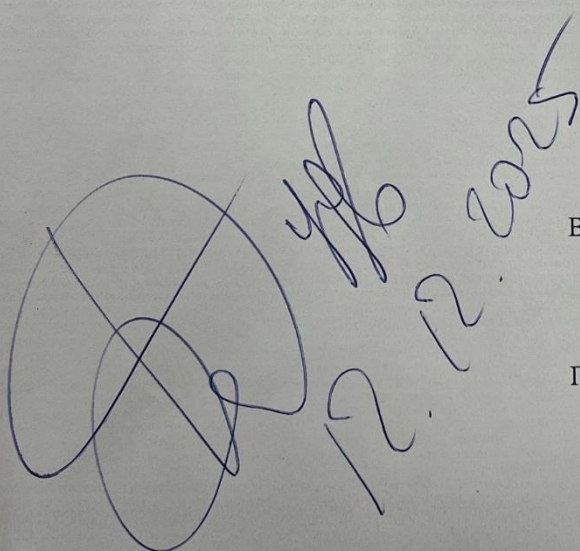


Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе
по курсу «Архитектура операционных систем»
на тему «Система программ обмена сообщениями»

A large, stylized handwritten signature in blue ink, followed by the date "12.12.2024" written vertically.

Выполнил:

студен группы 22BVB1

Ивлюшин М.Д.

Принял:

Федюнин Р.Н.

Пенза 2024

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

М.А. Митрохин

«___» _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Архитектура операционных систем
Студенту Иванову Максиму Дмитриевичу Группа 22 ВВВ-1
Тема проекта Система обработки обмена сообщениями

Исходные данные (технические требования) на проектирование

Разработать систему обработки клиент-серверной архитектуры взаимодействующих компонентов в виде роуд-договорного протокола передачи данных. Функциональные требования к системе: идентификация пользователей на сервере, обработка клиентских запросов, передача данных между клиентом и сервером, передача данных между серверами. Организация взаимодействия между клиентами (передача сообщений, файлов, объектов, передача данных).

Функции клиента: Графический пользовательский интерфейс, взаимодействие с сервером.

Функциональные требования к серверу: С, С++, С#

Дополнительные требования к системе: аутентификация пользователей, передача мультимедийной информации, хранение статистических данных на сервере, вхождение протоколов работы, нормативных СТУД для хранения информации о пользователях.

Объем работы по курсу

1. Расчетная часть

1. Разработка протокола взаимодействия программ
2. Разработка программы
3. Отладка программы и протоколов

2. Графическая часть

Не менее 5 UML диаграмм с описанием работы системы

3. Экспериментальная часть

Тестирование системы программ в локальной внешней
тливной сети

Срок выполнения проекта по разделам

- 1 В соответствии с графиком выполнения работы
- 2
- 3
- 4
- 5
- 6
- 7
- 8

Дата выдачи задания «__»

Дата защиты проекта «__»

Руководитель

Задание получил «__»

20__ г.

Студент

Иванов И.И.

Содержание

Введение	3
Основная часть.....	4
1. Постановка задачи	4
2. Теоритические сведения.....	5
3. Выбор решения	6
4. Описание Протокола	7
5. Описание программы	10
6. Руководство пользователя.....	16
Заключение	20
Список используемых источников	22
Приложение А.....	23
Листинг программы клиента	23
Файл ResponseCodes.cs	23
Файл RequestCodes.cs.....	23
Файл messeng.cs	25
Файл client.cs.....	32
Файл Programm.cs.....	39
Файл Registr.cs.....	40
Файл Avtorisation.cs	42
Приложение Б.....	44
Листинг программы сервера	44
Файл Programm.cs.....	44
Файл ResponseCodes.cs	58
Файл RequestCodes.cs.....	59
Приложение В.....	60

Введение

Современное общество трудно представить без технологий обмена данными и эффективной коммуникации. В эпоху стремительного развития информационных технологий мессенджеры занимают важное место среди средств общения, предлагая пользователям возможность обмениваться сообщениями, файлами и мультимедиа в режиме реального времени.

Данная курсовая работа посвящена созданию программного обеспечения, позволяющего осуществлять обмен текстовыми сообщениями и файлами в локальной сети. Основная цель работы – разработка удобного и функционального мессенджера, способного удовлетворить базовые потребности пользователей в коммуникации. Приложение ориентировано на упрощение взаимодействия между пользователями и реализацию надежного способа передачи данных.

В процессе выполнения работы будет рассмотрено множество аспектов, связанных с разработкой приложения, таких как проектирование архитектуры, реализация многопользовательского доступа, обеспечение авторизации и аутентификации пользователей, а также обработка и передача файлов. Особое внимание будет уделено созданию удобного интерфейса и обеспечения безопасности передаваемой информации.

Результатом работы станет полноценное приложение, способное обеспечивать удобное и безопасное взаимодействие пользователей в локальной сети. Полученный опыт в проектировании и разработке мессенджеров может быть полезен не только для дальнейшего совершенствования программы, но и для разработки аналогичных решений в других областях.

Основная часть

1. Постановка задачи

Цель данной работы заключается в создании клиент-серверной архитектуры, предназначенной для обмена текстовыми сообщениями и файлами в рамках локальной вычислительной сети. Предполагается разработка системы, которая позволит клиентам взаимодействовать с сервером с использованием специально разработанного протокола обмена данными. Основной задачей является реализация графического интерфейса для клиентского приложения и консольного интерфейса для сервера.

Ключевым этапом является создание протокола обмена данными, обеспечивающего надежную передачу текстовых сообщений и файлов между клиентами и сервером. Графический интерфейс клиентской части должен поддерживать отправку сообщений и файлов пользователям, Консольный интерфейс сервера должен предоставлять возможности для мониторинга текущего обмена сообщениями и файлами между клиентами.

Разработка будет осуществляться на языке программирования C#, а приложение будет поддерживать работу на любой операционной системе.

2. Теоритические сведения

Программы, работающие в сети Интернет, взаимодействуют между собой с помощью специальных наборов правил и инструкций, называемых протоколами. Протокол определяет, каким образом осуществляется передача данных между устройствами, а также порядок действий для отправки и приема информации.

Одним из ключевых протоколов, обеспечивающих функционирование сети Интернет, является протокол TCP/IP. Он состоит из двух частей: TCP (Transmission Control Protocol) и IP (Internet Protocol). TCP отвечает за управление передачей данных: он разбивает информацию на пакеты, отправляет их и следит за их корректной доставкой. Также TCP выполняет контроль ошибок передачи и обеспечивает последовательную сборку данных на стороне получателя.

IP, в свою очередь, отвечает за маршрутизацию пакетов. Каждый информационный пакет содержит IP-адреса отправителя и получателя. На основе этих адресов маршрутизаторы, специальные сетевые устройства, определяют оптимальный путь доставки пакетов к месту назначения.

Для работы прикладных программ, таких как мессенджеры или почтовые клиенты, используются дополнительные протоколы, которые регулируют процессы передачи данных и авторизации. Например, для доступа к электронной почте программа может запросить пароль пользователя, что требует выполнения заранее определенной последовательности действий.

Таким образом, благодаря набору различных протоколов обеспечивается согласованное и эффективное взаимодействие программ и устройств в сети Интернет, что делает возможным удобное использование современных сетевых технологий.

3. Выбор решения

Для реализации графического интерфейса клиентской части использовалась технология WinForms. Она предоставляет удобные инструменты для разработки приложений под Windows, позволяя легко интегрировать такие элементы управления, как текстовые поля, кнопки и списки, в функциональную структуру мессенджера.

Взаимодействие между сервером и клиентами осуществляется с использованием протокола TCP/IP, который гарантирует надежную передачу данных благодаря механизму подтверждений и повторной отправки в случае ошибок. Этот протокол широко используется в приложениях, таких как HTTP, FTP, SMTP и Telnet. TCP требует предварительной установки соединения, где сервер ожидает входящих запросов, а клиент инициирует связь, отправляя синхронизирующий порядковый номер.

В системе применяется топология "звезда": каждый клиент подключается к серверу, который выступает в роли центрального узла. Для наглядности топология представлена на рисунке 1.

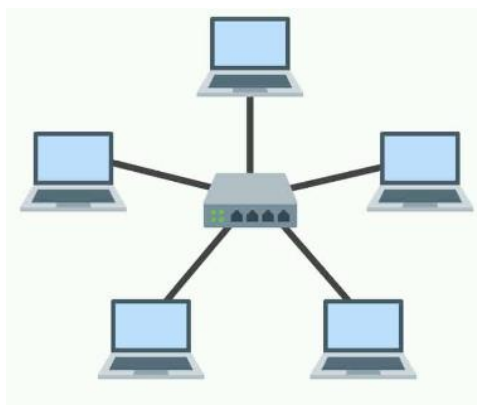


Рисунок 1 – Топология "звезда"

Для обеспечения безопасного обмена данными между клиентами и сервером был реализован механизм идентификации и авторизации пользователей. Это позволяет контролировать доступ к системе и предотвращать несанкционированное использование.

Дополнительно для повышения удобства работы с приложением были внедрены функции, такие как отображение статуса пользователей в сети.

4. Описание Протокола

Протокол представляет собой договоренность, применяемую как клиентом, так и сервером. С использованием данного протокола клиент и сервер взаимодействуют, осуществляя согласованный обмен информацией.

Протокол реализован как в клиентской части (файл client.cs), так и в серверной части (файл Program.cs). Каждая из команд, таких как: REGISTER, LOGIN, GET_USERS, SEND_MESSAGE, GET_MESSAGES, NEW_MESSAGE, SEND_FILE, RECIEVE_FILE, CLOSE представляет собой часть структуры протокола и выполняет определенную функцию в процессе обмена сообщениями между клиентом и сервером. Реализация протокола в обеих частях приложения обеспечивает стабильность в процессе взаимодействия между клиентом и сервером.

Ниже представлено описание каждой команды протокола.

Команда SEND_MESSAGE используется для отправки сообщений соответствующему пользователю. В сообщении указывается логин отправителя, логин получателя, сообщение получателю. На стороне сервера сообщение разделяется на логин отправителя, логин получателя, сообщение получателю. Если получатель онлайн, сервер сохраняет сообщение в файл истории чата и отправляет ответ в виде NEW_MESSAGE. В ответе указывается логин отправителя, сообщение получателю. Если получатель оффлайн, то просто сохраняется в файл истории чата.

Команда GET_MESSAGES позволяет пользователю получить историю переписки с другим участником. Клиент отправляет серверу запрос с указанием своего логина и логина собеседника. Сервер определяет файл истории чатов, проверяет наличие сообщений и возвращает содержимое в виде строк текста. Если файл отсутствует, клиенту отправляется сообщение об отсутствии истории сообщений. Этот механизм позволяет пользователям легко восстанавливать доступ к своим предыдущим диалогам.

Команда `SEND_FILE` позволяет передавать файлы между пользователями. Клиент отправляет серверу информацию о файле и его содержимое в виде байтов. Сервер сохраняет его как для отправителя, так и для получателя.

Команда `RECIEVE_FILE` предназначена для передачи файлов от сервера клиенту в ответ на запрос, связанный с вызовом команды `SEND_FILE`. Когда клиент отправляет запрос `SEND_FILE`, сервер сохраняет файл на сервер. Далее идет проверка на статус получателя, если он в сети, то файл ему пересылается, иначе файл остается на сервере. В дальнейшем можно будет доработать, то что отправка будет отложенной для пользователей, которые не были в сети, когда им отправляли файл.

Команда `LOGIN` обеспечивает аутентификацию пользователя. Клиент отправляет логин, пароль на сервер. Сервер проверяет данные с сохранёнными записями в *users.txt*. Если данные корректны, пользователю предоставляется доступ к месенджеру.

Команда `REGISTER` используется для регистрации нового пользователя. Клиент отправляет логин, пароль. Сервер проверяет отсутствие логина в файле *users.txt*. Если запись отсутствует, регистрация завершается успешно, и пользователю предоставляются права для входа.

Команда `GET_USERS` используется для обновления списка пользователей на клиентском приложении. Клиент отправляет запрос на сервер. Сервер возвращает список всех зарегистрированных пользователей с указанием их статуса (онлайн/офлайн). Для актуальности список обновляется каждые 5 секунды.

Команда `CLOSE` удаляет пользователя из списка активных. Она используется, когда пользователь выходит из системы.

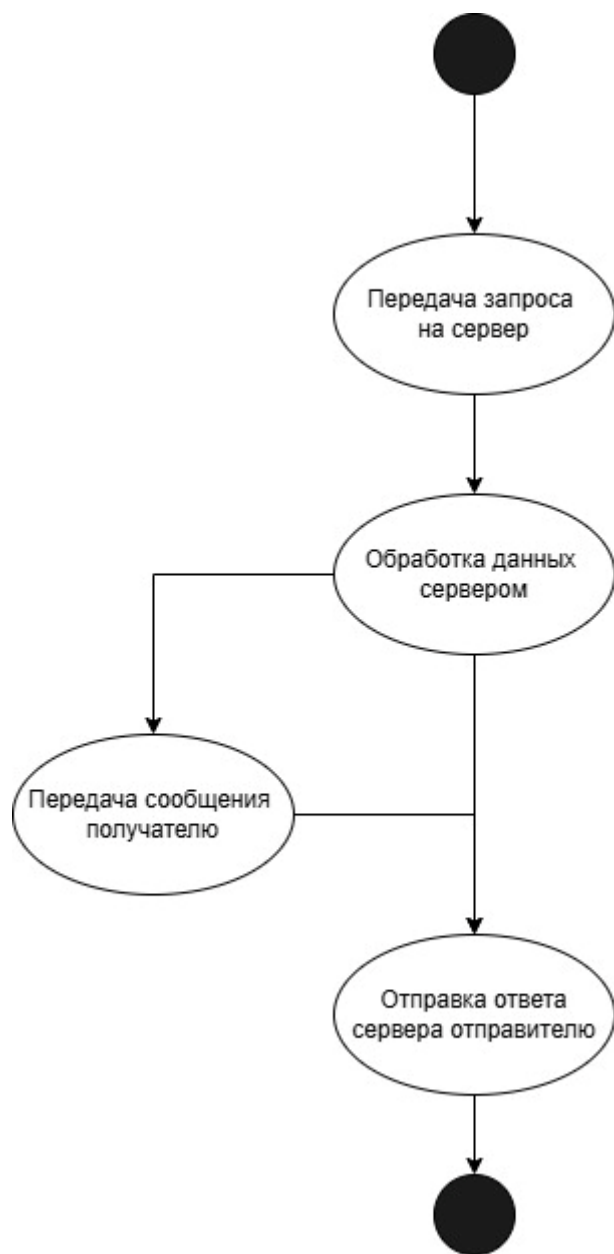


Рисунок 2 – UML Диаграмма состояний

5. Описание программы

Программа представляет собой клиент-серверное приложение, которое начинает свою работу с запуска сервера. Серверная часть программы инициирует прослушивание порта для входящих подключений. Он обрабатывает запросы от клиентов, обеспечивая корректную работу взаимодействия между ними.

