таблице 3.4.

Таблица 3.4 – Структура таблицы «Groups»

Имя поля	Тип поля	Размер поля, байт	Описание поля
Id	uniqueid entifier	16	Идентификатор группы
Name	nvarchar	50	Имя группы
CreatedAt	datetime	2	Дата создания группы

Схема базы данных представлена в виде ERD диаграммы в нотации Баркера на рисунке 3.1.

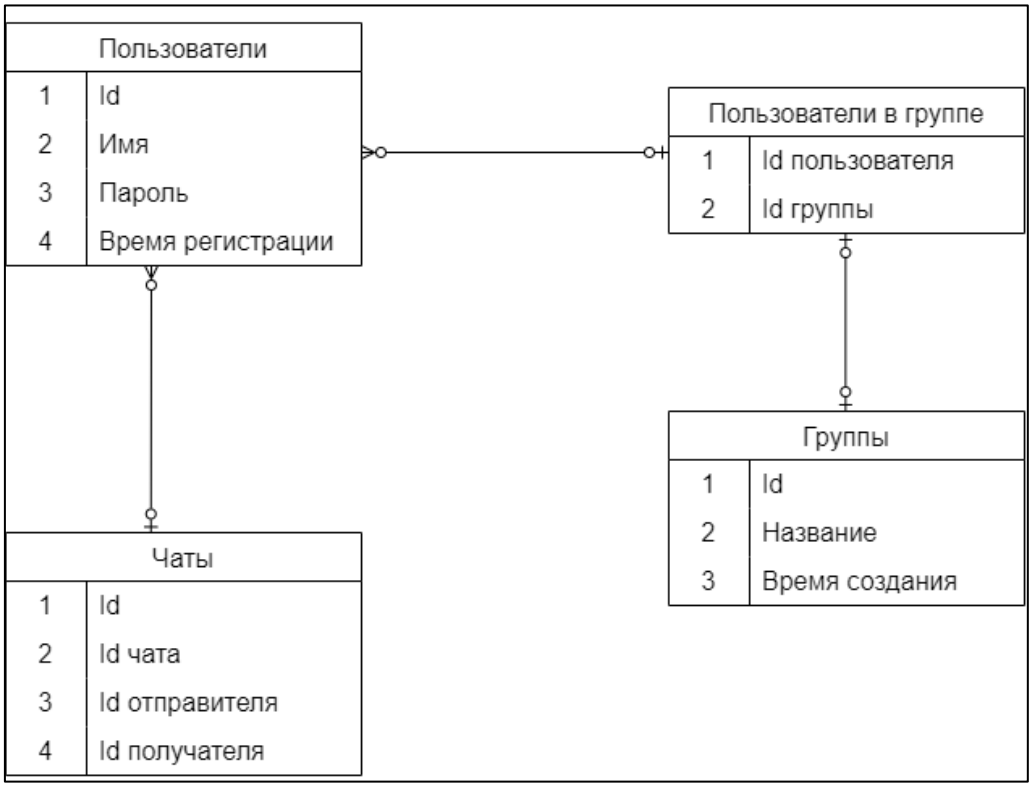

Рисунок 3.1

3.4 Функции: логическая и физическая организация

На основании диаграммы вариантов использования в программном средстве реализованы функции авторизации и регистрации.

Так как модель приложения имеет клиент-серверную структуру, код функций будет приводится как с сервера, так и с клиента.

Рассмотрим данные функции программы. Для того, чтобы авторизоваться в приложении, пользователю необходимо ввести имя пользователя и пароль. Предварительно пользователь должен быть зарегистрирован в системе. После нажатия кнопки «Войти» клиент отправляет запрос на сервер, на котором уже производится дальнейшая обработка запроса. Если в результате проверки данные не были найдены в базе данных сервера, то вход в приложение отменяется и на клиенте отображается сообщение об ошибке.

Код функции авторизации на клиенте предоставлен ниже.

```
namespace Client.Commands.Users;

public record ReceivedData(string Token, Guid Id);

public sealed class AuthenticationCommand : CommandBase
{
```

```
        private readonly AuthenticationViewModel _authenticationViewModel;

        private readonly HttpClient _httpClient;

        private readonly INavigationService _navigationService;

        private readonly UserStore _userStore;

        public AuthenticationCommand(AuthenticationViewModel authenticationViewModel,
            UserStore userStore, HttpClient httpClient, INavigationService navigationService)
        {
            _authenticationViewModel = authenticationViewModel;
            _httpClient = httpClient;
            _userStore = userStore;
            _navigationService = navigationService;

            authenticationViewModel.PropertyChanged += OnPropertyChanged;
        }

        private void OnPropertyChanged(object? sender,
            PropertyChangedEventArgs e)
        {
            if (e.PropertyName == nameof(AuthenticationViewModel.UserName) ||
                e.PropertyName == nameof(AuthenticationViewModel.Password))
                OnCanExecutedChanged();
        }

        public override bool CanExecute(object? parameter)
        {
            return !string.IsNullOrEmpty(_authenticationViewModel.UserName) &&
                   !string.IsNullOrEmpty(_authenticationViewModel.Password);
        }

        public override async void Execute(object? parameter)
        {
            _authenticationViewModel.IsLoading = true;

            _userStore.User = new UserModel(
                Guid.Empty,
                _authenticationViewModel.UserName,
                _authenticationViewModel.Password);
```

```
            var content = new StringContent(JsonConvert.SerializeObject(_userStore.User),
                Encoding.UTF8, "application/json");

            var response = await _httpClient.PostAsync("/authentication/auth", content);

            response.EnsureSuccessStatusCode();

            if (response.IsSuccessStatusCode)
            {
                var receivedData = await response.Content.ReadAsAsync<ReceivedData>();

                _userStore.User.Id = receivedData.Id;
                _userStore.Token = receivedData.Token.Trim('"');

                _navigationService.Navigate();
            }

            _authenticationViewModel.IsLoading = false;
        }
    }
```

Код функции авторизации на сервере предоставлен ниже.

```
namespace Application.Users.Commands.UserAuthentication;


public class UserAuthenticationCommandHandler : ICommandHandler<UserAuthenticationCommand, Result<AuthenticationResponse>>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IUserRepository _userRepository;

    public UserAuthenticationCommandHandler(IUserRepository userRepository, IJwtProvider jwtProvider)
    {
        _userRepository = userRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result<AuthenticationResponse>> Handle(UserAuthenticationCommand request,
```

```
            CancellationToken cancellationToken)
        {
            var user = await
_userRepository.GetByUsernameAsync(request.Username, cancellationToken)!;

            if (user == null)
                return Result.Failure<AuthenticationResponse>(
                    new Error("The user with the specified user name was not found."));

            if (user.Password != request.Password)
                return Result.Failure<AuthenticationResponse>
                    (new Error("Password wasn't match"));

            var token = _jwtProvider.GetJwtToken(user);

            return Result.Success(new AuthenticationResponse(token, user.Id));
        }
    }
```

Для регистрации, пользователю необходимо перейти на форму регистрации, нажав кнопку «Регистрация». Далее пользователь должен заполнить все текстовые поля. Для успешной регистрации пользователю должен ввести уникальное имя пользователя и пароль. После нажатия кнопки «Зарегистрироваться», клиент отправляет запрос на сервер. Код функции регистрации на клиенте представлен ниже

```
    namespace Client.Commands.Users;

    public class RegistrationCommand : CommandBase
    {
        private readonly HttpClient _httpClient;

        private readonly RegistrationViewModel _registrationViewModel;

        private readonly INavigationService _navigationService;

        private readonly UserStore _userStore;

        public RegistrationCommand(RegistrationViewModel
registrationViewModel, UserStore userStore,
            HttpClient httpClient, INavigationService navigationService)
        {
            _userStore = userStore;
            _httpClient = httpClient;
            _navigationService = navigationService;
```

```
            _registrationViewModel = registrationViewModel;

            _registrationViewModel.PropertyChanged += OnPropertyChanged;
        }

        private void OnPropertyChanged(object? sender, PropertyChangedEventArgs
e)
        {
            if (e.PropertyName == nameof(RegistrationViewModel.UserName) ||
                e.PropertyName == nameof(RegistrationViewModel.Password) ||
                e.PropertyName == nameof(RegistrationViewModel.IsAgree))
                OnCanExecutedChanged();
        }

        public override bool CanExecute(object? parameter)
        {
            return !string.IsNullOrEmpty(_registrationViewModel.UserName) &&
                   !string.IsNullOrEmpty(_registrationViewModel.Password) &&
                   _registrationViewModel.IsAgree;
        }

        public override async void Execute(object? parameter)
        {
            _registrationViewModel.IsLoading = true;

            _userStore.User = new UserModel(Guid.Empty,
_registrationViewModel.UserName,
                _registrationViewModel.Password);

            var content = new
StringContent(JsonConvert.SerializeObject(_userStore.User),
                Encoding.UTF8, "application/json");

            var response = await _httpClient.PostAsync("/registration/reg", content);

            response.EnsureSuccessStatusCode();

            if (response.IsSuccessStatusCode)
            {
                var receivedData = await
response.Content.ReadAsAsync<ReceivedData>();

                _userStore.Token = receivedData.Token.Trim('"');

                _userStore.User.Id = receivedData.Id;
```

```
            _navigationService.Navigate();
        }

        _registrationViewModel.IsLoading = false;
    }
}
```

Код функции регистрации на сервере представлен ниже.

```
namespace Application.Users.Commands.UserRegistration;

public sealed class
    UserRegistrationCommandHandler :
ICommandHandler<UserRegistrationCommand, Result<RegistrationResponse>>
{
    private readonly IApplicationDbContext _applicationDbContext;
    private readonly IJwtProvider _jwtProvider;
    private readonly IUserRepository _userRepository;

    public UserRegistrationCommandHandler(IUserRepository userRepository,
IApplicationDbContext applicationDbContext,
        IJwtProvider jwtProvider)
    {
        _userRepository = userRepository;
        _applicationDbContext = applicationDbContext;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result<RegistrationResponse>>
Handle(UserRegistrationCommand request,
        CancellationToken cancellationToken)
    {
        var id = Guid.NewGuid();

        var user = new User(id, request.Username, request.Password);

        var result = await _userRepository.CreateAsync(user, cancellationToken);

        if (result.IsFailure)
            return Result.Failure<RegistrationResponse>(result.Error);

        await _applicationDbContext.SaveChangesAsync(cancellationToken);
```

```
            var token = _jwtProvider.GetJwtToken(user);

            return Result.Success(new RegistrationResponse(token, id));
        }
    }
```

Полный текст программы представлен в приложении А.