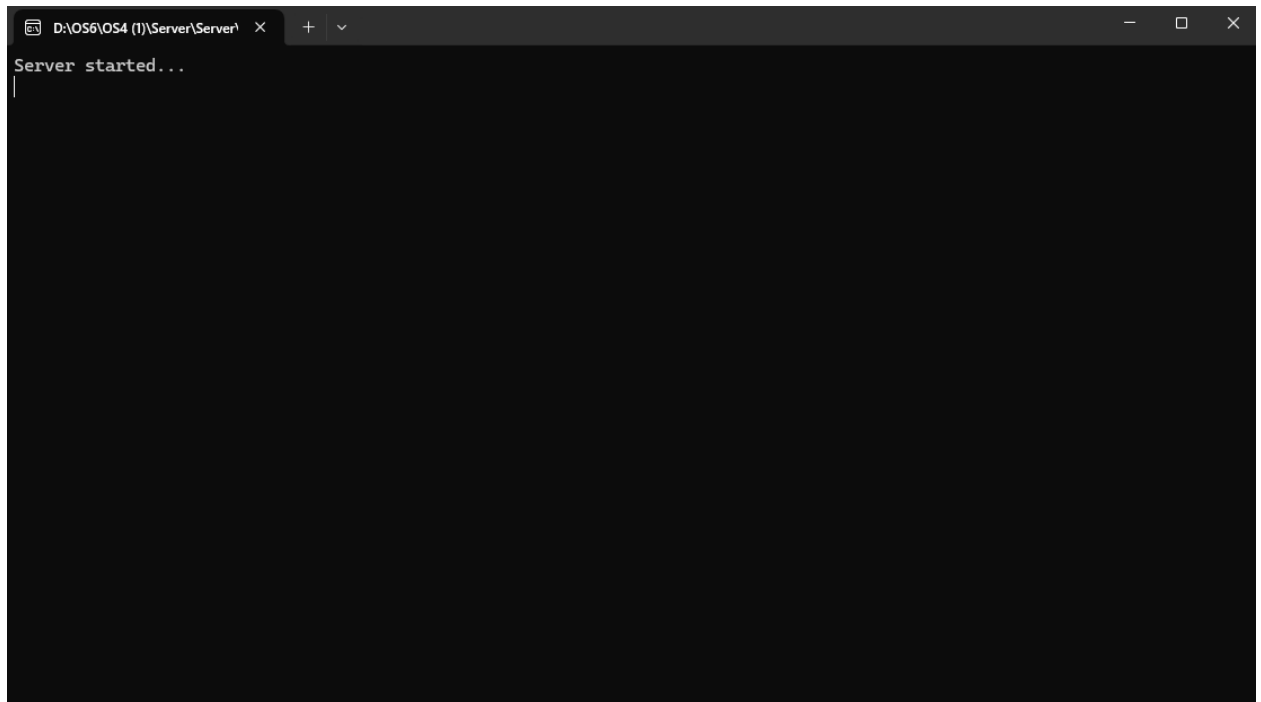


Рисунок 3 – Запуск сервера

После запуска сервера пользователь может запустить клиентскую программу, которая пытается установить соединение с сервером. При успешном подключении клиента к серверу открывается форма для авторизации. Здесь пользователь вводит свои данные для аутентификации. В случае неудачного подключения, клиент уведомит пользователя о проблемах с соединением.

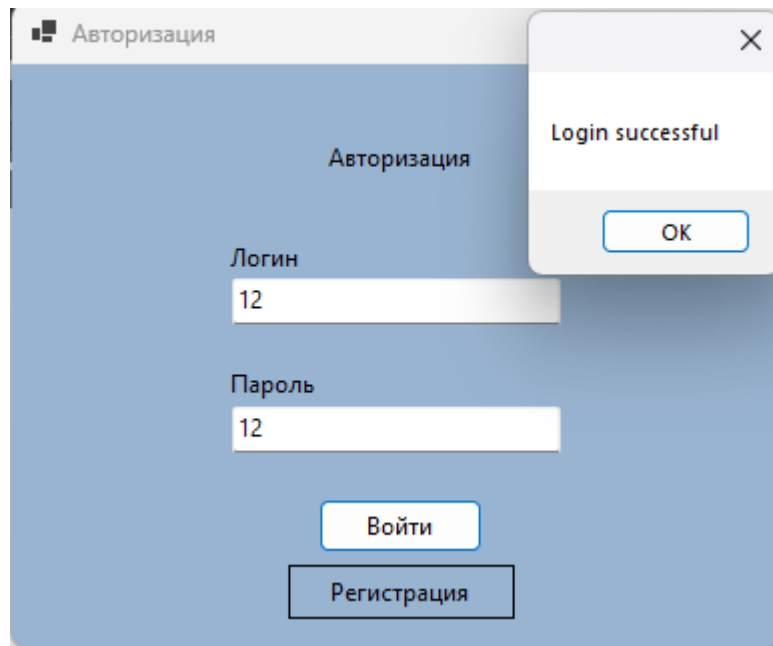


Рисунок 4 – Авторизация и аутентификация

После успешной авторизации клиент отправляет запрос на сервер, чтобы получить список пользователей, которые в данный момент зарегистрированы, а также информацию о том, кто из них онлайн. Эти данные отображаются слева в таблице мессенджера, позволяя пользователю быстро увидеть доступных для общения собеседников.

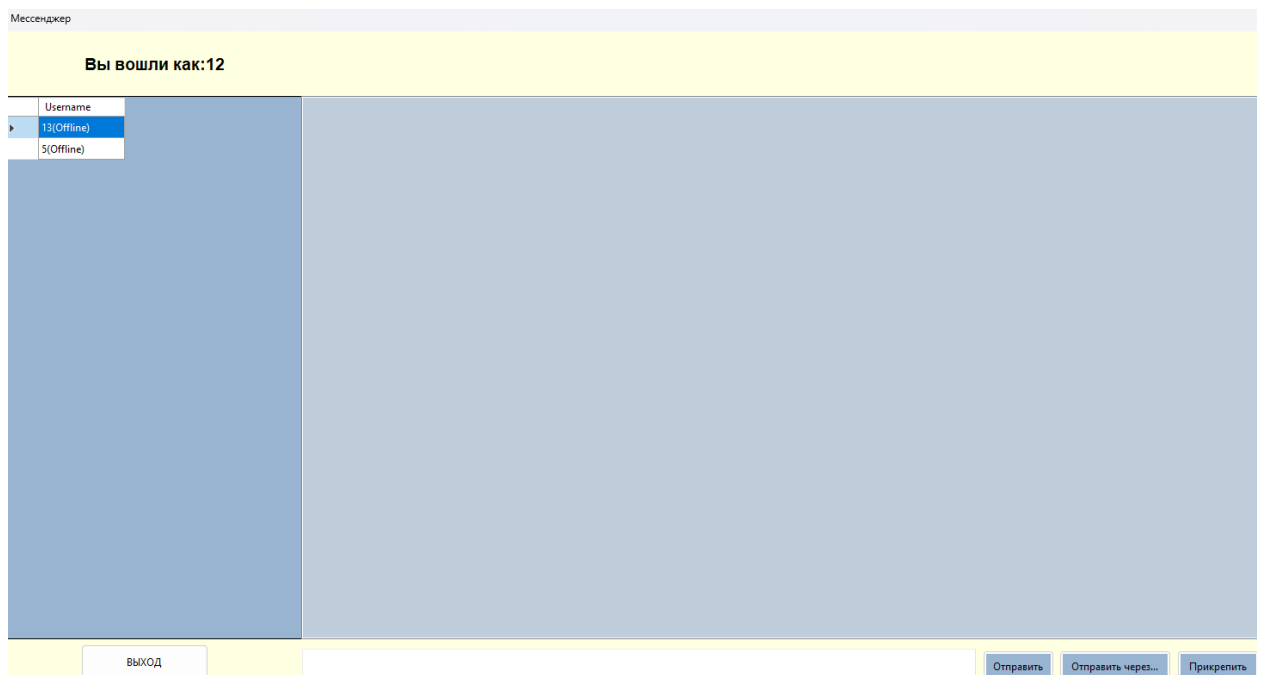


Рисунок 5 – Главное окно

Для обмена сообщениями достаточно выбрать нужного пользователя, ввести текст сообщения и нажать на кнопку «Отправить», чтобы отправить сообщение.

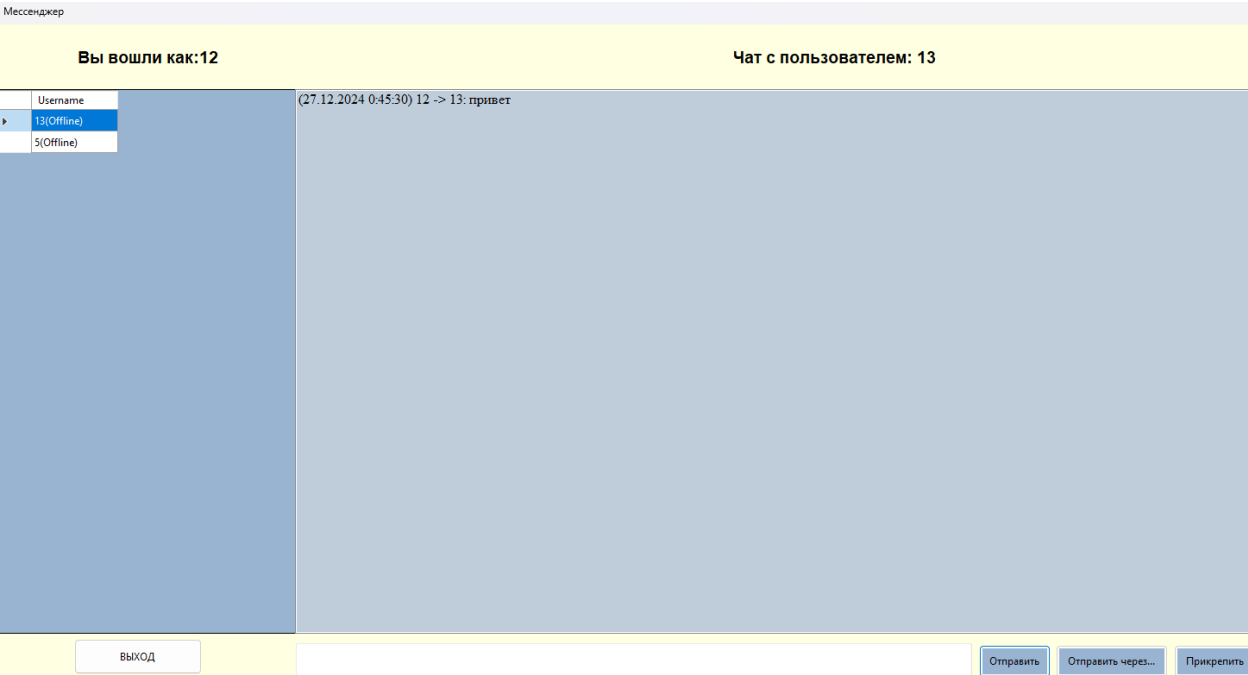


Рисунок 6 – Окно отправителя

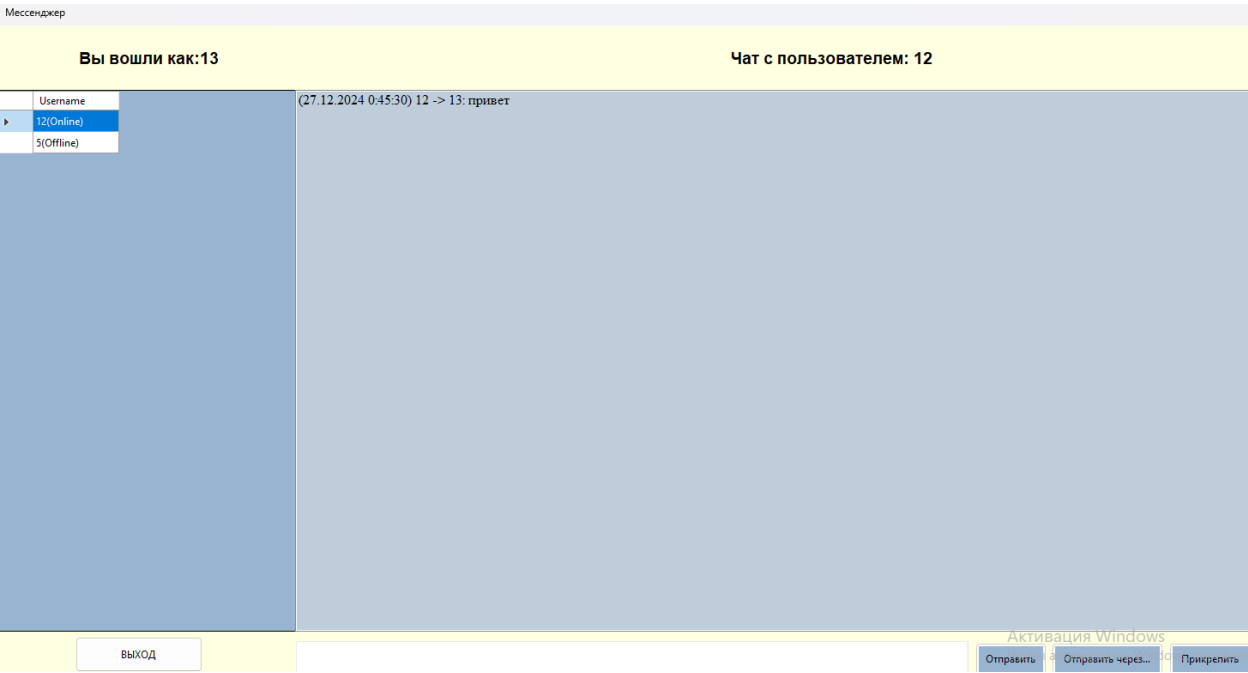


Рисунок 7 – Окно получателя

Если нужно отправить отложенное сообщение, достаточно после ввода текста кликнуть на кнопку «Отправить через». Откроется окно, где можно указать время в секундах, когда сообщение должно быть отправлено. Через 5 секунд сообщение будет автоматически отправлено.