3.5 Входные и выходные данные

Входными являются данные, введённые пользователем в таблицы базы данных: «Users», «Groups», «Chats», «UserGroups».

Входными данным при добавлении данных о новом пользователе будут являться:

- Имя пользователя;
- Пароль.

Входными данными при добавлении данных о новом сообщении являются:

- Текст сообщения;
- Дата отправки.

Входными данными при создании группы являются:

- имя группы;
- id пользователя.

Вся вносимая в программное средство информация хранится в локальной базе данных.

Выходные данные – это полученные с базы данных данные, преобразующиеся в понятный для пользователя вид, отображаемые при экспорте документа. Пример выходных данных представлен на рисунке Б.1.

3.6 Функциональное тестирование

Функциональное тестирование – это тестирование функций приложения на соответствие требованиям и проводится для выявления неполадок и недочетов программы на этапе ее сдачи в эксплуатацию.

Проведем тестирование проверки каждого пункта меню с целью проверки всех функций.

Тестирование программы будет производиться последовательно, переходя из одной части программы в другую. Во время теста будут проверяться все

действия с программой, навигация пунктам меню, которые может произвести пользователь.

В таблице 3.5 представлены тест-кейсы, подготовленные для проведения функционального тестирования.

Таблица 3.5 – Тест-кейсы

№	Модуль / Функция	Шаги выполнения	Ожидаемый результат	Фактический результат
1	Регистрация	1. Перейти в окно «Регистрации». 2. В поле «Имя» ввести строку «выаывдл». 3. В поле «Пароль» ввести строку «qwerty». 4. Нажать кнопку «Регистрация».	Создание нового аккаунта и открытие главного окна.	Пользователь зарегистрирован и открылось главное окно. Результат представлен на рисунке Б.1
2	Смена автарки	1. Перейти в главное окно. 2. Открыть всплывающее окно. 3. Нажать на токующую аватарку. 4. Выбрать необходимый файл.	Успешная смена аватарки пользователя.	Аватарка заменена на выбранный файл. Результат представлен на рисунке Б.2
3	Поиск пользователей	1. Перейти в главное окно. 2. В поле «Поиск» ввести строку «Му В». 3. Нажать кнопку поиска.	Отображение всех пользователей с именем начинающемся на «Му В»	В списке пользователей отобразились все пользователи начинающиеся на «Му В». Результат представлен на рисунке Б.3
4	Отправка сообщения	1. Перейти в главное окно. 2. В поле «Поиск» ввести строку «Му В».	Сообщение отправилось и отобразилось у пользователей.	Сообщение сохранено на сервере и отобразилось у пользователя.

Продолжение таблицы 3.5

№	Модуль / Функция	Шаги выполнения	Ожидаемый результат	Фактический результат
		3. Нажать кнопку поиска. 4. Выбрать пользователя «My Bebras». 5. В строку сообщения ввести строку «вававав». 6. Нажать кнопку отправить.		Результат представлен на рисунке Б.4
5	Смена темы	1. Перейти в главное окно. 2. Открыть всплывающее окно. 3. Нажать на кнопку «Settings». 4. Нажать на выпадающий список и выбрать «Light». 5. Закрыть приложение. 6 Открыть его заново.	Приложение перезагрузится в светлой теме.	Приложение открывается в светлой теме. Результат представлен на рисунке Б.5
6	Создание группы	1. Перейти в главное окно. 2. Нажать кнопку создания группы. 3. В поле название группы ввести «Супер-группа». 4. Нажать на кнопку «Create».	Группа появилась в списке чатов и групп.	Группа успешно создана и отображается в списке чатов и групп. Результат представлен на рисунке Б.6

3.7 Описание справочной системы

Для эффективной работы пользователя с приложением необходимо обеспечить его качественной справочной системой, в которой должны быть приведены методы и приемы работы с приложением, включающие данные о том, что произойдет после нажатия на определенную кнопку или при выборе пункта

меню, сведения о том, какую информацию и в каком виде следует вводить в соответствующие поля, каким образом можно работать.

Справочная система будет представлена в виде встроенного файла в приложении, содержащая информацию по эксплуатации программного средства.

Структура справочной системы приведена на рисунке 3.2.

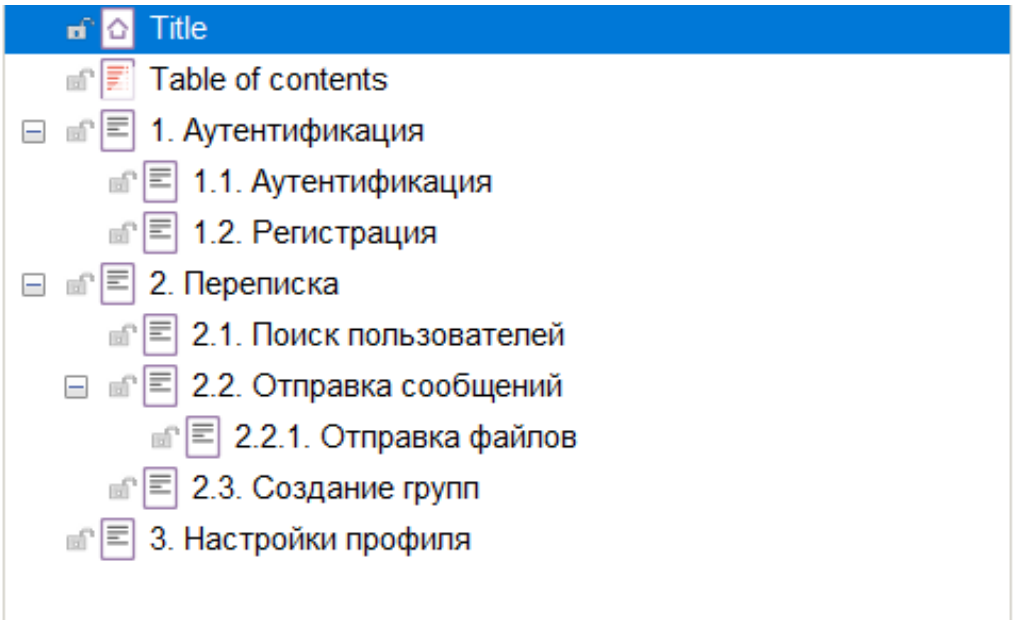

Рисунок 3.2

4 Применение

4.1 Назначение программного средства

Программное средство «Near Message» разрабатывалось с целью создания приложения, с помощью которого люди смогут вести переписку текстовыми сообщениями друг с другом. Данная программа была разработана на ноутбуке Asus VivoBook с операционной системой Windows 10 Pro, со следующей конфигурацией:

- процессор AMD Ryzen 7 4700U;
- интегрированная видеокарта Radeon Vega Mobile Gfx 2.30 GHz;
- ОЗУ 16,00 ГБ.

Программа создана в средстве разработки Microsoft Visual Studio 2022 на языке программирования C#. Она может работать в средах операционных систем семейства Microsoft Windows, начиная с Windows 7.

Для работы программы, необходима база данных «NearMessage.bak». Программа не требовательна к системным ресурсам и не имеет больших массивов данных, также проста в использовании и не требует специальных навыков при работе. Для работы данного программного средства необходима предварительная установка и настройка следующих программных продуктов:

- платформа Microsoft Net Framework версии 4.7.2.

В состав программных входят:

- программа «Client.exe»;
- программа «NearMessage.API.exe»;
- база данных «NearMessage.bak»;
- файл справочной системы «help.chm».

Объем установленного приложения не превышает 140 МБ.

4.2 Условия применения

Для работы с программой необходимо наличие программного обеспечения:

- операционная система, начиная с Windows 7;
- библиотека Microsoft .NET Framework 4.7.2..

На случай редактирования проекта программы необходимо наличие программного обеспечения:

- система управления базами данных SQL Server Management Studio (SSMS).

Заключение

В рамках курсового проекта на тему «Программная реализация мессенджера «Near Message» было разработано программное средство «Client.exe».

Основными функциями приложения являются:

- ведение локального хранилища сообщений, данных о пользователи на клиенте;
- ведение базы данных на сервере, в которой будет храниться вся информация;
- ведение хранилища на сервере, в которой будет храниться сообщения и данные о пользователях;
- авторизация пользователей;
- регистрация пользователей;
- переписка пользователей, с использованием текстовых сообщений и файлов;
- поиск других пользователей.

Для достижения целей курсового проекта были решены следующие задачи:

- изучена предметная область;
- разработана физическая и логическая модель данных;
- разработано программное средство;
- описана область применения, созданного программное средство.

Разработка имеет интуитивно понятный графический интерфейс, позволяющий даже с минимальным знанием компьютера использовать данное программного средство.

Программа реализована в полном объеме и в соответствии с заданными требованиями, полностью отлажена и протестирована. Поставленные задачи выполнены.

К достоинствам программного средства можно отнести простоту использования, малые массивы данных.