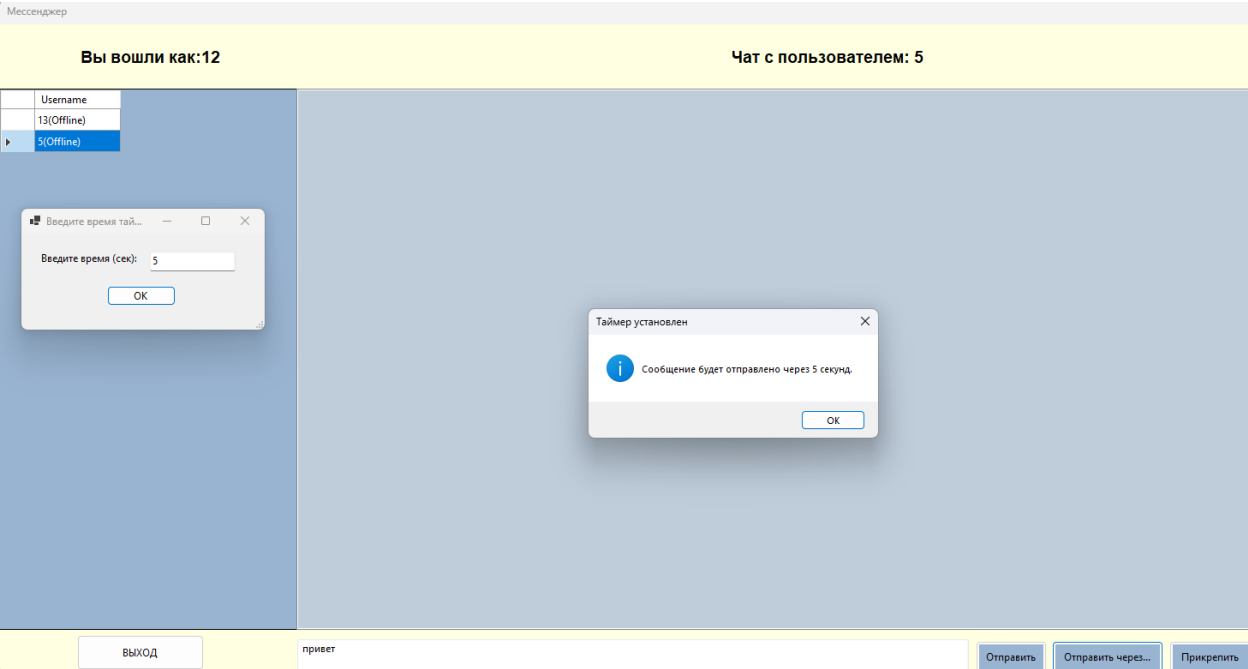


Рисунок 9 – Отправка сообщения с таймером

Для отправки файла в чат, достаточно кликнуть по кнопке «Прикрепить», выбрать нужный файл и он автоматически отправится.

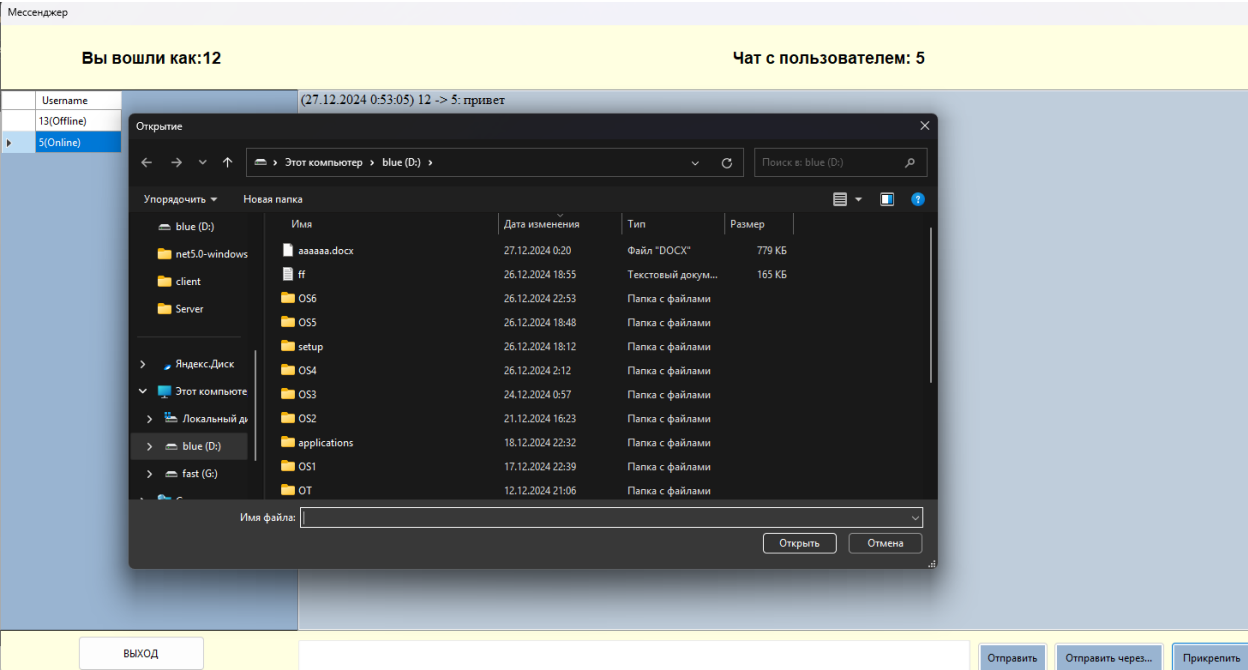


Рисунок 10 – Выбор файла для отправки

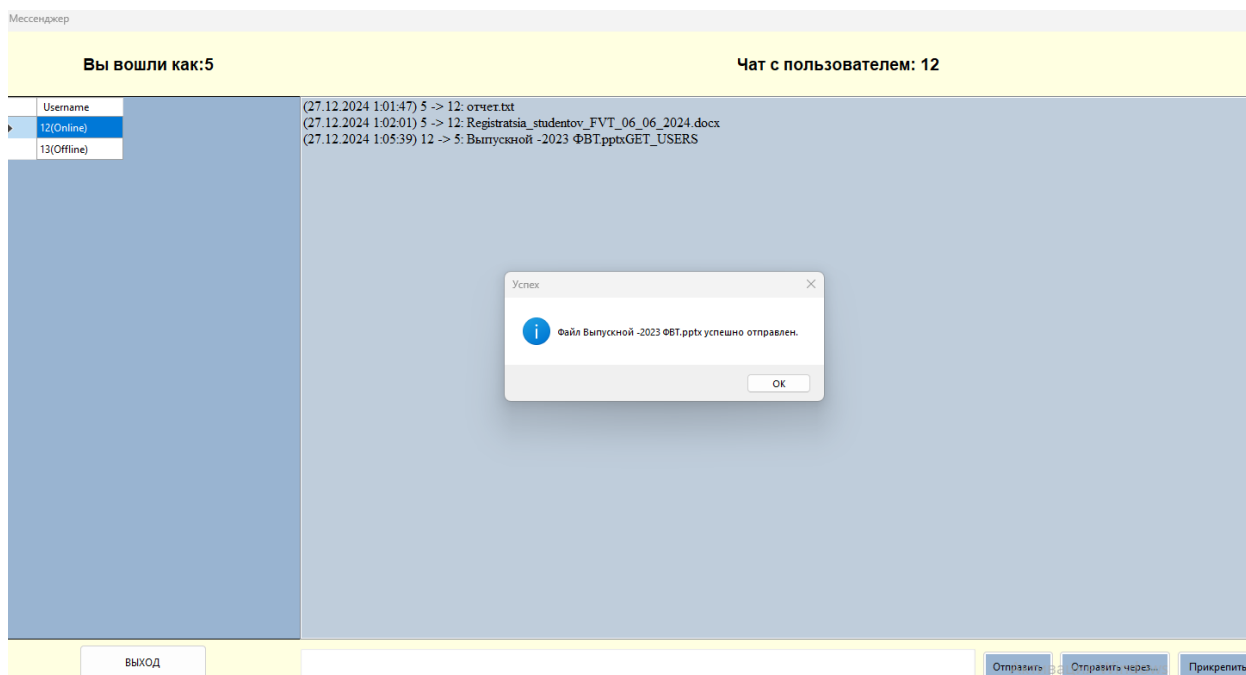


Рисунок 11 – Отправка файла

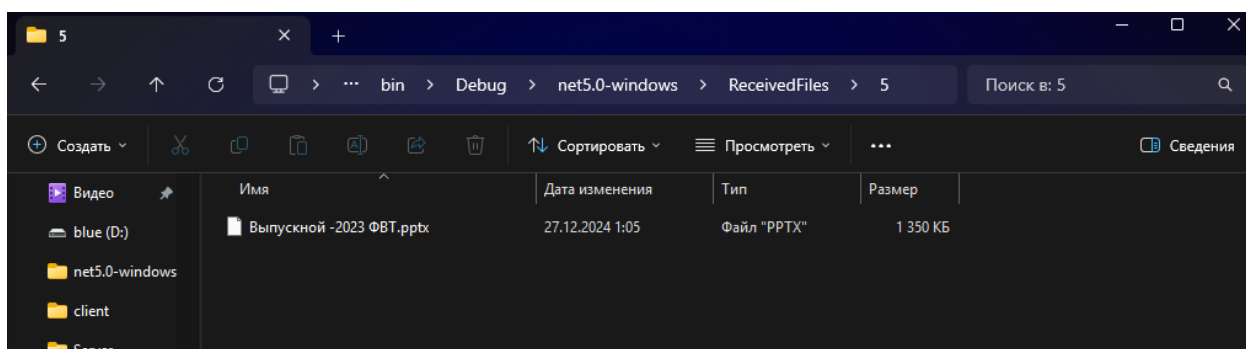


Рисунок 12 – Папка пользователя на сервере

После получения файла, получатель отправляет запрос на сервер для его скачивания.

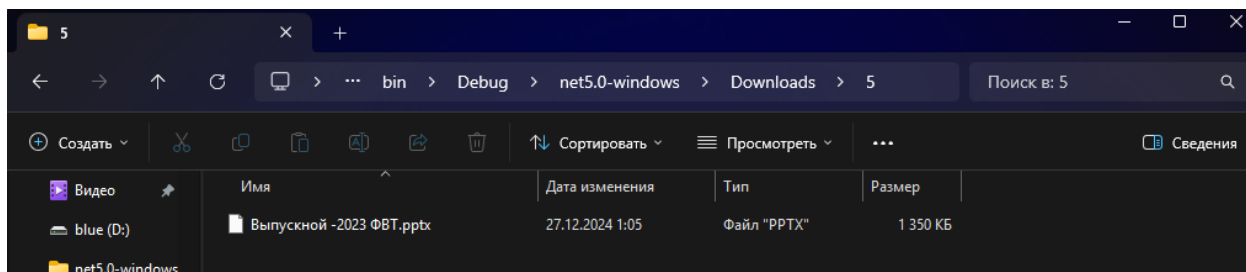


Рисунок 13 – Папка пользователя клиента

Для отображения ключевых взаимодействий между клиентской и серверной частями приложения была создана UML-диаграмма развертывания. Эта диаграмма помогает визуализировать физическую структуру системы и способы взаимодействия её компонентов. На рисунке 14 представлена данная диаграмма, где показаны как клиентская, так и серверная части приложения, а также методы их взаимодействия.

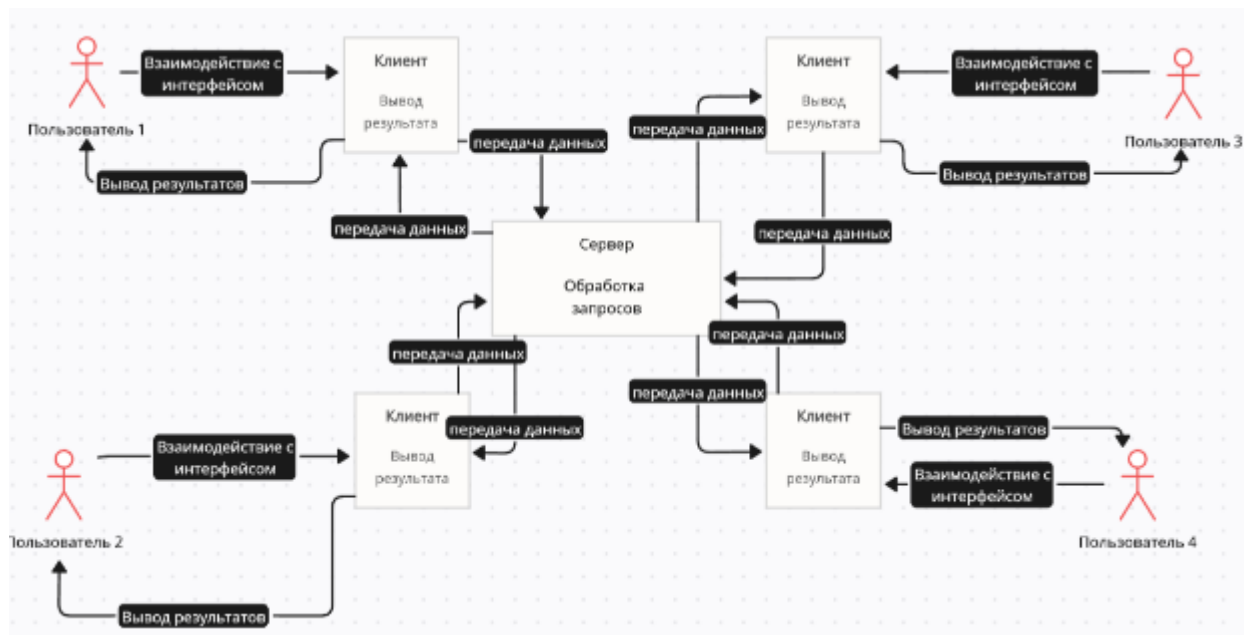


Рисунок 14 – UML - диаграмма развёртывания

В приложении **В** на рисунке 22 представлена UML-диаграмма состояний сервера, отображающая последовательность событий, происходящих на сервере в процессе обработки команд, поступающих от клиентов.

В приложении **В** на рисунке 23 представлена UML-диаграмма состояний клиента, отражающая последовательность событий, происходящих на клиенте при обработке команд пользователя и ответов, полученных от сервера.

6. Руководство пользователя

При запуске программы клиента откроется окно авторизации. Для авторизации введите ваш логин и пароль в соответствующие поля и нажмите кнопку «**Войти**». Если ваш логин и пароль верны и зарегистрированы на сервере, вы будете перенаправлены на основное окно программы. Если вы не зарегистрированы, нажмите кнопку «**Регистрация**». После этого откроется форма регистрации, где нужно ввести уникальный логин и пароль. После успешной регистрации будет выполнен вход в программу.

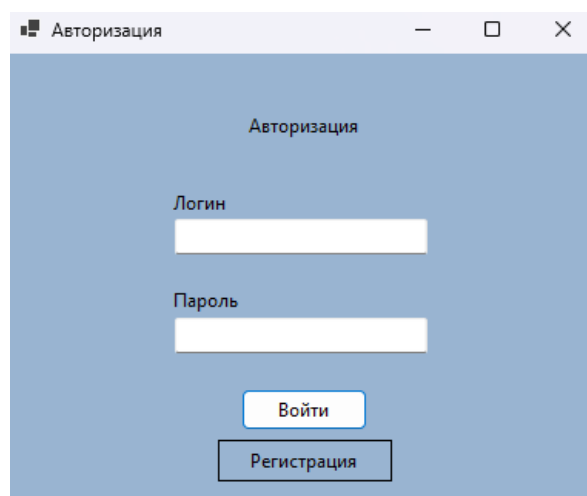


Рисунок 15 – Форма аутентификации и авторизации

После успешной авторизации откроется форма отправки сообщений. Что бы отправить сообщение нужно выбрать пользователя в таблице.

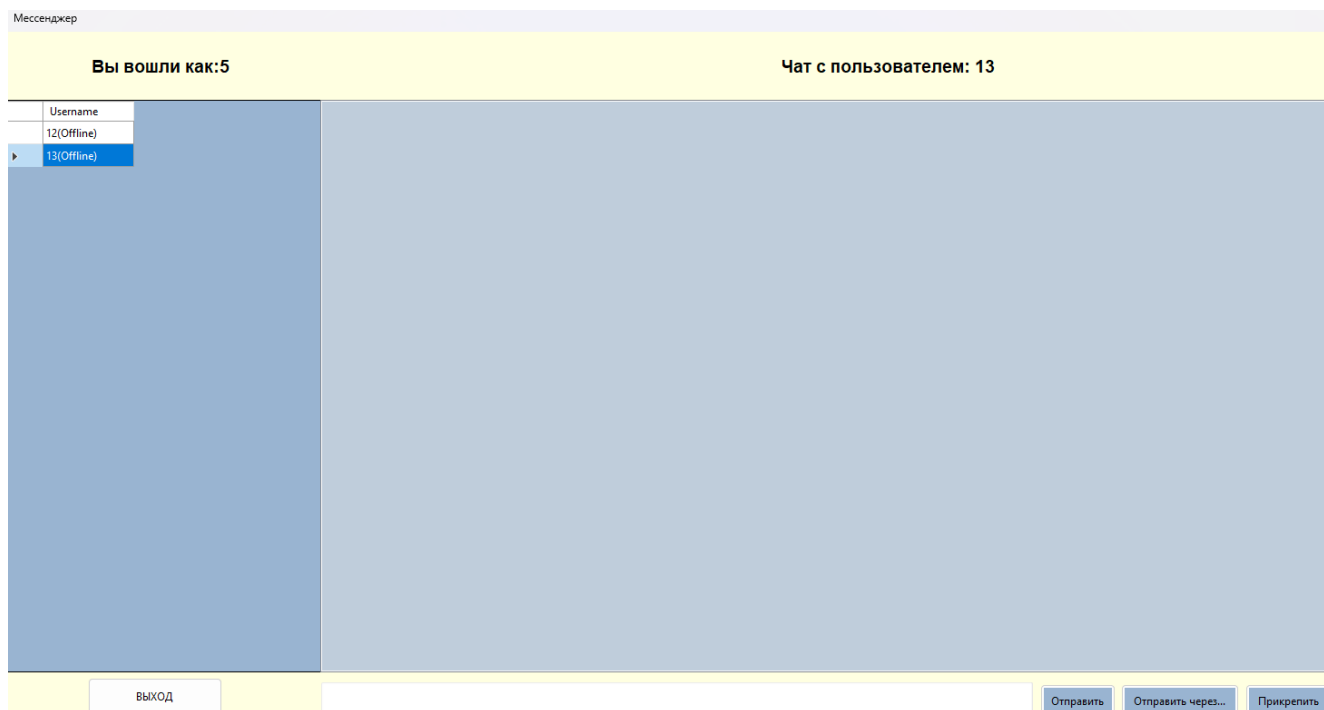


Рисунок 16 – Главное окно приложения

В поле для ввода текста сообщения напишите текст, который вы хотите отправить, также можно прикрепить файл нажав на кнопку «Прикрепить».

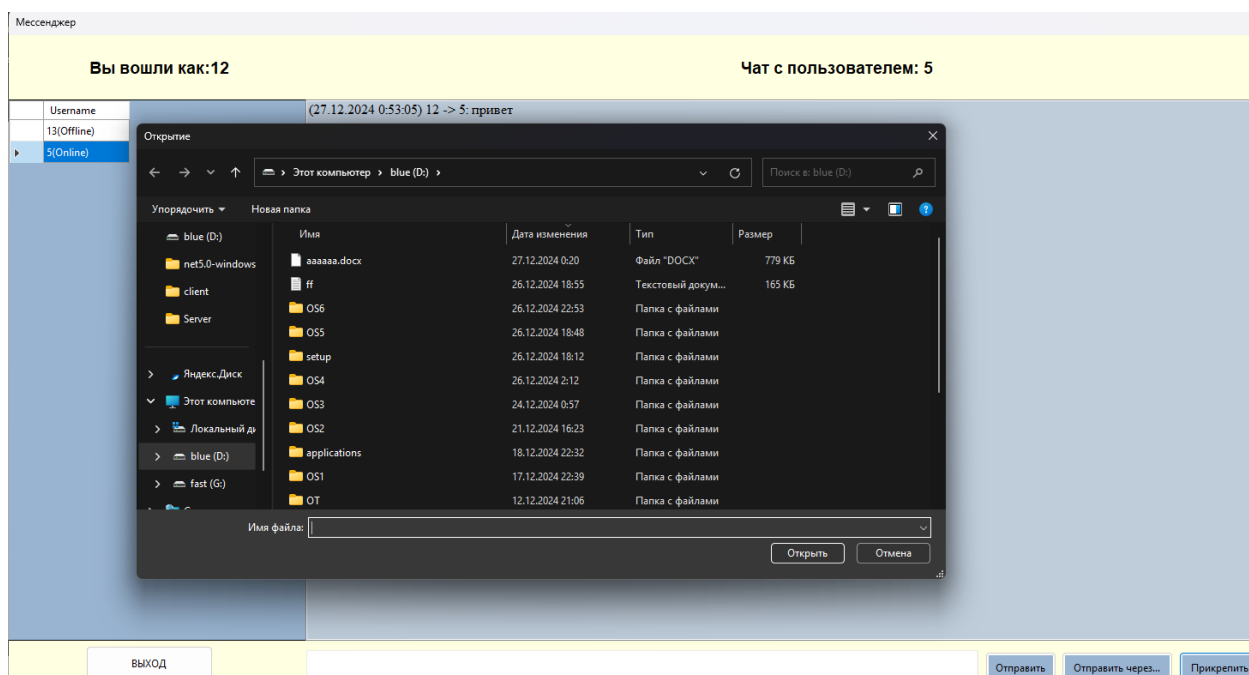


Рисунок 17 – Выбор файла

Чтобы отправить сообщение, нажмите кнопку «Отправить». Сообщение будет отправлено выбранному пользователю.