В процессе разработки данного программного средства были применены и закреплены знания по уже изученному материалу, были отработаны навыки владения методами надежного программирования и эффективности разработки программного обеспечения в Microsoft Visual Studio 2022 с использованием языка программирования C#, разработана база данных средствами системы управления базами данных SQL Server Management Studio.

Список использованных источников

1. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск: КБП, 2023

2. Михалевич В.Ю. Методические указания к курсовому проектированию / В.Ю. Михалевич. – Минск: КБП, 2023

3. Общие требования к текстовым документам: ГОСТ 2.105-95. – Введ. 01.01.1996. – Минск: Межгос. совет по стандартизации, метрологии и сертификации, 1995. – 84 с. 3

4. Visual Studio [Электронный ресурс]. – Microsoft, 2020. – Режим доступа : https://visualstudio.microsoft.com/vs/. – Дата доступа : 01.09.2020.

5. .NET Framework [Электронный ресурс]. – Microsoft, 2020. – Режим доступа : https://docs.microsoft.com/en-us/dotnet/framework/. – Дата доступа : 01.09.2023

6. SQL Server [Электронный ресурс]. – Microsoft, 2021. – Режим доступа: https://www.microsoft.com/en-us/sql-server/sql-server-2019. – Дата доступа: 04.05.2023.

7. GitHub [Электронный ресурс]. – github, 2020. – Режим доступа : https://www.github.com/. – Дата доступа : 05.05.2023.

8. Diagrams [Электронный ресурс]. – UML, 2021. – Режим доступа : https://www.diagrams.net/. – Дата доступа : 10.05.2023.

9. MS Office [Электронный ресурс]. – Microsoft, 2021. – Режим доступа : https://officeapplications.net/microsoft-word/. – Дата доступа : 03.05.2023.

10. Smart Install Maker [Электронный ресурс]. – InstallBuilders, 2020. – Режим доступа : https://www.ixbt.com/news/soft/index.shtml?10/52/79/. – Дата доступа : 05.05.2021.