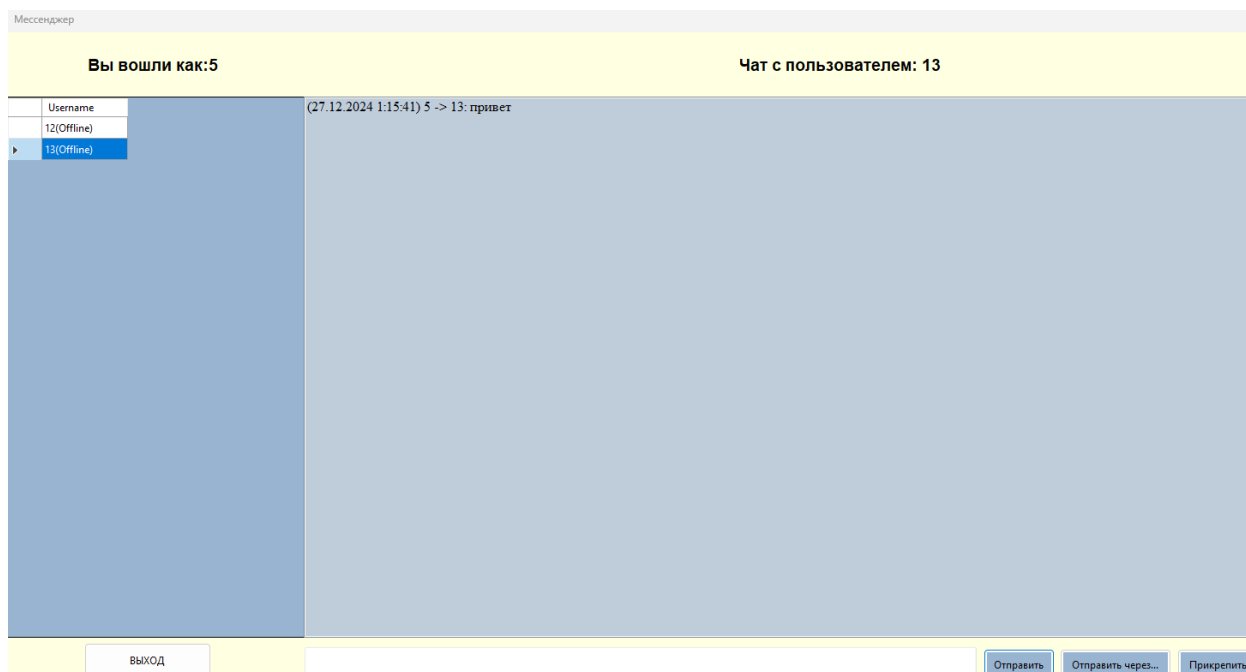


Рисунок 18 – Выбор пользователя и ввод сообщения

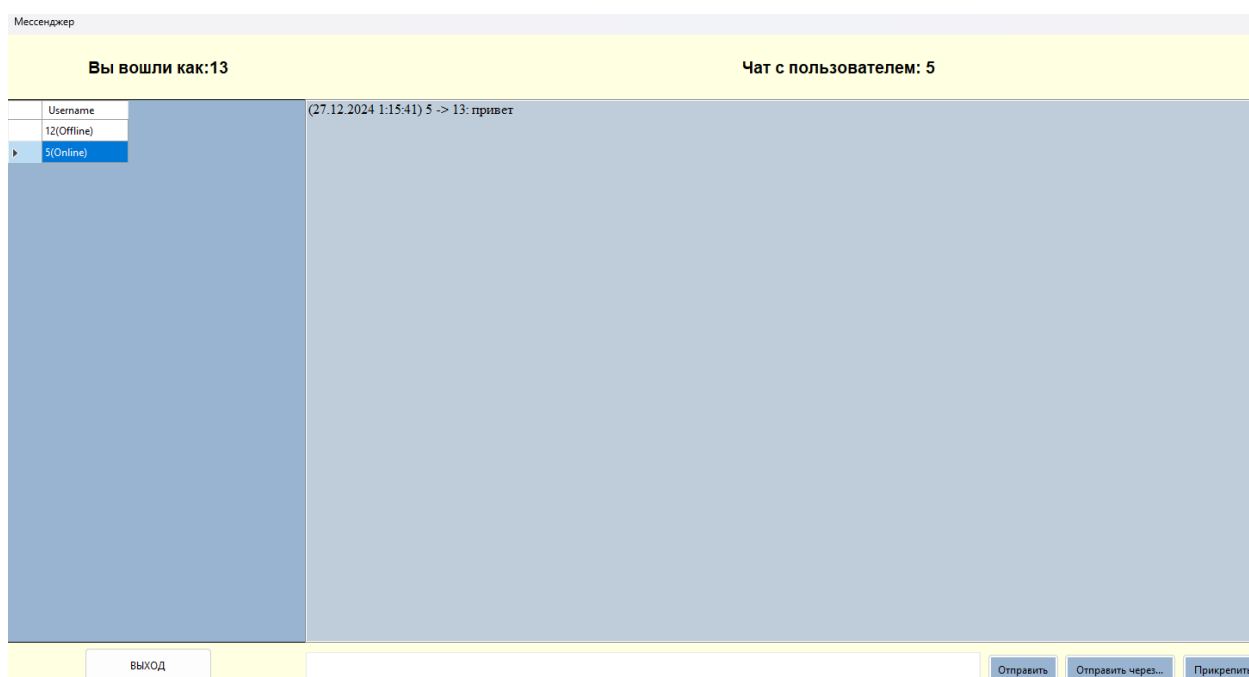


Рисунок 19 – Получение сообщения

Для запуска серверной части программы необходимо запустить исполняемый файл сервера **server.exe**. После запуска сервер начнёт работать, обрабатывая запросы пользователей, поддерживая соединение с клиентами. На рисунке 21 представлена UML-диаграмма сценариев клиента.

```

D:\OS6\OS4 (1)\Server\Server1 x + v
сообщение: System.Collections.Generic.List`1[Server.Program+Message]
[ЛОГ] Сообщений загружено: 6
[ЛОГ] Сообщение: Отправитель: 5, Получатель: 12, Сообщение: отчет.txt, Дата: 27.12.2024 1:01:47
[ЛОГ] Сообщение: Отправитель: 5, Получатель: 12, Сообщение: Registratsia_studentov_FVT_06_06_2024.docx, Дата: 27.12.2024 1:02:01
[ЛОГ] Сообщение: Отправитель: 12, Получатель: 5, Сообщение: Выпускной -2023 ФВТ.pptxGET_USERS, Дата: 27.12.2024 1:05:39
[ЛОГ] Сообщение: Отправитель: 5, Получатель: 12, Сообщение: Выпускной -2023 ФВТ.pptx, Дата: 27.12.2024 1:06:25
[ЛОГ] Сообщение: Отправитель: 12, Получатель: 5, Сообщение: Выпускной -2023 ФВТ.pptx, Дата: 27.12.2024 1:10:16
[ЛОГ] Сообщение: Отправитель: 5, Получатель: 13, Сообщение: привет, Дата: 27.12.2024 1:15:41
[ЛОГ] Количество сообщений, подходящих под критерии: 1
[ЛОГ] Сформирован ответ: GET_MESSAGES:SUCCESS:5:13:привет:27.12.2024 1:15:41
Отправляем ответ: GET_MESSAGES:SUCCESS:5:13:привет:27.12.2024 1:15:41
status: GET_MESSAGES
Подготовка к отправке ответа...
Ответ отправлен клиенту.
Получено сообщение: GET_USERS:13
Получено сообщение: GET_USERS:13
command: GET_USERS
Отправляем ответ: GET_USERS:SUCCESS:12(Offline),5(Online)
status: GET_USERS
Подготовка к отправке ответа...
Ответ отправлен клиенту.
Получено сообщение: GET_USERS:5
Получено сообщение: GET_USERS:5
command: GET_USERS
Отправляем ответ: GET_USERS:SUCCESS:12(Offline),13(Online)
status: GET_USERS
Подготовка к отправке ответа...
Ответ отправлен клиенту.
Получено сообщение: GET_USERS:13

```

Рисунок 20 – Работа сервера

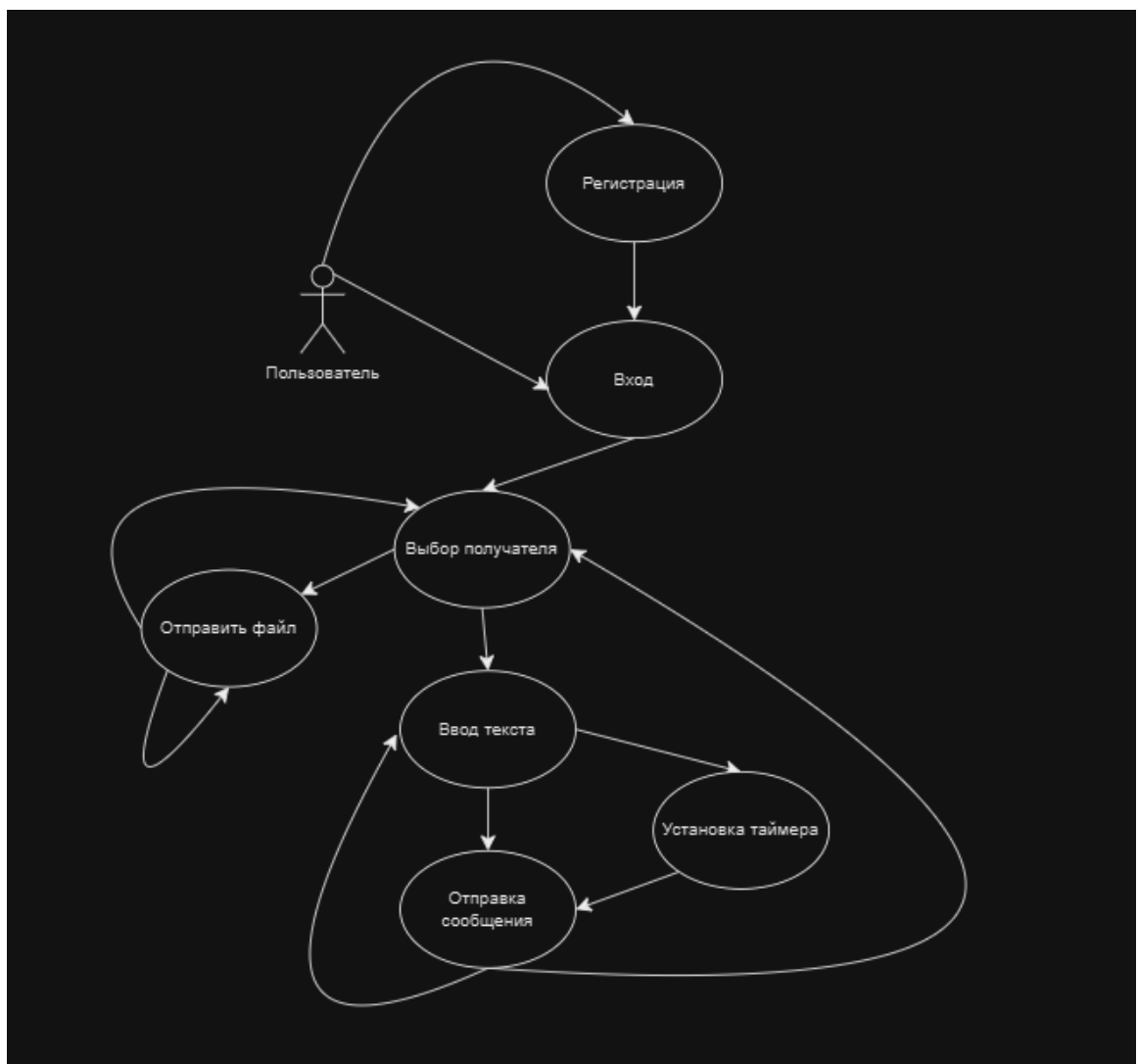


Рисунок 21 – UML-Диаграмма сценариев

Заключение

В результате выполнения данной курсовой работы был разработан мессенджер, обеспечивающий эффективное и безопасное взаимодействие пользователей в локальной сети. Основным достижением стало создание клиент-серверного приложения, которое реализует передачу текстовых сообщений и файлов с использованием надежного протокола TCP/IP.

В процессе разработки приложения были решены важные задачи, такие как проектирование архитектуры системы, создание протокола обмена данными, реализация авторизации и аутентификации пользователей. Особое внимание было уделено удобству взаимодействия пользователей с приложением.

Клиентская часть программы была разработана с использованием технологии WinForms, что обеспечило простой и понятный интерфейс для пользователей. В свою очередь, серверная часть предоставила возможность администрирования и мониторинга состояния системы через консольный интерфейс, что повысило гибкость и удобство работы с приложением.

Протокол обмена данными, который был разработан и реализован в рамках данной работы, стал основой для успешной передачи сообщений и файлов между клиентами. Его надежность была обеспечена механизмом контроля ошибок и повторных попыток передачи данных, что является ключевым аспектом для стабильной работы системы в условиях локальной сети.

Полученный опыт разработки мессенджера с поддержкой обмена текстовыми сообщениями и файлами может быть полезен для дальнейших улучшений программы, а также для создания аналогичных решений в других сферах. В будущем приложение может быть дополнено новыми функциональными возможностями, такими как поддержка мультимедийных сообщений, улучшенные механизмы защиты данных и расширенная функциональность для администраторов сети.

В заключение, выполненная работа позволила приобрести знания и практические навыки в области разработки многопользовательских приложений с использованием клиент-серверной архитектуры, а также углубить понимание процессов передачи данных и сетевых технологий.

Список используемых источников

1. Таненбаум Э. **"Компьютерные сети"**. – М.: Питер, 2011. – 800 с.
2. Риччиардсон Б., Столл Д. **"Сетевое программирование в С#"**. – М.: Вильямс, 2010. – 512 с.
3. Официальная документация Microsoft по .NET и С#: <https://learn.microsoft.com/ru-ru/>
4. Крис Харт. **"Разработка сетевых приложений на основе TCP/IP"**. – СПб.: БХВ-Петербург, 2012. – 640 с.
5. Шилдт Г. **"С# 4.0: Полное руководство"**. – М.: Вильямс, 2011. – 1104 с.

Приложение А.

Листинг программы клиента

Файл ResponseCodes.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace client
{
    public static class ResponseCodes
    {
        public const string SUCCESS = "SUCCESS";
        public const string ERROR = "ERROR";
        public const string USERNAME_ALREADY_EXISTS = "Имя пользователя уже существует";
        public const string USER_NOT_FOUND = "Такого пользователя не существует. Зарегистрируйтесь.";
        public const string USER_ALREADY_LOGGED_IN = "Пользователь уже в сети";
        public const string USER_ALREADY_REGISTR_IN = "Пользователь уже зарегистрирован";
        public const string MISSING_USERNAME_OR_PASSWORD = "Имя пользователя или пароль отсутствуют";
        public const string MISSING_USERNAME = "Пользователь не выбран";
        public const string USER_NOT_ONLINE = "Пользователь не в сети";
        public const string MISSING_USERNAME_OR_RECIPIENT = "Получатель отсутствуют";
        public const string NO_MESSAGES = "Сообщений не найдено";
        public const string ONLY_YOU = "Кроме вас здесь больше никого нет";
        public const string NO_PASSWORD = "Неверный пароль";
        public const string NO_USERS = "В списке только вы";
    }
}
```

Файл RequestCodes.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace client
{
    public static class RequestCodes
    {
        public const string REGISTER = "REGISTER";
        public const string LOGIN = "LOGIN";
        public const string GET_USERS = "GET_USERS";
        public const string SEND_MESSAGE = "SEND_MESSAGE";
    }
}
```

```

        public const string GET_MESSAGES = "GET_MESSAGES";

        public const string NEW_MESSAGE = "NEW_MESSAGE";

        public const string SEND_FILE = "SEND_FILE";

        public const string RECIEVE_FILE = "RECIEVE_FILE";

        public const string CLOSE = "CLOSE";

    }
}

```

Файл messeng.cs

```

using Microsoft.VisualBasic.ApplicationServices;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Collections.Concurrent;

using System.Text;

using System.Net.WebSockets;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;

using System.Timers;

using System.IO;

using System.Threading;

using System.Net.Sockets;

namespace client
{
    public partial class messeng : Form
    {
        private readonly Client _client;

        private bool _isConnected = false;

        private string currentUsername;

        private System.Windows.Forms.Timer messageTimer; // Таймер для отправки сообщений

        private int delayInSeconds = 0; // Задержка для отправки сообщения

        private string pendingMessage; // Сообщение, ожидающее отправки

        private string Username;

        public messeng(string username)
        {
            InitializeComponent();

            _client = Client.GetInstance();

```

```

currentUsername = username;
label1.Text = $"Вы вошли как:{currentUsername}";
this.Load += new System.EventHandler(this.messeng_Load);
dataGridView1.CellClick += DataGridView1_CellClick;
_client.SetMessengInstance(this); // Передача текущей формы
}

```

```

private void DataGridView1_CellClick(object sender, DataGridViewCellEventArgs e)
{
    // Проверяем, чтобы клик был по строке, а не по заголовку
    if (e.RowIndex >= 0)
    {
        // Получаем имя пользователя из выбранной строки
        var selectedUser = dataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString();
        OpenChatWithUser(selectedUser); // Открываем чат с этим пользователем
    }
}

```

```

private async void OpenChatWithUser(string selectedUser)
{
    var userNameWithoutStatus = selectedUser.Split('(')[0].Trim();
    Username = selectedUser.Split('(')[0].Trim();
    label2.Text = $"Чат с пользователем: {userNameWithoutStatus}";
    listBox1.Items.Clear(); // Очищаем чат

```

```

    await _client.SendMessageToServer($"{{RequestCodes.GET_MESSAGES}}:{{currentUsername}}:{{userNameWithoutStatus}}");
}

```

```

private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
    // Проверяем, если нажатая клавиша - двоеточие
    if (e.KeyChar == ':' || e.KeyChar == ';')
    {
        // Отменяем добавление символа в текстовое поле
        e.Handled = true;
        MessageBox.Show("Символы ':' и ';' не могут быть введены.");
    }
}

```

```

private async void button1_Click(object sender, EventArgs e)
{
    // Логика отправки сообщения (пока без сервера)
    var message = textBox1.Text.Trim();
    if (string.IsNullOrEmpty(message))
    {
        MessageBox.Show("Сообщение не может быть пустым!");
    }
}

```

```

        return;
    }
    var chatUser = label2.Text.Replace("Чат с пользователем: ", "");
    await _client.SendMessageToServer($"{RequestCodes.SEND_MESSAGE}:{currentUsername}:{chatUser}:{message}");
}

public static async Task listBox1ItemsAdd(string message, messeng instance)
{
    var c = instance.label2.Text.Replace("Чат с пользователем: ", "");
    instance.listBox1.Items.Add($"{DateTime.Now} {instance.currentUsername} -> {c}: {message}");

    instance.textBox1.Clear();
}

public static async Task GetMessagesFromServer(string result, messeng instance)
{
    try
    {
        // Ответ должен быть в виде списка сообщений
        var responseParts = result.Split(new[] { ':' }, 3);

        // Логируем части ответа
        for (int i = 0; i < responseParts.Length; i++)
        {
            Console.WriteLine($"[LOG] Response part {i}: {responseParts[i]}");
        }

        if (responseParts[1] == ResponseCodes.SUCCESS)
        {
            // Получаем строку с сообщениями (она после "GET_MESSAGES:SUCCESS:")
            var messagesString = responseParts[2]; // Строка сообщений

            // Разделяем сообщения по символу ";"
            var chatMessages = messagesString.Split(';');

            // Логируем количество сообщений
            Console.WriteLine($"[LOG] Количество сообщений: {chatMessages.Length}");

            // Обрабатываем каждое сообщение
            foreach (var msg in chatMessages)
            {
                // Разделяем каждое сообщение на части по ":"
                var msgParts = msg.Split(new[] { ':' }, 4);
                if (msgParts.Length >= 4)
                {
                    // Логируем части сообщения
                    Console.WriteLine($"[LOG] Сообщение разделено на части: Отправитель: {msgParts[0]}, Получатель: {msgParts[1]}, Сообщение: {msgParts[2]}, Дата: {msgParts[3]}");

                    // Формируем строку и добавляем в список

```

```

var formattedMessage = $"({msgParts[3]}) {msgParts[0]} -> {msgParts[1]}: {msgParts[2]}";
Console.WriteLine($"[LOG] Добавляем сообщение в чат: {formattedMessage}");

// Добавляем сообщение в listBox1 (с проверкой на InvokeRequired)
if (instance.listBox1.InvokeRequired)
{
    instance.listBox1.Invoke(new Action(() => instance.listBox1.Items.Add(formattedMessage)));
}
else
{
    instance.listBox1.Items.Add(formattedMessage);
}
}
else
{
    Console.WriteLine("[LOG] Неверный формат сообщения.");
}
}
}
else if (responseParts[1] == ResponseCodes.NO_MESSAGES)
{
    MessageBox.Show("Сообщений нет для этого чата.");
}
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка при получении сообщений: {ex.Message}");
}
}

private async void messeng_Load(object sender, EventArgs e)
{
    // При загрузке формы подключаемся к серверу и получаем список пользователей
    await _client.ConnectAsync(); // Подключение к серверу
    _isConnected = true;

    Task.Run(async () =>
    {
        await _client.SendMessageToServer($"{{RequestCodes.GET_USERS}}:{{currentUsername}}");
        while (_isConnected)
        {
            Console.WriteLine($"зашел в цикл");
            await Task.Delay(5000);
            await _client.SendMessageToServer($"{{RequestCodes.GET_USERS}}:{{currentUsername}}");
        }
    });
    await _client.StartListeningAsync();
}
}

```

```

private void button3_Click(object sender, EventArgs e) //таймер
{
    // Создаем форму для ввода времени таймера
    Form inputForm = new Form();
    inputForm.Text = "Введите время таймера";
    inputForm.Size = new System.Drawing.Size(300, 150);

    Label label = new Label() { Left = 20, Top = 20, Text = "Введите время (сек):", AutoSize = true };
    TextBox inputBox = new TextBox() { Left = 150, Top = 20, Width = 100 };
    Button confirmation = new Button() { Text = "ОК", Left = 100, Top = 60, Width = 80 };

    confirmation.Click += (s, args) =>
    {
        if (int.TryParse(inputBox.Text, out int seconds) && seconds > 0)
        {
            delayInSeconds = seconds;
            MessageBox.Show($"Сообщение будет отправлено через {seconds} секунд.", "Таймер установлен",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
            inputForm.Close();

            StartMessageTimer();
        }
        else
        {
            MessageBox.Show("Введите корректное число секунд.", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    };

    inputForm.Controls.Add(label);
    inputForm.Controls.Add(inputBox);
    inputForm.Controls.Add(confirmation);
    inputForm.ShowDialog();
}

private void StartMessageTimer()
{
    if (messageTimer?.Enabled == true) return; // Предотвращаем повторный запуск

    if (messageTimer == null)
    {
        messageTimer = new System.Windows.Forms.Timer();
        messageTimer.Tick += async (s, args) =>
        {
            messageTimer.Stop();
            messageTimer = null;
            textBox1.KeyDown -= TextBox1_KeyDown; // Убираем обработчик
            textBox1.ReadOnly = false; // Снимаем блокировку
            button1.PerformClick();
        };
    }
}

```

```

textBox1.ReadOnly = true; // Делаем текстовый только для чтения
textBox1.KeyDown += TextBox1_KeyDown; // Добавляем обработчик для блокировки удаления

messageTimer.Interval = delayInSeconds * 1000;
messageTimer.Start();
}

private void TextBox1_KeyDown(object sender, KeyEventArgs e)
{
    // Блокируем кнопки удаления
    if (e.KeyCode == Keys.Back || e.KeyCode == Keys.Delete)
    {
        e.SuppressKeyPress = true;
    }
}

public static async Task OnMessageReceived(string message, messeng instance)
{
    var parts = message.Split(':');

    var sender = parts[1];
    var text = parts[2];
    var formattedMessage = $"({DateTime.Now}) {sender} -> {instance.CurrentUsername}: {text}";
    if (sender == instance.Username)
    {
        // Обновляем UI потока
        if (instance.listBox1.InvokeRequired)
        {
            instance.listBox1.Invoke(new Action(() => instance.listBox1.Items.Add(formattedMessage)));
        }
        else
        {
            instance.listBox1.Items.Add(formattedMessage);
        }
    }
}

public static async Task UpdateUsersList(string result, messeng instance)
{
    try
    {
        //Console.WriteLine($"[LOG] UpdateUsersList вызван с результатом: {result}");

        var responseParts = result.Split(':');
        if (responseParts[1] == ResponseCodes.SUCCESS)
        {
            var users = responseParts[2].Split(',').ToList();

            // Проверяем доступ из UI-потока
            if (instance.InvokeRequired)
            {
                instance.Invoke(new Action(() => instance.PopulateUsers(users)));
            }
            else

```

```

        {
            instance.PopulateUsers(users);
        }
    }
    else if (responseParts[1] == ResponseCodes.NO_USERS)
    {
        Console.WriteLine($"{ResponseCodes.NO_USERS}");
    }
    else if (responseParts[1] == ResponseCodes.ONLY_YOU)
    {

        MessageBox.Show($"{ResponseCodes.ONLY_YOU}");
    }
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка при обновлении списка пользователей: {ex.Message}");
}
}

```

```

private void PopulateUsers(List<string> users)
{
    if (dataGridView1.InvokeRequired)
    {
        dataGridView1.Invoke(new Action(() => PopulateUsers(users)));
        return; // Завершаем выполнение в рабочем потоке
    }

    // Сохраняем выбранного пользователя
    string selectedUser = null;
    if (dataGridView1.CurrentRow != null)
    {
        selectedUser = dataGridView1.CurrentRow.Cells[0].Value?.ToString();
    }

    // Отключаем обработчик CellClick
    dataGridView1.CellClick -= DataGridView1_CellClick;

    // Обновляем данные
    var bindingSource = new BindingSource();
    bindingSource.DataSource = users.Select(u => new { Username = u }).ToList();
    dataGridView1.DataSource = bindingSource;

    // Восстанавливаем выбор
    if (!string.IsNullOrEmpty(selectedUser))
    {
        foreach (DataGridViewRow row in dataGridView1.Rows)
        {
            if (row.Cells[0].Value?.ToString() == selectedUser)
            {

```

```

        row.Selected = true;
        dataGridView1.CurrentCell = row.Cells[0];
        break;
    }
}
}

// Включаем обработчик CellClick обратно
dataGridView1.CellClick += DataGridView1_CellClick;

}

```

```

private async void button2_Click(object sender, EventArgs e) // кнопка отправки файла
{
    string filePath = null;
    string fileName = null;
    Thread staThread = new Thread(() =>
    {
        using (OpenFileDialog openFileDialog = new OpenFileDialog())
        {
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                filePath = openFileDialog.FileName;
                fileName = Path.GetFileName(filePath);
            }
        }
    });
    staThread.SetApartmentState(ApartmentState.STA);
    staThread.Start();
    staThread.Join(); // Ожидаем завершения работы потока

    if (!string.IsNullOrEmpty(filePath))
    {
        var chatUser = label2.Text.Replace("Чат с пользователем: ", "");
        string message = $"{RequestCodes.SEND_FILE}:{currentUsername}:{chatUser}:{fileName}";
        await _client.SendMessageToServer(message);
        _client.file_send = true;
        try
        {
            await SendFileAsync(_client._networkStream, message, filePath);
            MessageBox.Show($"Файл {fileName} успешно отправлен.", "Успех", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
            await _client.SendMessageToServer($"{RequestCodes.SEND_MESSAGE}:{currentUsername}:{chatUser}:{fileName}");
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при отправке файла: {ex.Message}", "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        }
    }
}

```

```

    }
}
else
{
    MessageBox.Show("Файл не выбран.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
_client.file_send = false;
}
public async Task SendFileAsync(NetworkStream stream, string message, string filePath)
{
    try
    {
        // Подготовка буфера для передачи файла
        byte[] buffer = new byte[4096];
        using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))
        {
            int bytesRead;
            while ((bytesRead = fs.Read(buffer, 0, buffer.Length)) > 0)
            {
                // Отправка содержимого файла порциями
                await stream.WriteAsync(buffer, 0, bytesRead);
            }
        }

        // Очистка потока после завершения передачи
        await stream.FlushAsync();
        Console.WriteLine("Файл успешно отправлен.");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Ошибка при отправке файла: {ex.Message}");
    }
}

private async void button4_Click(object sender, EventArgs e)
{
    await _client.SendMessageToServer($"{RequestCodes.CLOSE}:{currentUsername}");
    Application.Exit();
    _isConnected = false;
    _client.Disconnect(); // Закрытие соединения
}
}
}

```

Файл client.cs

```

using System;

using System.IO;

using System.Net.Sockets;

using System.Text;

using System.Threading.Tasks;

```

```
using System.Windows.Forms;
```

```
namespace client
```

```
{
```

```
    public class Client
```

```
    {
```

```
        private static Client _instance;
```

```
        private readonly string _serverAddress;
```

```
        private readonly int _serverPort;
```

```
        private TcpClient _client;
```

```
        public NetworkStream _networkStream;
```

```
        private static readonly object _lock = new object(); // Блокировка для поточной безопасности
```

```
        private messeng _messengInstance;
```

```
        private Registr _reg;
```

```
        private Avtorisation _Avt;
```

```
        private Client(string serverAddress, int serverPort)
```

```
        {
```

```
            _serverAddress = serverAddress;
```

```
            _serverPort = serverPort;
```

```
        }
```

```
        public void SetMessengInstance(messeng messengInstance)
```

```
        {
```

```
            _messengInstance = messengInstance;
```

```
        }
```

```
        public void SetRegistrInstance(Registr registrInstance)
```

```
        {
```

```
            _reg = registrInstance;
```

```
        }
```

```
        public void SetAvtInstance(Avtorisation AvtInstance)
```

```
        {
```

```
            _Avt = AvtInstance;
```

```
        }
```

```
        public static Client GetInstance(string serverAddress = "127.0.0.1", int serverPort = 5000)
```

```
        {
```

```
            lock (_lock) // Поточная безопасность
```

```
            {
```

```
                if (_instance == null)
```

```

    {
        _instance = new Client(serverAddress,serverPort);
    }

    return _instance;
}
}

public async Task ConnectAsync()
{

    if (_client != null && _client.Connected)
    {
        return; // Уже подключено
    }

    try
    {
        _client = new TcpClient();
        Console.WriteLine("Connecting to the server...");
        await _client.ConnectAsync(_serverAddress, _serverPort);
        _networkStream = _client.GetStream();
        Console.WriteLine("Connected to the server.");
    }
    catch (SocketException ex)
    {
        throw new Exception($"ERROR: {ex.Message}");
    }
    catch (Exception ex)
    {
        throw new Exception($"ERROR: {ex.Message}");
    }

    await Task.Delay(1000);

}

int l = 0;

public bool file_send = false;

public async Task<string> SendMessageToServer(string message)

```

```

{
    //while (file_send) { }

    Console.WriteLine($"Отправка на сервер: {message}");

    if (_networkStream == null || !_client.Connected)
    {
        throw new InvalidOperationException("Client is not connected.");
    }

    try
    {

        var data = Encoding.UTF8.GetBytes(message);
        await _networkStream.WriteAsync(data, 0, data.Length);
        return "ERROR:";
    }
    catch (SocketException ex)
    {
        return $"ERROR: SocketException: {ex.Message}";
    }
    catch (ObjectDisposedException ex)
    {
        return $"ERROR: ObjectDisposedException: {ex.Message}";
    }
    catch (Exception ex)
    {
        return $"ERROR: {ex.Message}";
    }
}

bool rec_file = false;
FileStream fs = null;
int k = 0;
public async Task StartListeningAsync()
{
    if (_networkStream == null || !_client.Connected)
    {
        throw new InvalidOperationException("Client is not connected.");
    }
}

```

```

var buffer = new byte[4096];
while (_client.Connected)
{
    try
    {
        var bytesRead = await _networkStream.ReadAsync(buffer, 0, buffer.Length);

        if (bytesRead == 0) break;

        var message = Encoding.UTF8.GetString(buffer, 0, bytesRead);
        Console.WriteLine($"получено в StartListeningAsync{message}");

        if (message.Split(':')[0] == RequestCodes.GET_USERS)
        {
            if (_messengInstance != null)
            {
                await messeng.UpdateUsersList(message, _messengInstance);
            }
        }
        else if (message.Split(':')[0] == RequestCodes.LOGIN)
        {
            Avtorisation.Login(message, _Avt);
        }
        else if (message.Split(':')[0] == RequestCodes.REGISTER)
        {
            var avtorisationForm = new Avtorisation();
            Registr.registr(message, _reg, avtorisationForm);
        }
        else if (message.Split(':')[0] == RequestCodes.GET_MESSAGES)
        {
            await messeng.GetMessagesFromServer(message, _messengInstance);
        }
        else if (message.Split(':')[0] == RequestCodes.NEW_MESSAGE)
        {
            await messeng.OnMessageReceived(message, _messengInstance);
        }
    }
}

```

```

else if (message.Split(':')[0] == RequestCodes.SEND_MESSAGE)

{
    if (message.Split(':')[1] == ResponseCodes.MISSING_USERNAME)
    {
        MessageBox.Show($"{ResponseCodes.MISSING_USERNAME}");
    }
    else if (message.Split(':')[1] == ResponseCodes.SUCCESS)
    {
        await messeng.listBox1ItemsAdd(message.Split(':')[2], _messengInstance);
    }
}

else if (message.Split(':')[0] == RequestCodes.RECIEVE_FILE || rec_file == true)
{
    try
    {
        if (rec_file == false)
        {
            // Парсим сообщение
            var parts = message.Split(':');
            if (parts.Length < 3)
            {
                Console.WriteLine("Ошибка: Неверный формат команды RECEIVE_FILE.");
                return;
            }

            string senderUsername = parts[1]; // Имя отправителя
            string fileName = CleanFileName(parts[2]); // Очистка имени файла

            // Путь для сохранения файла
            string saveDirectory = Path.Combine("Downloads", senderUsername);
            if (!Directory.Exists(saveDirectory))
            {
                Directory.CreateDirectory(saveDirectory);
            }
            string savePath = Path.Combine(saveDirectory, fileName);

            Console.WriteLine($"Получение файла '{fileName}' от '{senderUsername}' ...");