Приложение А
(обязательное)
Текст программных модулей

```
using NearMessage.Domain.Users;

namespace NearMessage.Domain.Chats;

public class Chat
{
    public Chat(Guid chatId, Guid senderId, Guid receiverId)
    {
        ChatId = chatId;
        SenderId = senderId;
        ReceiverId = receiverId;
    }

    public int Id { get; set; }

    public Guid ChatId { get; set; }

    public Guid SenderId { get; set; }

    public Guid ReceiverId { set; get; }

    public virtual User? Sender { get; set; }

    public virtual User? Receiver { get; set; }

    public Chat InversedChat => new(ChatId, ReceiverId, SenderId);
}

using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;

namespace NearMessage.Domain.Chats;

public interface IChatRepository
{
    Task<Result<Chat>> CreateChatAsync(Guid user1, Guid user2,
        CancellationToken cancellationToken);

    Task<Maybe<Chat>> GetChatByUsersIdAsync(Guid user1, Guid user2,
        CancellationToken cancellationToken);

    Task<Maybe<Chat>> GetChatByIdAsync(Guid chatId, CancellationToken cancellationToken);
}

using NearMessage.Domain.Primitives;
```

```
namespace NearMessage.Domain.Contacts;

public class Contact : Entity
{
    public Contact(Guid id, string username, Guid? chatId)
        : base(id)
    {
        Username = username;
        ChatId = chatId;
    }

    public string Username { get; set; }

    public Guid? ChatId { get; set; }
}

using NearMessage.Common.Primitives.Errors;

namespace NearMessage.Domain.Exceptions;

public class DomainException : Exception
{
    public DomainException(Error error)
        : base(error.Message)
    {
        Error = error;
    }

    public Error Error { get; }
}

using NearMessage.Domain.UserGroups;
using NearMessage.Domain.Users;

namespace NearMessage.Domain.Groups;

public class Group
{
    public Group(Guid id, string name)
    {
        Id = id;
        Name = name;
        CreatedAt = DateTime.Now;
    }

    public Guid Id { get; set; }

    public string Name { get; set; }

    public DateTime CreatedAt { get; set; }
```

```
    public virtual List<UserGroup>? UserGroups { get; set; }
}

using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Users;

namespace NearMessage.Domain.Groups;

public interface IGroupRepository
{
    Task<Result<Group>> CreateGroupAsync(Group group, CancellationToken cancellationToken);
    Task<IEnumerable<Group>> GetGroupsByUsernameAsync(string groupName,
CancellationToken cancellationToken);
}

using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;

namespace NearMessage.Domain.Messages;

public interface IMessageRepository
{
    Task<Result> SaveMessageAsync(Media message, CancellationToken cancellationToken);

    Task<Result<IEnumerable<Media>>> GetMessagesAsync(Guid chatId, CancellationToken
cancellationToken);

    Task<Maybe<IEnumerable<Media>>> GetLastMessagesAsync(Guid chatId, DateTime
lastMessageDate,
        CancellationToken cancellationToken);
}

using NearMessage.Domain.Contacts;

namespace NearMessage.Domain.Messages;

public class Media : Message
{
    public Media(Guid id, string content, Guid sender,
        Guid receiverChatId, byte[]? fileData = null, string? fileName = null)
        : base(id, content, sender, receiverChatId)
    {
        FileData = fileData;
        FileName = fileName;
    }

    public string? FileName { get; set; }

    public byte[]? FileData { get; set; }
}
```

```
using NearMessage.Domain.Contacts;
using NearMessage.Domain.Primitives;

namespace NearMessage.Domain.Messages;

public class Message : Entity
{
    public Message(Guid id, string content, Guid sender, Guid receiverChatId)
        : base(id)
    {
        Content = content;
        SendTime = DateTime.Now;
        Sender = sender;
        ReceiverChatId = receiverChatId;
    }

    public string Content { get; set; }

    public DateTime SendTime { get; set; }

    public Guid Sender { get; set; }

    public Guid ReceiverChatId { get; set; }
}

namespace NearMessage.Domain.Primitives;

public abstract class Entity
{
    protected Entity(Guid id)
    {
        Id = id;
    }

    public Guid Id { get; protected set; }
}

using NearMessage.Common.Primitives.Result;

namespace NearMessage.Domain.UserGroups;

public interface IUserGroupRepository
{
    Task<Result<UserGroup>> AddUserGroupAsync(UserGroup userGroup, CancellationToken
cancellationToken);
}

using NearMessage.Domain.Groups;
using NearMessage.Domain.Users;

namespace NearMessage.Domain.UserGroups;
```

```
public class UserGroup
{
    public UserGroup(Guid userId, Guid groupId)
    {
        UserId = userId;
        GroupId = groupId;
    }

    public Guid UserId { get; set; }

    public virtual User? User { get; set; }

    public Guid GroupId { get; set; }

    public virtual Group? Group { get; set; }
}

using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Contacts;

namespace NearMessage.Domain.Users;

public interface IUserRepository
{
    Task<Result<User>> CreateUserAsync(User user, CancellationToken cancellationToken);
    Task<Maybe<User>> GetByIdAsync(Guid id, CancellationToken cancellationToken);
    Task<Maybe<User>> GetByUsernameAsync(string username, CancellationToken
cancellationToken);
    Task<IEnumerable<User>> GetUsersByUsernameAsync(string username, CancellationToken
cancellationToken);
    Task<Result<Contact>> ConvertToContactAsync(User user, Guid user1, CancellationToken
cancellationToken);
}

using NearMessage.Domain.Chats;
using NearMessage.Domain.Groups;
using NearMessage.Domain.Primitives;
using NearMessage.Domain.UserGroups;

namespace NearMessage.Domain.Users;

public class User : Entity
{
    public User(Guid id, string username, string password)
        : base(id)
    {
        Username = username;
        Password = password;
        CreatedAt = DateTime.Now;
```

```
    }

    public string Username { get; set; }

    public string Password { get; set; }

    public DateTime CreatedAt { get; set; }

    public virtual List<Chat>? SentChats { get; set; }

    public virtual List<Chat>? ReceivedChats { get; set; }

    public virtual List<UserGroup>? UserGroups { get; set; }

    public bool VerifyPassword(string password)
    {
        return password == Password;
    }
}

using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;

namespace NearMessage.Domain.UsersInformation;

public interface IUserInformationRepository
{
    Task<Result> SaveUserInformationAsync(UserInformation userInformation,Guid id,
        CancellationToken cancellationToken);
}

using NearMessage.Domain.Primitives;

namespace NearMessage.Domain.UsersInformation;

public class UserInformation : Entity
{
    public UserInformation(Guid id, string? about, string name) : base(id)
    {
        About = about;
        Name = name;
    }

    public string? About { get; set; }

    public string Name { get; set; }
}

namespace NearMessage.Infrastructure.Authentication;

public class JwtOptions
```

```
{
    public string Issuer { get; init; } = string.Empty;

    public string Audience { get; init; } = string.Empty;

    public string SecurityKey { get; init; } = string.Empty;
}

using Microsoft.Extensions.Options;
using Microsoft.IdentityModel.Tokens;
using NearMessage.Application.Abstraction;
using NearMessage.Common.Primitives.Maybe;
using NearMessage.Domain.Users;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace NearMessage.Infrastructure.Authentication;

public sealed class JwtProvider : IJwtProvider
{
    private readonly JwtOptions _options;

    public JwtProvider(IOptions<JwtOptions> options)
    {
        _options = options.Value;
    }

    public string Generate(User user)
    {
        var claims = new Claim[]
        {
            new(JwtRegisteredClaimNames.Name, user.Username),
            new(JwtRegisteredClaimNames.Sub, user.Id.ToString())
        };

        var signingCredentials = new SigningCredentials(
            new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(_options.SecurityKey)),
            SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            _options.Issuer,
            _options.Audience,
            claims,
            null,
            DateTime.UtcNow.AddDays(1),
            signingCredentials);

        var tokenValue = new JwtSecurityTokenHandler().WriteToken(token);
```

```
        return tokenValue;
    }

    public Maybe<Guid> GetUserId(ClaimsPrincipal principal)
    {
        var userIdClaim = principal.FindFirst(ClaimTypes.NameIdentifier);

        return userIdClaim == null ? Maybe<Guid>.None :
Maybe<Guid>.From(Guid.Parse(userIdClaim.Value));
    }
}

using Microsoft.EntityFrameworkCore;
using NearMessage.Application.Abstraction;
using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Chats;

namespace NearMessage.Infrastructure.Repository;

internal class ChatRepository : IChatRepository
{
    private readonly INearMessageDbContext _nearMessageDbContext;

    public ChatRepository(INearMessageDbContext nearMessageDbContext)
    {
        _nearMessageDbContext = nearMessageDbContext;
    }

    public async Task<Result<Chat>> CreateChatAsync(Guid user1, Guid user2,
        CancellationToken cancellationToken)
    {
        var chat = new Chat(
            Guid.NewGuid(),
            user1,
            user2);

        await _nearMessageDbContext.Chats.AddAsync(chat, cancellationToken);
        await _nearMessageDbContext.Chats.AddAsync(chat.InversedChat, cancellationToken);
        await _nearMessageDbContext.SaveChangesAsync(cancellationToken);

        return Result.Success(chat);
    }

    public async Task<Maybe<Chat>> GetChatByIdAsync(Guid chatId, CancellationToken
cancellationToken)
    {
        var chat = await _nearMessageDbContext.Chats.SingleOrDefaultAsync(c =>
            c.ChatId == chatId, cancellationToken);

        return chat == null ? Maybe<Chat>.None : Maybe<Chat>.From(chat);
```

```
    }

    public async Task<Maybe<Chat>> GetChatByUsersIdAsync(Guid user1, Guid user2,
CancellationToken cancellationToken)
    {
        var chat = await _nearMessageDbContext.Chats
            .FirstOrDefaultAsync(c =>
                    c.Sender.Id == user1 && c.Receiver.Id == user2,
                cancellationToken);

        if (chat == null) return Maybe<Chat>.None;

        return Maybe<Chat>.From(chat);
    }
}

using Microsoft.EntityFrameworkCore;
using NearMessage.Application.Abstraction;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Groups;
using NearMessage.Domain.Users;

namespace NearMessage.Infrastructure.Repository;

public class GroupRepository : IGroupRepository
{
    private readonly INearMessageDbContext _context;

    public GroupRepository(INearMessageDbContext context)
    {
        _context = context;
    }

    public async Task<Result<Group>> CreateGroupAsync(Group group, CancellationToken
cancellationToken)
    {
        await _context.Groups.AddAsync(group, cancellationToken);
        await _context.SaveChangesAsync(cancellationToken);

        return Result.Success(group);
    }

    public async Task<IEnumerable<Group>> GetGroupsByUsernameAsync(string groupName,
CancellationToken cancellationToken)
    {
        var groups = await _context.Groups
            .AsNoTracking()
            .Where(g => g.Name.StartsWith(groupName))
            .Include(g => g.UserGroups)
            .ToListAsync(cancellationToken);
```

```
        return groups;
    }
}

using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;
using Newtonsoft.Json;
using System.Text;

namespace NearMessage.Infrastructure.Repository;

public class MessageRepository : IMessageRepository
{
    private readonly string _filePath;

    public MessageRepository(string filePath)
    {
        _filePath = filePath;
    }

    public async Task<Result<IEnumerable<Media>>> GetMessagesAsync(Guid chatId,
        CancellationToken cancellationToken)
    {
        var directoryPath = Path.Combine(_filePath, chatId.ToString());
        var fileNames = Directory.GetFiles(directoryPath);

        List<Media> messages = new();

        foreach (var fileName in fileNames)
        {
            var json = await File.ReadAllTextAsync(fileName, cancellationToken);
            var message = JsonConvert.DeserializeObject<Media>(json);
            if (message == null) continue;

            messages.Add(message);
        }

        return Result.Success<IEnumerable<Media>>(messages);
    }

    public async Task<Maybe<IEnumerable<Media>>> GetLastMessagesAsync(Guid chatId,
        DateTime lastMessageDate, CancellationToken cancellationToken)
    {
        var directoryPath = Path.Combine(_filePath, chatId.ToString());
        var lastModified = Directory.GetLastWriteTime(directoryPath);

        if (lastModified <= lastMessageDate) return Maybe<IEnumerable<Media>>.None;

        var files = Directory.GetFiles(directoryPath);
        var messages = new List<Media>();
```

```
        foreach (var file in files)
        {
            var creationTime = File.GetCreationTime(file);

            if (creationTime > lastMessageDate)
            {
                var json = await File.ReadAllTextAsync(file, cancellationToken);
                var message = JsonConvert.DeserializeObject<Media>(json);
                if (message == null) continue;

                messages.Add(message);
            }
        }

        return Maybe<IEnumerable<Media>>.From(messages);
    }