```

```

        fs = new FileStream(savePath, FileMode.Create, FileAccess.Write);
    }
    // Читаем данные из потока и записываем в файл

    var message1 = Encoding.UTF8.GetString(buffer, 0, bytesRead);
    await fs.WriteAsync(buffer, 0, bytesRead);
    if (bytesRead < 4096 && k > 0)
    {
        k = 0;
        rec_file = false;
        fs?.Dispose(); // Закрываем поток
        fs = null;

    }
    rec_file = true;
    k++;
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка при получении файла: {ex.Message}");
    MessageBox.Show($"Ошибка при получении файла: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка при получении сообщения: {ex.Message}");
    break;
}
}

}

string CleanFileName(string fileName)
{

```

```

        // Заменяем недопустимые символы на подчеркивания
        var invalidChars = Path.GetInvalidFileNameChars();
        foreach (var c in invalidChars)
        {
            fileName = fileName.Replace(c, '_');
        }
        return fileName;
    }

    public bool IsConnected()
    {
        return _client != null && _client.Connected && _networkStream != null;
    }

    public void Disconnect()
    {
        _networkStream?.Close();
        _client?.Close();
    }
}
}

```

Файл Programm.cs

```

using System;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace client
{
    static class Program
    {
        [STAThread]

        static async Task Main() // Сделаем метод асинхронным
        {
            Application.EnableVisualStyles();

            Application.SetCompatibleTextRenderingDefault(false);

            var client = Client.GetInstance();

            try
            {

```

```

        await client.ConnectAsync(); // Подключаемся асинхронно

        Application.Run(new Avtorisation());
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при запуске приложения: {ex.Message}");// Если что-то пошло не так - покажем
        сообщение
    }
    }
}
}

```

Файл Registr.cs

```

using System;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace client
{
    public partial class Registr : Form
    {
        private readonly Client _client;

        public Registr()
        {
            InitializeComponent();

            _client = Client.GetInstance();
            _client.SetRegistrInstance(this);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // Переход на форму авторизации
            var avtorisationForm = new Avtorisation();
            avtorisationForm.Show();
            this.Hide();
        }
    }
}

```

```

private async void button1_Click(object sender, EventArgs e)
{
    var username = textBox2.Text;
    var password = textBox1.Text;
    // Console.WriteLine($"username: {username}");
    await _client.SendMessageToServer($"{RequestCodes.REGISTER}:{username}:{password}");
    await _client.StartListeningAsync();
    this.Hide();
}

public static void registr(string result, Registr registrForm, Avtorisation avtorisationForm) {

    var responseParts = result.Split(':');
    if (responseParts.Length == 3 || responseParts.Length == 2)
    {

        if (responseParts[1] == ResponseCodes.SUCCESS)
        {
            var username = responseParts[2];
            MessageBox.Show("Registration successful");

            avtorisationForm.Show();
            registrForm.Hide();
        }
        else if (responseParts[1] == ResponseCodes.MISSING_USERNAME_OR_PASSWORD)
        {
            MessageBox.Show($"{ResponseCodes.MISSING_USERNAME_OR_PASSWORD}");
        }
        else if (responseParts[1] == ResponseCodes.USER_ALREADY_REGISTR_IN)
        {
            MessageBox.Show($"{ResponseCodes.USER_ALREADY_REGISTR_IN}");
        }
    }

}
}
}

```

Файл Avtorisation.cs

```
using System;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.StartPanel;


namespace client
{
    public partial class Avtorisation : Form
    {
        private readonly Client _client;

        public Avtorisation()
        {
            InitializeComponent();

            _client = Client.GetInstance();

            _client.SetAvtInstance(this);
        }

        private void button2_Click(object sender, EventArgs e)
        {
            // Переход на форму регистрации

            var registrForm = new Registr();

            registrForm.Show();

            this.Hide();
        }

        private async void button1_Click(object sender, EventArgs e)
        {

            var username = textBox2.Text;

            var password = textBox1.Text;

            await _client.SendMessageToServer($"{RequestCodes.LOGIN}:{username}:{password}");
        }
    }
}
```

```

        await _client.StartListeningAsync();
    }

    public static void Login (string result, Avtorisation avtorisation)
    {
        var responseParts = result.Split(':');

        if (responseParts.Length == 4 || responseParts.Length == 2)
        {
            if (responseParts[1] == ResponseCodes.SUCCESS)
            {
                var username = responseParts[2];

                Console.WriteLine($"username: {username}");

                MessageBox.Show("Login successful");

                // Переход на форму мессенджера
                var messengForm = new messeng(username);

                messengForm.Show();

                avtorisation.Hide(); // Скрываем текущую форму
            }
            else if (responseParts[1] == ResponseCodes.MISSING_USERNAME_OR_PASSWORD)
            {
                MessageBox.Show($"{ResponseCodes.MISSING_USERNAME_OR_PASSWORD}");
            }
            else if (responseParts[1] == ResponseCodes.USER_ALREADY_LOGGED_IN)
            {
                MessageBox.Show($"{ResponseCodes.USER_ALREADY_LOGGED_IN}");
            }
            else if (responseParts[1] == ResponseCodes.USER_NOT_FOUND)
            {
                MessageBox.Show($"{ResponseCodes.USER_NOT_FOUND}");
            }
            else if (responseParts[1] == ResponseCodes.NO_PASSWORD)
            {
                MessageBox.Show($"{ResponseCodes.NO_PASSWORD}");
            }
        }
    }
}

```

Приложение Б

Листинг программы сервера

Файл Programm.cs

```
using Microsoft.VisualBasic.ApplicationServices;

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace Server
{
    class Program
    {
        private static readonly string FilePath = "users.txt";

        private static List<string> onlineUsers = new List<string>();

        private static readonly object onlineUsersLock = new object();

        private static readonly Dictionary<string, TcpClient> onlineClients = new Dictionary<string, TcpClient>();

        private static readonly SemaphoreSlim onlineUsersSemaphore = new SemaphoreSlim(1, 1);

        private static bool getUsers = true;

        public class Message
        {
            public string Sender { get; set; }

            public string Recipient { get; set; }

            public string message { get; set; }

            public DateTime Timestamp { get; set; }
        }

        static async Task Main(string[] args)
        {
            var listener = new TcpListener(IPAddress.Any, 5000);

            listener.Start();
```

```

Console.WriteLine("Server started...");

while (true)
{
    var client = await listener.AcceptTcpClientAsync();
    _ = HandleClientAsync(client);
}
}

private static async Task HandleClientAsync(TcpClient client)
{
    string username = null;
    bool isConnected = false;

    try
    {
        using (var networkStream = client.GetStream())
        {
            var buffer = new byte[1024];
            while (true)
            {
                var bytesRead = await networkStream.ReadAsync(buffer, 0, buffer.Length);
                if (bytesRead == 0) break;

                var message = Encoding.UTF8.GetString(buffer, 0, bytesRead);
                Console.WriteLine($"Получено сообщение: {message}");
                var response = await ProcessMessageAsync(message, networkStream, client);

                Console.WriteLine($"Отправляем ответ: {response}");

                var responseParts = response.Split(':');
                Console.WriteLine($"status: {responseParts[0]}");
                var data = Encoding.UTF8.GetBytes(response);
                Console.WriteLine("Подготовка к отправке ответа...");
                await networkStream.WriteAsync(data, 0, data.Length);
                Console.WriteLine("Ответ отправлен клиенту.");
            }
        }
    }
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    finally
    {
    }
}

private static async Task SaveUsersToFile(List<User> users)
{
    var lines = users.Select(u => $"{u.Username}:{u.Password}");
    await File.WriteAllLinesAsync(FilePath, lines);
}

private static async Task<string> ProcessMessageAsync(string message, NetworkStream stream, TcpClient client)
{
    Console.WriteLine($"Получено сообщение: {message}");
    var parts = message.Split(':');

    var command = parts[0];
    string username = null, password = null, recipient = null, text = null;
    string fileName = null;

    switch (command)
    {
        case RequestCodes.REGISTER:
        {
            username = parts.Length > 1 ? parts[1] : null;
            password = parts.Length > 2 ? parts[2] : null;
            break;
        }
        case RequestCodes.LOGIN:
        {
            username = parts.Length > 1 ? parts[1] : null;
            password = parts.Length > 2 ? parts[2] : null;
            break;
        }
    }
}

```

case RequestCodes.SEND_MESSAGE:

```
{

    username = parts.Length > 1 ? parts[1] : null;
    recipient = parts.Length > 2 ? parts[2] : null;
    text = parts.Length > 3 ? parts[3] : null;
    Console.WriteLine($"Получено сообщение: {text}");
    break;
}
```

case RequestCodes.GET_MESSAGES:

```
{
    // Для получения сообщений параметры: username, recipient
    username = parts.Length > 1 ? parts[1] : null;
    recipient = parts.Length > 2 ? parts[2] : null;
    break;
}
```

case RequestCodes.GET_USERS:

```
{
    // Для получения списка пользователей параметр только username
    username = parts.Length > 1 ? parts[1] : null;
    break;
}
```

case RequestCodes.SEND_FILE:

```
{
    username = parts.Length > 1 ? parts[1] : null;
    recipient = parts.Length > 2 ? parts[2] : null;
    fileName = parts.Length > 3 ? parts[3] : null;

    break;
}
```

case RequestCodes.CLOSE:

```
{
    username = parts.Length > 1 ? parts[1] : null;
    break;
}
```

default:

```

        return $"{ResponseCodes.ERROR}: Unknown command";
    }

    var users = await GetUsersFromFile();
    Console.WriteLine($" command: {command}");
    return command switch
    {
        RequestCodes.REGISTER => await HandleRegisterCommand(users, username, password),
        RequestCodes.LOGIN => await HandleLoginCommand(users, username, password, client),
        RequestCodes.GET_USERS => await HandleGetUsersCommand(users, username),
        RequestCodes.SEND_MESSAGE => await HandleSendMessageCommand(username, recipient, text),
        RequestCodes.GET_MESSAGES => await HandleGetMessagesCommand(username, recipient),
        RequestCodes.SEND_FILE => await HandleSendFile(username, recipient, fileName, stream),
        RequestCodes.CLOSE => Disconnect(username),
        _ => $"{ResponseCodes.ERROR}: Unknown command"
    };
}

static async Task<string> HandleSendFile(string username, string recipient, string fileName, NetworkStream stream)
{
    try
    {
        // Проверяем, что параметры указаны
        if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(recipient) || string.IsNullOrEmpty(fileName))
        {
            return "Ошибка: Недостаточно параметров для обработки файла.";
        }

        // Директория для сохранения файла
        string saveDirectory = Path.Combine("ReceivedFiles", recipient);

        // Создаем директорию, если она не существует
        if (!Directory.Exists(saveDirectory))
        {
            Directory.CreateDirectory(saveDirectory);
        }

        // Полный путь для сохранения файла

```

```

string filePath = Path.Combine(saveDirectory, fileName);

// Чтение данных из потока и запись в файл
using (FileStream fs = new FileStream(filePath, FileMode.Create, FileAccess.Write))
{
    byte[] buffer = new byte[4096];
    int bytesRead;

    while ((bytesRead = await stream.ReadAsync(buffer, 0, buffer.Length)) > 0)
    {
        await fs.WriteAsync(buffer, 0, bytesRead);

        // Проверяем, если данных больше нет, завершаем чтение
        if (!stream.DataAvailable)
        {
            break;
        }
    }
}

string sendResult = await HandleReciveFile(username, recipient, fileName);

Console.WriteLine($"Файл '{fileName}' от пользователя '{username}' успешно сохранен для '{recipient}' в '{filePath}'.");

return $"Файл '{fileName}' успешно отправлен.";
}

catch (Exception ex)
{
    Console.WriteLine($"Ошибка при обработке файла: {ex.Message}");
    return $"Ошибка: {ex.Message}";
}
}

static string Disconnect(string username)
{
    onlineClients.Remove(username);
    onlineUsers.Remove(username);
    Console.WriteLine($"Пользователь {username} вышел.");

    return $"{RequestCodes.CLOSE}";
}

```

```

}

static async Task<string> HandleReciveFile(string username, string recipient, string fileName)
{
    try
    {
        // Проверяем, что параметры указаны
        if (string.IsNullOrEmpty(recipient) || string.IsNullOrEmpty(fileName))
        {
            return "Ошибка: Недостаточно параметров для отправки файла.";
        }

        string filePath = Path.Combine("ReceivedFiles", recipient, fileName);

        // Проверяем, существует ли файл
        if (!File.Exists(filePath))
        {
            return $"Ошибка: Файл '{fileName}' для '{recipient}' не найден.";
        }

        await onlineUsersSemaphore.WaitAsync();
        try
        {
            // Проверяем, находится ли получатель в сети
            if (onlineUsers.Contains(recipient))
            {
                var recipientClient = onlineClients[recipient];
                if (recipientClient != null && recipientClient.Connected)
                {
                    using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))
                    {
                        byte[] buffer = new byte[4096];
                        int bytesRead;

                        // Отправляем команду получателю о получении файла
                        var header = $"{RequestCodes.RECIEVE_FILE}:{recipient}:{fileName}";
                        var headerData = Encoding.UTF8.GetBytes(header + "\n");
                        await recipientClient.GetStream().WriteAsync(headerData, 0, headerData.Length);
                        await Task.Delay(100);
                    }
                }
            }
        }
    }
}