    public async Task<Result> SaveMessageAsync(Media message, CancellationToken
cancellationToken)
    {
        var directoryPath = Path.Combine(_filePath, message.ReceiverChatId.ToString());
        Directory.CreateDirectory(directoryPath);

        var json = JsonConvert.SerializeObject(message);

        var encoding = Encoding.UTF8;

        await File.WriteAllTextAsync(Path.Combine(directoryPath, $"{message.Id}.json"), json,
            encoding, cancellationToken);

        return Result.Success();
    }
}

using NearMessage.Application.Abstraction;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.UserGroups;

namespace NearMessage.Infrastructure.Repository;

public class UserGroupRepository : IUserGroupRepository
{
    private readonly INearMessageDbContext _context;

    public UserGroupRepository(INearMessageDbContext context)
    {
        _context = context;
    }
```

```
    public async Task<Result<UserGroup>> AddUserGroupAsync(UserGroup userGroup,
CancellationToken cancellationToken)
    {
        await _context.UserGroups.AddAsync(userGroup, cancellationToken);
        await _context.SaveChangesAsync(cancellationToken);

        return Result.Success(userGroup);
    }
}

using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;
using NearMessage.Domain.UsersInformation;
using Newtonsoft.Json;
using System.Text;

namespace NearMessage.Infrastructure.Repository;

public class UserInformationRepository : IUserInformationRepository
{
    private readonly string _filePath;

    public UserInformationRepository(string filePath)
    {
        _filePath = filePath;
    }

    public async Task<Result> SaveUserInformationAsync(UserInformation userInformation, Guid
id,
        CancellationToken cancellationToken)
    {
        Directory.CreateDirectory(_filePath);

        var json = JsonConvert.SerializeObject(userInformation);
        var encoding = Encoding.UTF8;

        await File.WriteAllTextAsync(Path.Combine(_filePath, $"{id}.json"), json,
            encoding, cancellationToken);

        return Result.Success();
    }
}

using Microsoft.EntityFrameworkCore;
using NearMessage.Application.Abstraction;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Contacts;
using NearMessage.Domain.Users;
```

```
namespace NearMessage.Infrastructure.Repository;

public class UserRepository : IUserRepository
{
   private readonly INearMessageDbContext _context;

   public UserRepository(INearMessageDbContext nearMessageDbContext)
   {
      _context = nearMessageDbContext;
   }

   public async Task<Result<Contact>> ConvertToContactAsync(User user, Guid sender,
      CancellationToken cancellationToken)
   {
      var chat = await _context.Chats
         .FirstOrDefaultAsync(c =>
               c.SenderId == sender && c.ReceiverId == user.Id,
            cancellationToken);

      if (chat == null) return Result.Failure<Contact>(new Error($"Chat for contact {user.Id}
doesn't exist"));

      return Result.Success(new Contact(
         user.Id,
         user.Username,
         chat.ChatId));

      throw new NotImplementedException();
   }

   public async Task<Result<User>> CreateUserAsync(User user, CancellationToken
cancellationToken)
   {
      await _context.Users.AddAsync(user, cancellationToken);
      await _context.SaveChangesAsync(cancellationToken);

      return Result.Success(user);
   }

   public async Task<Maybe<User>> GetByIdAsync(Guid id, CancellationToken
cancellationToken)
   {
      var user = await _context.Users.SingleOrDefaultAsync(i => i.Id == id, cancellationToken);

      return user == null ? Maybe<User>.None : Maybe<User>.From(user);
   }

   public async Task<Maybe<User>> GetByUsernameAsync(string username, CancellationToken
cancellationToken)
   {
```

```
        return await _context.Users.SingleOrDefaultAsync(i => i.Username == username,
cancellationToken);
    }

    public async Task<IEnumerable<User>> GetUsersByUsernameAsync(string username,
CancellationToken cancellationToken)
    {
        var users = await _context.Users
            .AsNoTracking()
            .Where(u => u.Username.StartsWith(username))
            .Include(u => u.SentChats)
            .ToListAsync(cancellationToken);

        return users;
    }
}
```

```
using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Chats;
using NearMessage.Domain.Contacts;

namespace NearMessage.Application.Chats.Commands.CreateChat;

public sealed record CreateChatCommand(
    Contact Contact,
    HttpContext HttpContext) : ICommand<Result<Chat>>;
```

```
using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Chats;

namespace NearMessage.Application.Chats.Commands.CreateChat;

public sealed class CreateChatCommandHandler : ICommandHandler<CreateChatCommand,
Result<Chat>>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IChatRepository _chatRepository;

    public CreateChatCommandHandler(IJwtProvider jwtProvider, IChatRepository chatRepository)
    {
        _jwtProvider = jwtProvider;
        _chatRepository = chatRepository;
    }

    public async Task<Result<Chat>> Handle(CreateChatCommand request, CancellationToken
cancellationToken)
```

```
    {
        var maybeUserId = _jwtProvider.GetUserId(request.HttpContext.User);
        if (maybeUserId.HasNoValue) return Result.Failure<Chat>(new Error("Can't find sender
identifier"));

        var maybeChat = await _chatRepository.CreateChatAsync(
            maybeUserId.Value, request.Contact.Id, cancellationToken);

        return maybeChat;
    }
}

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Chats;
using NearMessage.Domain.Contacts;
using NearMessage.Domain.Users;

namespace NearMessage.Application.Chats.Commands.GetChat;

public sealed record GetChatCommand(
    Contact Contact,
    HttpContext HttpContext) : ICommand<Result<Chat>>;

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Maybe;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Chats;
using NearMessage.Domain.Users;

namespace NearMessage.Application.Chats.Commands.GetChat;

public sealed class GetChatCommandHandler : ICommandHandler<GetChatCommand,
Result<Chat>>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IChatRepository _chatRepository;

    public GetChatCommandHandler(IChatRepository chatRepository, IJwtProvider jwtProvider)
    {
        _chatRepository = chatRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result<Chat>> Handle(GetChatCommand request, CancellationToken
cancellationToken)
    {
```

```
        var maybeUserId = _jwtProvider.GetUserId(request.HttpContext.User);
        if (maybeUserId.HasNoValue) return Result.Failure<Chat>(new Error("Can't find sender
identifier"));

        var maybeChat = await _chatRepository.GetChatByUsersIdAsync(
            maybeUserId.Value, request.Contact.Id, cancellationToken);

        if (maybeChat.HasNoValue) return Result.Failure<Chat>(new Error("Can't find chat"));

        return Result.Success(maybeChat.Value);
    }
}

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;

namespace NearMessage.Application.Groups.Commands.CreateGroup;

public sealed record CreateGroupCommand(
    String Name,
    HttpContext HttpContext) : ICommand<Result<Domain.Groups.Group>>;

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Groups;
using NearMessage.Domain.Messages;
using NearMessage.Domain.UserGroups;

namespace NearMessage.Application.Groups.Commands.CreateGroup;

public sealed record CreateGroupCommandHandler
    : ICommandHandler<CreateGroupCommand, Result<Domain.Groups.Group>>
{
    private readonly IGroupRepository _groupRepository;
    private readonly IUserGroupRepository _userGroupRepository;
    private readonly IJwtProvider _jwtProvider;
    private readonly IMessageRepository _messageRepository;

    public CreateGroupCommandHandler(IGroupRepository groupRepository,
        IUserGroupRepository userGroupRepository, IJwtProvider jwtProvider, IMessageRepository
messageRepository)
    {
        _groupRepository = groupRepository;
        _userGroupRepository = userGroupRepository;
        _jwtProvider = jwtProvider;
        _messageRepository = messageRepository;
    }
```

```
    public async Task<Result<Domain.Groups.Group>> Handle(CreateGroupCommand request,
        CancellationToken cancellationToken)
    {
        var maybeUserId = _jwtProvider.GetUserId(request.HttpContext.User);
        if (maybeUserId.HasNoValue)
            return Result.Failure<Domain.Groups.Group>(
                new Error("Can't find sender identifier"));

        var group = new Domain.Groups.Group(
            Guid.NewGuid(),
            request.Name + " • Group");

        var groupResult = await _groupRepository.CreateGroupAsync(group, cancellationToken);

        if (groupResult.IsFailure)
            return Result.Failure<Domain.Groups.Group>(groupResult.Error);

        var userGroup = new UserGroup(
            maybeUserId.Value,
            group.Id);

        var userGroupResult = await _userGroupRepository.AddUserGroupAsync(userGroup,
cancellationToken);

        if (userGroupResult.IsFailure)
            Result.Failure<Domain.Groups.Group>(userGroupResult.Error);

        var message = new Media(
            Guid.NewGuid(),
            $"Created new group {request.Name}",
            Guid.Empty,
            group.Id);

        var result = await _messageRepository.SaveMessageAsync(
            message,
            cancellationToken);

        if (result.IsFailure)
            Result.Failure<Domain.Groups.Group>(result.Error);

        return groupResult;
    }
}

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;

namespace NearMessage.Application.Messages.Commands.SaveMessage;
```

```
public sealed record class SaveMessageCommand(
    Media Media,
    HttpContext HttpContext) : ICommand<Result>;

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Chats;
using NearMessage.Domain.Messages;

namespace NearMessage.Application.Messages.Commands.SaveMessage;

public sealed class SaveMessageCommandHandler : ICommandHandler<SaveMessageCommand,
Result>
{
    private readonly IChatRepository _chatRepository;
    private readonly IJwtProvider _jwtProvider;
    private readonly IMessageRepository _messageRepository;

    public SaveMessageCommandHandler(IMessageRepository messageRepository,
        IChatRepository chatRepository, IJwtProvider jwtProvider)
    {
        _messageRepository = messageRepository;
        _chatRepository = chatRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result> Handle(SaveMessageCommand request, CancellationToken
cancellationToken)
    {
        var maybeSenderId = _jwtProvider.GetUserId(request.HttpContext.User);
        if (maybeSenderId.HasNoValue) return Result.Failure(new Error("Can't find sender
identifier"));

        request.Media.Sender = maybeSenderId.Value;

        var result = await _messageRepository.SaveMessageAsync(
            request.Media,
            cancellationToken);

        return result;
    }
}

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;

namespace NearMessage.Application.Messages.Queries.GetLastMessages;

public sealed record class GetLastMessagesQuery(
```

```
    DateTime LastResponseTime,
    HttpContext HttpContext) : IQuery<LastMessagesResponse>;

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;
using NearMessage.Domain.Users;

namespace NearMessage.Application.Messages.Queries.GetLastMessages;

public sealed class GetLastMessagesQueryHandler : IQueryHandler<GetLastMessagesQuery,
LastMessagesResponse>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IMessageRepository _messageRepository;
    private readonly IUserRepository _userRepository;

    public GetLastMessagesQueryHandler(IJwtProvider jwtProvider,
        IUserRepository userRepository, IMessageRepository messageRepository)
    {
        _jwtProvider = jwtProvider;
        _userRepository = userRepository;
        _messageRepository = messageRepository;
    }

    public async Task<LastMessagesResponse> Handle(GetLastMessagesQuery request,
CancellationToken cancellationToken)
    {
        var maybeUserId = _jwtProvider.GetUserId(request.HttpContext.User);

        if (maybeUserId.HasNoValue)
            return new LastMessagesResponse(Result.Failure<IEnumerable<Media>>
                (new Error("User don't recognized")));

        var maybeUser = await _userRepository.GetByIdAsync(maybeUserId.Value,
cancellationToken);

        if (maybeUser.HasNoValue)
            return new LastMessagesResponse(Result.Failure<IEnumerable<Media>>
                (new Error("User doesn't exist")));

        var receivedChats = maybeUser.Value.ReceivedChats;
        var messages = new List<Media>();

        if (receivedChats == null)
            return new LastMessagesResponse(Result.Success<IEnumerable<Media>>(messages));

        foreach (var receivedChat in receivedChats)
        {
```

```
            var lastMessages = await _messageRepository.GetLastMessagesAsync(
                receivedChat.ChatId, request.LastResponseTime, cancellationToken);

            if (lastMessages.HasNoValue)
                continue;

            messages.AddRange(lastMessages.Value);
        }

        var receivedGroups = maybeUser.Value.UserGroups;

        if (receivedGroups == null)
            return new LastMessagesResponse(Result.Success<IEnumerable<Media>>(messages));

        foreach (var receivedGroup in receivedGroups)
        {
            var lastMessages = await _messageRepository.GetLastMessagesAsync(
                receivedGroup.GroupId, request.LastResponseTime, cancellationToken);

            if (lastMessages.HasNoValue)
                continue;

            messages.AddRange(lastMessages.Value);
        }

        return new LastMessagesResponse(Result.Success<IEnumerable<Media>>(messages));
    }
}

using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;

namespace NearMessage.Application.Messages.Queries.GetLastMessages;

public sealed record class LastMessagesResponse(Result<IEnumerable<Media>> Messages);

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Domain.Contacts;

namespace NearMessage.Application.Messages.Queries.GetMessages;

public sealed record GetMessagesQuery(
    Contact Sender) : IQuery<MessagesResponse>;

using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;

namespace NearMessage.Application.Messages.Queries.GetMessages;
```

```
public class GetMessagesQueryHandler : IQueryHandler<GetMessagesQuery, MessagesResponse>
{
   private readonly IMessageRepository _messageRepository;

   public GetMessagesQueryHandler(IMessageRepository messageRepository)
   {
      _messageRepository = messageRepository;
   }

   public async Task<MessagesResponse> Handle(GetMessagesQuery request,
      CancellationToken cancellationToken)
   {
      if (!request.Sender.ChatId.HasValue)
         return new MessagesResponse(Result.Failure<IEnumerable<Media>>(
            new Error("Chat not exist")));

      return new MessagesResponse(await _messageRepository.GetMessagesAsync(
         request.Sender.ChatId.Value, cancellationToken));
   }
}

using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Messages;

namespace NearMessage.Application.Messages.Queries.GetMessages;

public sealed record MessagesResponse(Result<IEnumerable<Media>> Messages);

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.UserGroups;

namespace NearMessage.Application.UserGroups.Commands.CreateUserGroup;

public sealed record CreateUserGroupCommand(
   Guid GroupId,
   HttpContext HttpContext) : ICommand<Result<Guid>>;

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.UserGroups;

namespace NearMessage.Application.UserGroups.Commands.CreateUserGroup;

public sealed class CreateUserGroupCommandHandler
: ICommandHandler<CreateUserGroupCommand, Result<Guid>>
{
```

```
    private readonly IUserGroupRepository _userGroupRepository;
    private readonly IJwtProvider _jwtProvider;

    public CreateUserGroupCommandHandler(IUserGroupRepository userGroupRepository,
IJwtProvider jwtProvider)
    {
        _userGroupRepository = userGroupRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result<Guid>> Handle(CreateUserGroupCommand request,
CancellationToken cancellationToken)
    {
        var maybeUserId = _jwtProvider.GetUserId(request.HttpContext.User);
        if (maybeUserId.HasNoValue)
            return Result.Failure<Guid>(
                new Error("Can't find sender identifier"));

        var userGroup = new UserGroup(
            maybeUserId.Value,
            request.GroupId);

        var userGroupResult = await _userGroupRepository.AddUserGroupAsync(userGroup,
cancellationToken);

        if (userGroupResult.IsFailure)
            Result.Failure<UserGroup>(userGroupResult.Error);

        return Result.Success(userGroupResult.Value.GroupId);
    }
}

using NearMessage.Common.Primitives.Result;

namespace NearMessage.Application.Users.Commands.UserAuthentication;

public sealed record AuthenticationResponse(string Token, Guid Id);

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Users;

namespace NearMessage.Application.Users.Commands.UserAuthentication;

public sealed class UserAuthenticationAsyncCommandHandler
    : ICommandHandler<UserAuthenticationCommand, Result<AuthenticationResponse>>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IUserRepository _userRepository;
```

```
    public UserAuthenticationAsyncCommandHandler(IUserRepository userRepository,
        IJwtProvider jwtProvider)
    {
        _userRepository = userRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result<AuthenticationResponse>> Handle(UserAuthenticationCommand
request,
        CancellationToken cancellationToken)
    {
        var maybeUser = await _userRepository.GetByUsernameAsync(request.Username,
cancellationToken);

        if (maybeUser.HasNoValue)
            return Result.Failure<AuthenticationResponse>
                (new Error("The user with the specified user name was not found."));

        var user = maybeUser.Value;

        if (!user.VerifyPassword(request.Password))
            return Result.Failure<AuthenticationResponse>(new Error("Password wasn't match"));

        var token = _jwtProvider.Generate(user);

        return Result.Success(new AuthenticationResponse(token, user.Id));
    }
}

using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;

namespace NearMessage.Application.Users.Commands.UserAuthentication;

public sealed record UserAuthenticationCommand(
    string Username,
    string Password) : ICommand<Result<AuthenticationResponse>>;

using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Users;

namespace NearMessage.Application.Users.Commands.UserRegistration;

public sealed class UserRegistrationAsyncCommandHandler
    : ICommandHandler<UserRegistrationCommand, Result<string>>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IUserRepository _userRepository;
```

```
    public UserRegistrationAsyncCommandHandler(IUserRepository userRepository,
        IJwtProvider jwtProvider)
    {
        _userRepository = userRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<Result<string>> Handle(UserRegistrationCommand request,
        CancellationToken cancellationToken)
    {
        var id = Guid.NewGuid();

        var user = new User(
            id,
            request.Username,
            request.Password);

        var result = await _userRepository.CreateUserAsync(user, cancellationToken);

        if (result.IsFailure) return Result.Failure<string>(result.Error);

        var token = _jwtProvider.Generate(user);

        return Result.Success(token);
    }
}
```

```
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;

namespace NearMessage.Application.Users.Commands.UserRegistration;

public sealed record UserRegistrationCommand(
    string Username,
    string Password) : ICommand<Result<string>>;
```

```
using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;

namespace NearMessage.Application.Users.Queries.GetAllUsers;

public sealed record class GetAllUsersQuery(
    HttpContext HttpContext) : IQuery<UsersResponse>;
```

```
using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Contacts;
using NearMessage.Domain.Users;
```

```
namespace NearMessage.Application.Users.Queries.GetAllUsers;

public sealed class GetAllUsersQueryHandler : IQueryHandler<GetAllUsersQuery,
UsersResponse>
{
    private readonly IJwtProvider _jwtProvider;
    private readonly IUserRepository _userRepository;

    public GetAllUsersQueryHandler(IUserRepository userRepository, IJwtProvider jwtProvider)
    {
        _userRepository = userRepository;
        _jwtProvider = jwtProvider;
    }

    public async Task<UsersResponse> Handle(GetAllUsersQuery request, CancellationToken
cancellationToken)
    {
        var maybeSenderId = _jwtProvider.GetUserId(request.HttpContext.User);

        if (maybeSenderId.HasNoValue)
            return new UsersResponse(
                Result.Failure<IEnumerable<Contact>?>(new Error("Used don't recognized")));

        var maybeSender = await _userRepository.GetByIdAsync(maybeSenderId.Value,
cancellationToken);

        if (maybeSender.HasNoValue)
            return new UsersResponse(
                Result.Failure<IEnumerable<Contact>?>(new Error("Used doesn't exist")));

        var contacts = maybeSender.Value.SentChats
            ?.Select(c =>
            {
                if (c.Receiver != null)
                    return new Contact(
                        c.Receiver.Id,
                        c.Receiver.Username,
                        c.ChatId);
                return null;
            })
            .ToList();

        var groupContacts = maybeSender.Value.UserGroups
            ?.Select(ug =>
            {
                if (ug.Group != null)
                    return new Contact(
                        ug.GroupId,
                        ug.Group.Name,
                        ug.GroupId);
```

```
                return null;
            })
            .ToList();

        if (groupContacts != null) contacts?.AddRange(groupContacts);

        return new UsersResponse(Result.Success<IEnumerable<Contact>?>(contacts));
    }
}
```

```
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Contacts;

namespace NearMessage.Application.Users.Queries.GetAllUsers;

public sealed record UsersResponse(Result<IEnumerable<Contact>?> Contacts);
```

```
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Contacts;

namespace NearMessage.Application.Users.Queries.SearchUser;

public sealed record SearchedUserResponse(Result<IEnumerable<Contact>> SearchedUsers);
```

```
using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;

namespace NearMessage.Application.Users.Queries.SearchUser;

public sealed record SearchUserQuery(
    string Username,
    HttpContext HttpContext) : IQuery<SearchedUserResponse>;
```

```
using NearMessage.Application.Abstraction;
using NearMessage.Application.Users.Queries.GetAllUsers;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.Contacts;
using NearMessage.Domain.Groups;
using NearMessage.Domain.Users;

namespace NearMessage.Application.Users.Queries.SearchUser;

public sealed class SearchUserQueryHandler : IQueryHandler<SearchUserQuery,
SearchedUserResponse>
{
    private readonly IUserRepository _userRepository;
    private readonly IJwtProvider _jwtProvider;
    private readonly IGroupRepository _groupRepository;
```

```
    public SearchUserQueryHandler(IUserRepository userRepository, IJwtProvider jwtProvider,
        IGroupRepository groupRepository)
    {
        _userRepository = userRepository;
        _jwtProvider = jwtProvider;
        _groupRepository = groupRepository;
    }

    public async Task<SearchedUserResponse> Handle(SearchUserQuery request,
CancellationToken cancellationToken)
    {
        var maybeSenderId = _jwtProvider.GetUserId(request.HttpContext.User);

        if (maybeSenderId.HasNoValue)
            return new SearchedUserResponse(
                Result.Failure<IEnumerable<Contact>>(new Error("Used don't recognized")));

        var users = await _userRepository.GetUsersByUsernameAsync(request.Username,
cancellationToken);


        var contacts = users.Select(u =>
                new Contact(
                    u.Id,
                    u.Username,
                    u.SentChats?.Find(
                            c => c.ReceiverId == maybeSenderId.Value)?
                        .ChatId))
            .ToList();

        var groups = await _groupRepository.GetGroupsByUsernameAsync(request.Username,
cancellationToken);

        var groupContacts = groups.Select(g =>
                {
                    var isGroupId = g.UserGroups?.Any(ug =>
                        ug.UserId.Equals(maybeSenderId.Value)) ?? false;

                    var groupId = isGroupId ? g.Id : Guid.Empty;

                    return new Contact(
                        groupId,
                        g.Name,
                        g.Id);
                }
            )
            .ToList();

        contacts.AddRange(groupContacts);

        return new SearchedUserResponse(Result.Success<IEnumerable<Contact>>(contacts));
```

```
    }
}

using Microsoft.AspNetCore.Http;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.UsersInformation;

namespace NearMessage.Application.UsersInformation.Command.SaveUserInformation;

public sealed record SaveUserInformationCommand(
    UserInformation UsersInformation,
    HttpContext HttpContext) : ICommand<Result>;
using NearMessage.Application.Abstraction;
using NearMessage.Common.Abstractions.Messaging;
using NearMessage.Common.Primitives.Errors;
using NearMessage.Common.Primitives.Result;
using NearMessage.Domain.UsersInformation;

namespace NearMessage.Application.UsersInformation.Command.SaveUserInformation;
public sealed class SaveUserInformationCommandHandler
    : ICommandHandler<SaveUserInformationCommand, Result>
{
private readonly IJwtProvider _jwtProvider;
private readonly IUserInformationRepository _userInformationRepository;

public SaveUserInformationCommandHandler(IJwtProvider jwtProvider,
    IUserInformationRepository userInformationRepository)
{
    _jwtProvider = jwtProvider;
    _userInformationRepository = userInformationRepository;
}

public async Task<Result> Handle(SaveUserInformationCommand request, CancellationToken
cancellationToken)
    {
        var maybeUserId = _jwtProvider.GetUserId(request.HttpContext.User);
        if (maybeUserId.HasNoValue)
            return Result.Failure(
                new Error("Can't find sender identifier"));

        var result = await _userInformationRepository
            .SaveUserInformationAsync(request.UsersInformation, maybeUserId.Value,
cancellationToken);

        if (result.IsFailure)
            return Result.Failure(result.Error);

        return result;
    }
}
```

Приложение Б
(обязательное)
Формы выходных документов

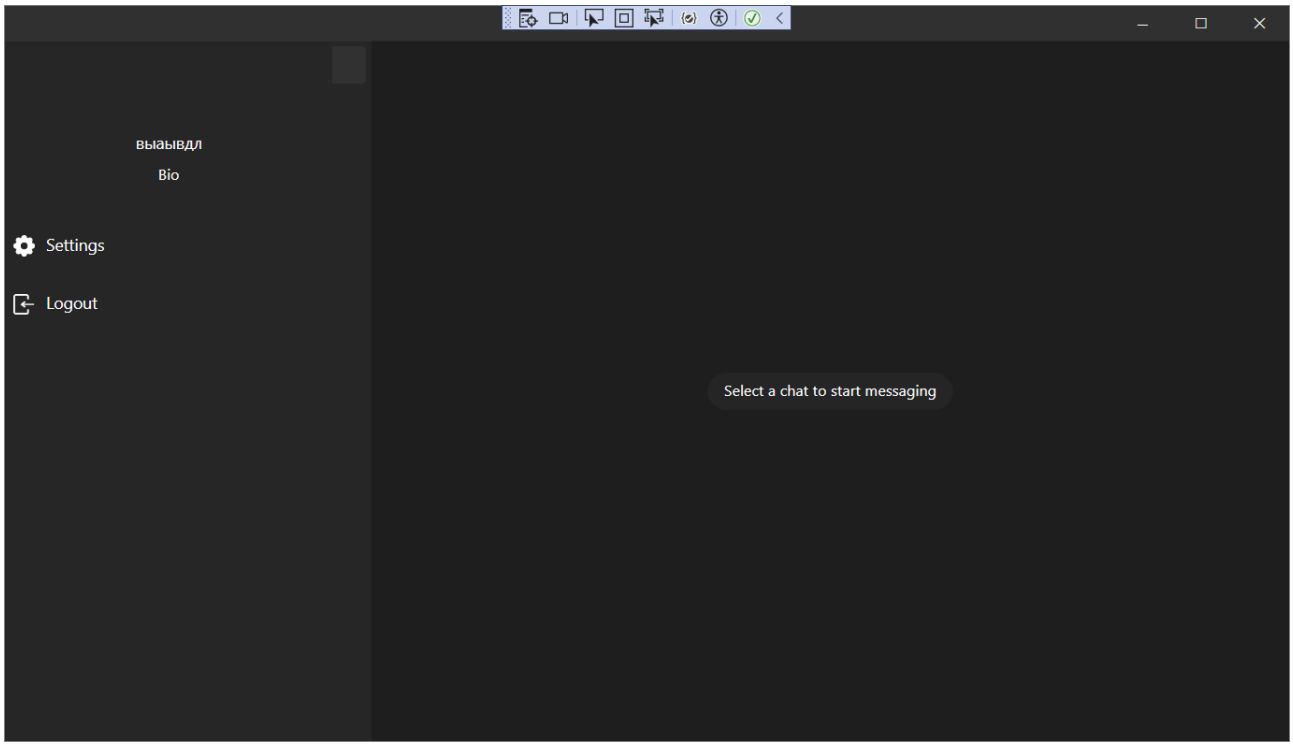

Рисунок Б.1 – Регистрация пользователя

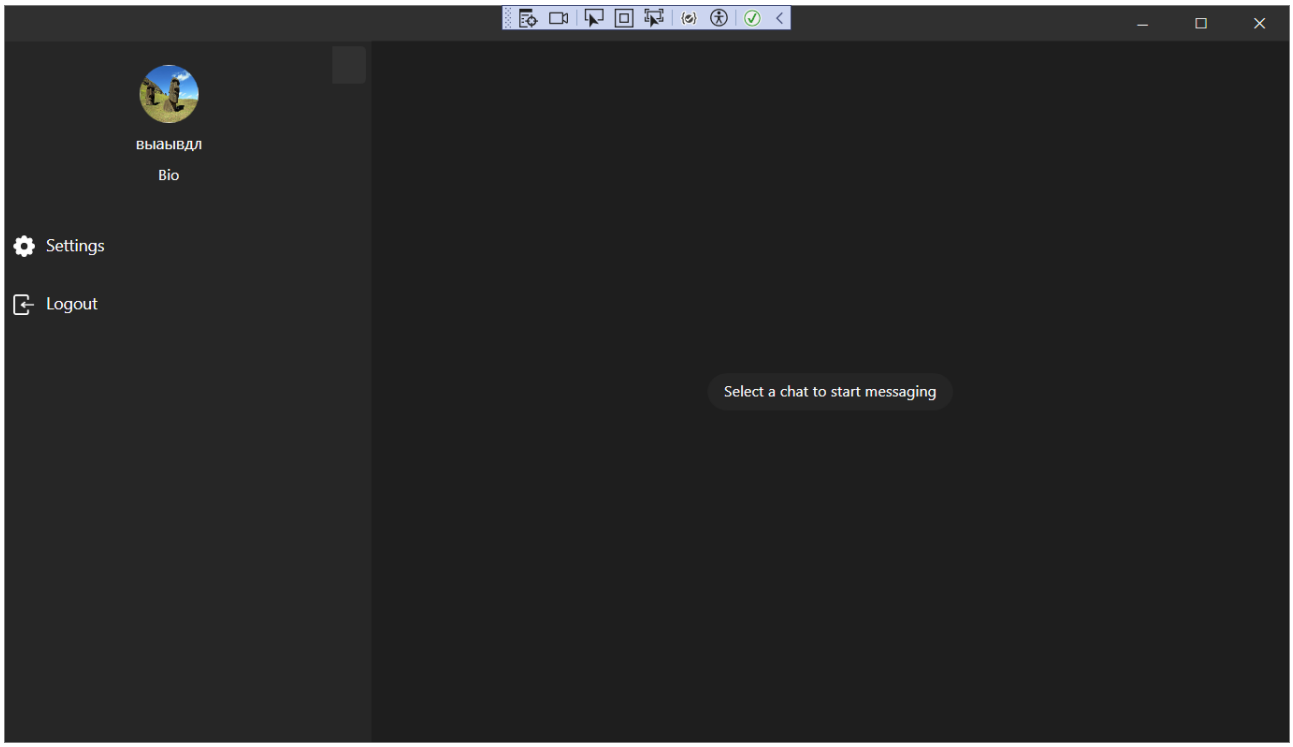

Рисунок Б.2 – Изменение аватарки

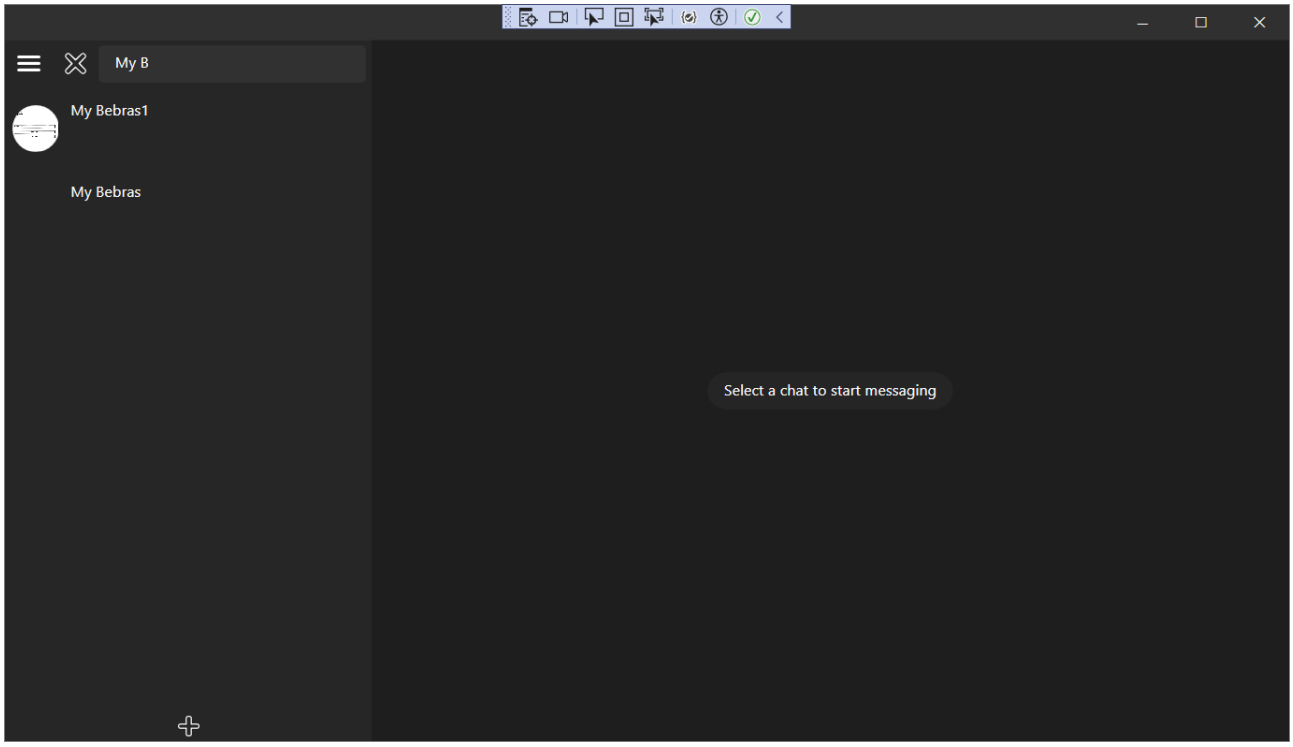

Рисунок Б.3 –

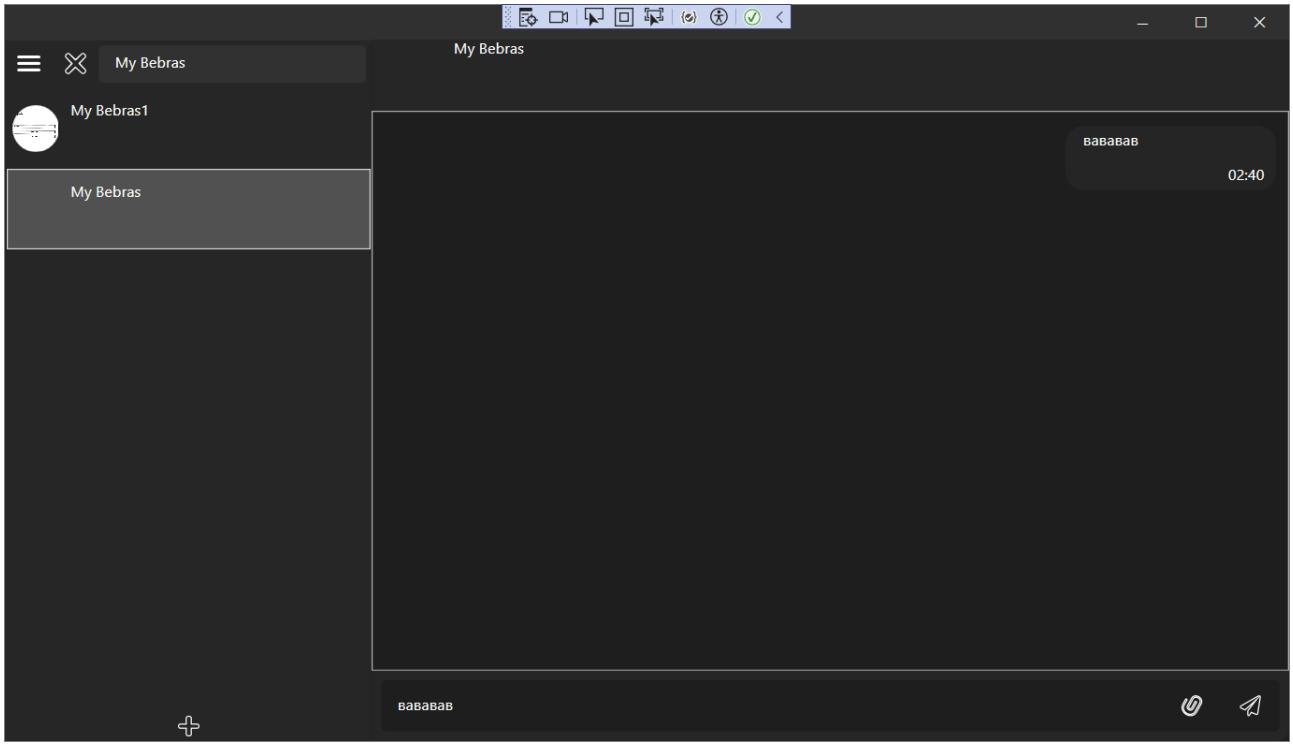

Рисунок Б.4 – Отправка сообщения

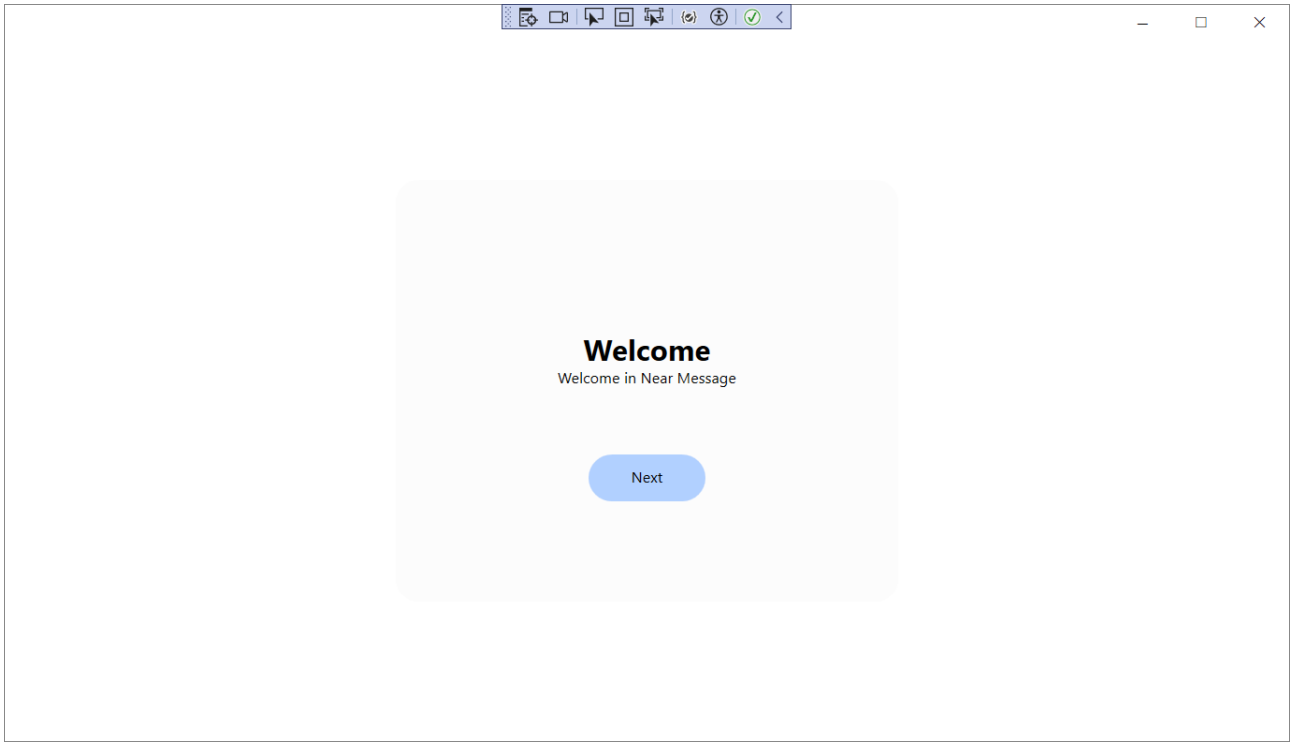

Рисунок Б.5 – Смена темы

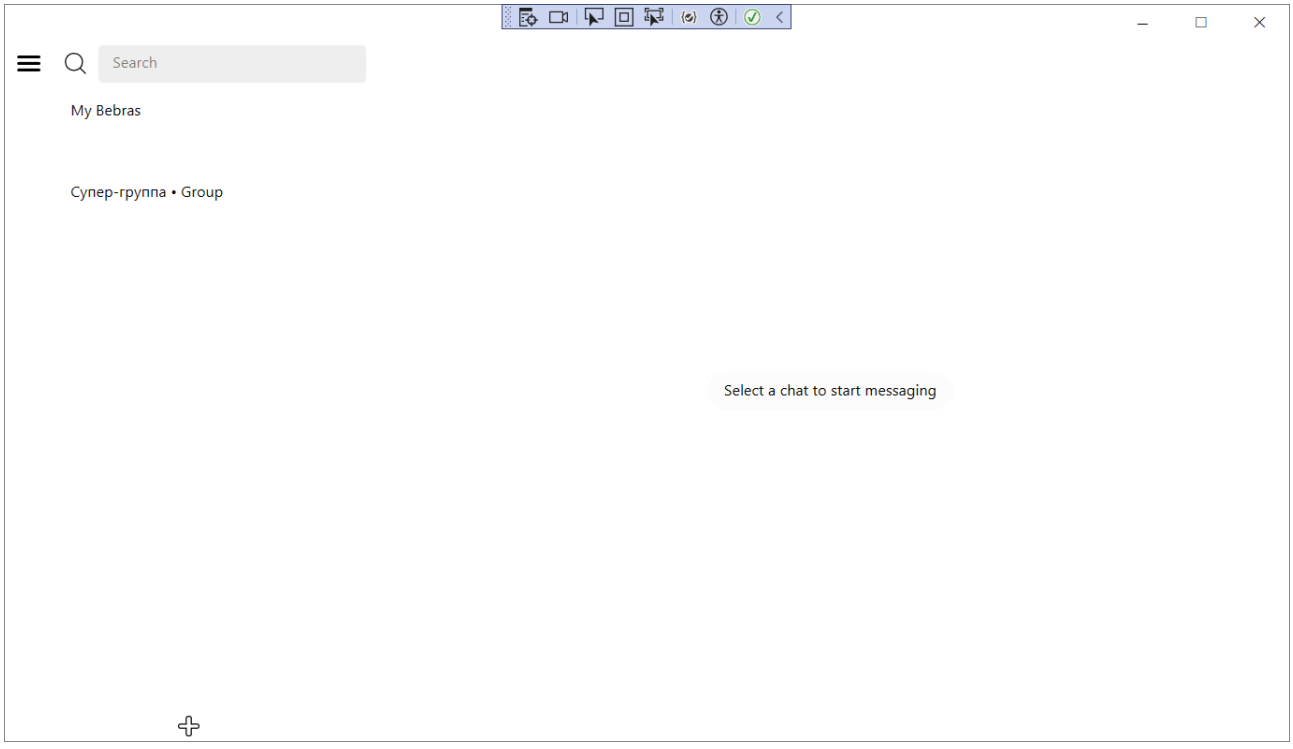

Рисунок Б.6 – Создание группы

Приложение В
(обязательное)
Формы выходных документов