```

```

        // Отправляем сам файл
        while ((bytesRead = await fs.ReadAsync(buffer, 0, buffer.Length)) > 0)
        {
            await recipientClient.GetStream().WriteAsync(buffer, 0, bytesRead);
            await Task.Delay(100);
        }
    }

    Console.WriteLine($"Файл '{fileName}' успешно отправлен пользователю '{recipient}'.");
    return $"Файл '{fileName}' успешно отправлен пользователю '{recipient}'. ";
}

else
{
    return $"Ошибка: Пользователь '{recipient}' не подключен.";
}

}

else
{
    return $"Ошибка: Пользователь '{recipient}' не онлайн.";
}

}

finally
{
    onlineUsersSemaphore.Release();
}

}

catch (Exception ex)
{
    Console.WriteLine($"Ошибка при отправке файла: {ex.Message}");
    return $"Ошибка: {ex.Message}";
}

}

static async Task<string> HandleSendMessageCommand(string username, string recipient, string message)
{
    if (string.IsNullOrEmpty(recipient) )

```

```

{
    return $"{RequestCodes.SEND_MESSAGE}:{ResponseCodes.MISSING_USERNAME}";
}

// Сохранение сообщения
await SaveMessage(username, recipient, message);

await onlineUsersSemaphore.WaitAsync();

try
{
    if (onlineUsers.Contains(recipient))
    {
        var recipientClient = onlineClients[recipient]; // onlineClients — это словарь для хранения TcpClient получателей
        if (recipientClient != null && recipientClient.Connected)
        {
            try
            {
                var data = Encoding.UTF8.GetBytes($"{RequestCodes.NEW_MESSAGE}:{username}:{message}");
                await recipientClient.GetStream().WriteAsync(data, 0, data.Length);
                Console.WriteLine($"отправили сообщение {recipientClient}");
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Ошибка отправки сообщения получателю {recipient}: {ex.Message}");
            }
        }
    }
}

finally
{
    onlineUsersSemaphore.Release();
}

return $"{RequestCodes.SEND_MESSAGE}:{ResponseCodes.SUCCESS}:{message}";
}

static async Task<string> HandleGetMessagesCommand(string username, string recipient)
{
    Console.WriteLine($"HandleGetMessagesCommand: {username}");
}

```

```

Console.WriteLine($"HandleGetMessagesCommand: {recipient}");

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(recipient))
{

    return ResponseCodes.MISSING_USERNAME_OR_RECIPIENT;
}

var messages = await GetMessagesFromFile();

Console.WriteLine($"[ЛОГ] Сообщений загружено: {messages.Count}");

foreach (var msg in messages)
{
    Console.WriteLine($"[ЛОГ] Сообщение: Отправитель: {msg.Sender}, Получатель: {msg.Recipient}, Сообщение: {msg.message}, Дата: {msg.Timestamp}");
}

// Фильтрация сообщений между пользователем и получателем
var chatMessages = messages

    .Where(m => (m.Sender == username && m.Recipient == recipient) || (m.Sender == recipient && m.Recipient == username))

    .OrderBy(m => m.Timestamp)

    .Select(m => $"{m.Sender}:{m.Recipient}:{m.message}:{m.Timestamp}") // Формируем полные строки с датой

    .ToList();

Console.WriteLine($"[ЛОГ] Количество сообщений, подходящих под критерии: {chatMessages.Count}");

if (chatMessages.Count == 0)
{
    // Обработка случая, когда нет сообщений
    Console.WriteLine("[ЛОГ] Нет сообщений для этого чата.");

    return $"{RequestCodes.GET_MESSAGES}:{ResponseCodes.NO_MESSAGES}:No messages found";
}

var response = $"{RequestCodes.GET_MESSAGES}:{ResponseCodes.SUCCESS}:{string.Join(" ", chatMessages)}";

Console.WriteLine($"[ЛОГ] Сформирован ответ: {response}");

return response;
}

private static async Task<List<Message>> GetMessagesFromFile()
{
    Console.WriteLine("[ЛОГ] Начинаем чтение сообщений из файла...");

    if (!File.Exists("messages.txt"))

```

```

{
    return new List<Message>();
}

var lines = await File.ReadAllLinesAsync("messages.txt");
Console.WriteLine($"[ЛОГ] Прочитано строк из файла: {lines.Length}");
var messages = new List<Message>();

foreach (var line in lines)
{
    try
    {
        Console.WriteLine($"[ЛОГ] Обработка строки: {line}");

        // Разделение строки по ':'
        var parts = line.Split(':');

        Console.WriteLine($"[ЛОГ] Строка разделена на {parts.Length} части");

        // Проверка, что строка содержит ожидаемое количество частей
        if (parts.Length < 4)
        {
            Console.WriteLine($"Неверный формат строки: {line}");
            continue;
        }

        // Собираем отправителя, получателя, сообщение
        var sender = parts[0].Trim();
        var recipient = parts[1].Trim();
        var message = parts[2].Trim();

        Console.WriteLine($"[ЛОГ] Отправитель: {sender}, Получатель: {recipient}, Сообщение: {message}");

        // Соединяем оставшиеся части для корректной даты
        var timestampRaw = string.Join(":", parts.Skip(3)).Trim();

        Console.WriteLine($"[ЛОГ] Исходная дата и время: {timestampRaw}");

        // Парсим дату и добавляем сообщение в список, если дата корректна
        if (DateTime.TryParse(timestampRaw, out var timestamp))
        {
            messages.Add(new Message
            {
                Sender = sender,
                Recipient = recipient,

```

```

        message = message,
        Timestamp = timestamp
    });

    Console.WriteLine($"[ЛОГ] Сообщение добавлено с датой: {timestamp}");
}

else
{
    Console.WriteLine($"Ошибка парсинга даты: {timestampRaw}");
}
}

catch (Exception ex)
{
    Console.WriteLine($"Ошибка обработки строки: {line}. Exception: {ex.Message}");
}

}

Console.WriteLine($"сообщение: {messages}");
return messages;
}

private static async Task SaveMessage(string sender, string recipient, string message)
{
    var messageLine = $"{sender}:{recipient}:{message}:{DateTime.Now}";

    // Сохраняем сообщения в файл
    await File.AppendAllLinesAsync("messages.txt", new[] { messageLine });
}

static async Task<string> HandleRegisterCommand(List<User> users, string username, string password)
{
    if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
    {
        return $"{RequestCodes.REGISTER}:{ResponseCodes.MISSING_USERNAME_OR_PASSWORD}";
    }

    if (users.Any(u => u.Username == username))
    {
        return $"{RequestCodes.REGISTER}:{ResponseCodes.USER_ALREADY_REGISTR_IN}";
    }
}

```

```

        users.Add(new User { Username = username, Password = password });

        await SaveUsersToFile(users);

        return $"{RequestCodes.REGISTER}:{ResponseCodes.SUCCESS}:User successfully registered";
    }

static async Task<string> HandleLoginCommand(List<User> users, string username, string password, TcpClient client)
{
    if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password))
    {
        return $"{RequestCodes.LOGIN}:{ResponseCodes.MISSING_USERNAME_OR_PASSWORD}";
    }

    if (onlineUsers.Contains(username))
    {
        return $"{RequestCodes.LOGIN}:{ResponseCodes.USER_ALREADY_LOGGED_IN}";
    }

    var logUser = users.FirstOrDefault(u => u.Username == username );
    if (logUser == null)
    {
        return $"{RequestCodes.LOGIN}:{ResponseCodes.USER_NOT_FOUND}";
    }

    var pasUser = users.FirstOrDefault( u => u.Username == username && u.Password == password );
    if (pasUser == null)
    {
        return $"{RequestCodes.LOGIN}:{ResponseCodes.NO_PASSWORD}";
    }

    onlineUsers.Add(username);
    onlineClients[username] = client;
    return $"{RequestCodes.LOGIN}:{ResponseCodes.SUCCESS}:{username}:Login successful";
}

static async Task<string> HandleGetUsersCommand(List<User> users, string currentUsername)
{
    var allLines = File.ReadAllLines(FilePath);

    if (allLines.Length == 0)
    {

```

```

        getUsers = true;

    }

    var usernames = File.ReadAllLines(FilePath)
    .Select(line => line.Split(':')[0]) // Извлекаем имена пользователей
    .Where(user => user != currentUsername) // Исключаем текущего пользователя
    .Select(user => $"{user} {(onlineUsers.Contains(user) ? "(Online)" : "(Offline))}");

    if (!usernames.Any() && getUsers)
    {
        getUsers = false;

        // Если пользователей после фильтрации нет
        return $"{RequestCodes.GET_USERS}:{ResponseCodes.ONLY_YOU}";
    }

    else if (!usernames.Any() && !getUsers)
    {
        return $"{RequestCodes.GET_USERS}:{ResponseCodes.NO_USERS}";
    }

    // Возвращаем строку с пользователями, разделенную запятой
    string result = $"{RequestCodes.GET_USERS}:{ResponseCodes.SUCCESS}:{string.Join(",", usernames)}";

    //Console.WriteLine($"Sending response: {result}");

    return result;
}

```

```

private static async Task<List<User>> GetUsersFromFile()
{
    if (!File.Exists(FilePath))
    {
        return new List<User>();
    }

    var lines = await File.ReadAllLinesAsync(FilePath);

    var users = new List<User>();

```

```

        foreach (var line in lines)
        {
            var parts = line.Split(':');
            if (parts.Length == 2)
            {
                users.Add(new User { Username = parts[0], Password = parts[1] });
            }
        }

        return users;
    }
}

```

```

public class User
{
    public string Username { get; set; }
    public string Password { get; set; }
}

```

Файл ResponseCodes.cs

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace client
{
    public static class ResponseCodes
    {
        public const string SUCCESS = "SUCCESS";
        public const string ERROR = "ERROR";
        public const string USERNAME_ALREADY_EXISTS = "Имя пользователя уже существует";
        public const string USER_NOT_FOUND = "Такого пользователя не существует. Зарегистрируйтесь.";
        public const string USER_ALREADY_LOGGED_IN = "Пользователь уже в сети";
        public const string USER_ALREADY_REGISTR_IN = "Пользователь уже зарегистрирован";
        public const string MISSING_USERNAME_OR_PASSWORD = "Имя пользователя или пароль отсутствуют";
        public const string MISSING_USERNAME = "Пользователь не выбран";
        public const string USER_NOT_ONLINE = "Пользователь не в сети";
        public const string MISSING_USERNAME_OR_RECIPIENT = "Получатель отсутствуют";
    }
}

```

```

        public const string NO_MESSAGES = "Сообщений не найдено";

        public const string ONLY_YOU = "Кроме вас здесь больше никого нет(";
        public const string NO_PASSWORD = "Неверный пароль";
        public const string NO_USERS = "В списке только вы";
    }
}

```

Файл RequestCodes.cs

```

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace client
{
    public static class RequestCodes
    {
        public const string REGISTER = "REGISTER";
        public const string LOGIN = "LOGIN";
        public const string GET_USERS = "GET_USERS";
        public const string SEND_MESSAGE = "SEND_MESSAGE";
        public const string GET_MESSAGES = "GET_MESSAGES";
        public const string NEW_MESSAGE = "NEW_MESSAGE";
        public const string SEND_FILE = "SEND_FILE";
        public const string RECIEVE_FILE = "RECIEVE_FILE";
        public const string CLOSE = "CLOSE";
    }
}

```

Приложение В.

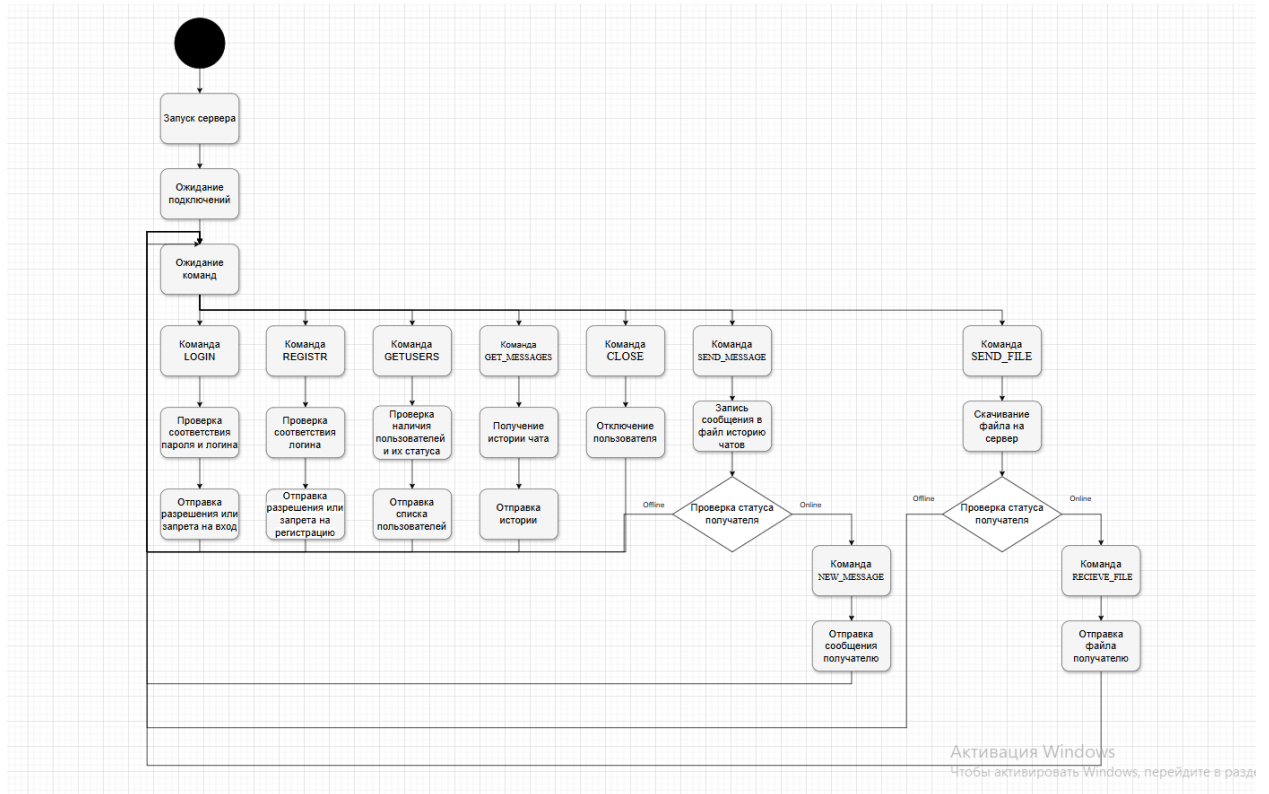


Рисунок 22 – UML - диаграмма состояний сервера

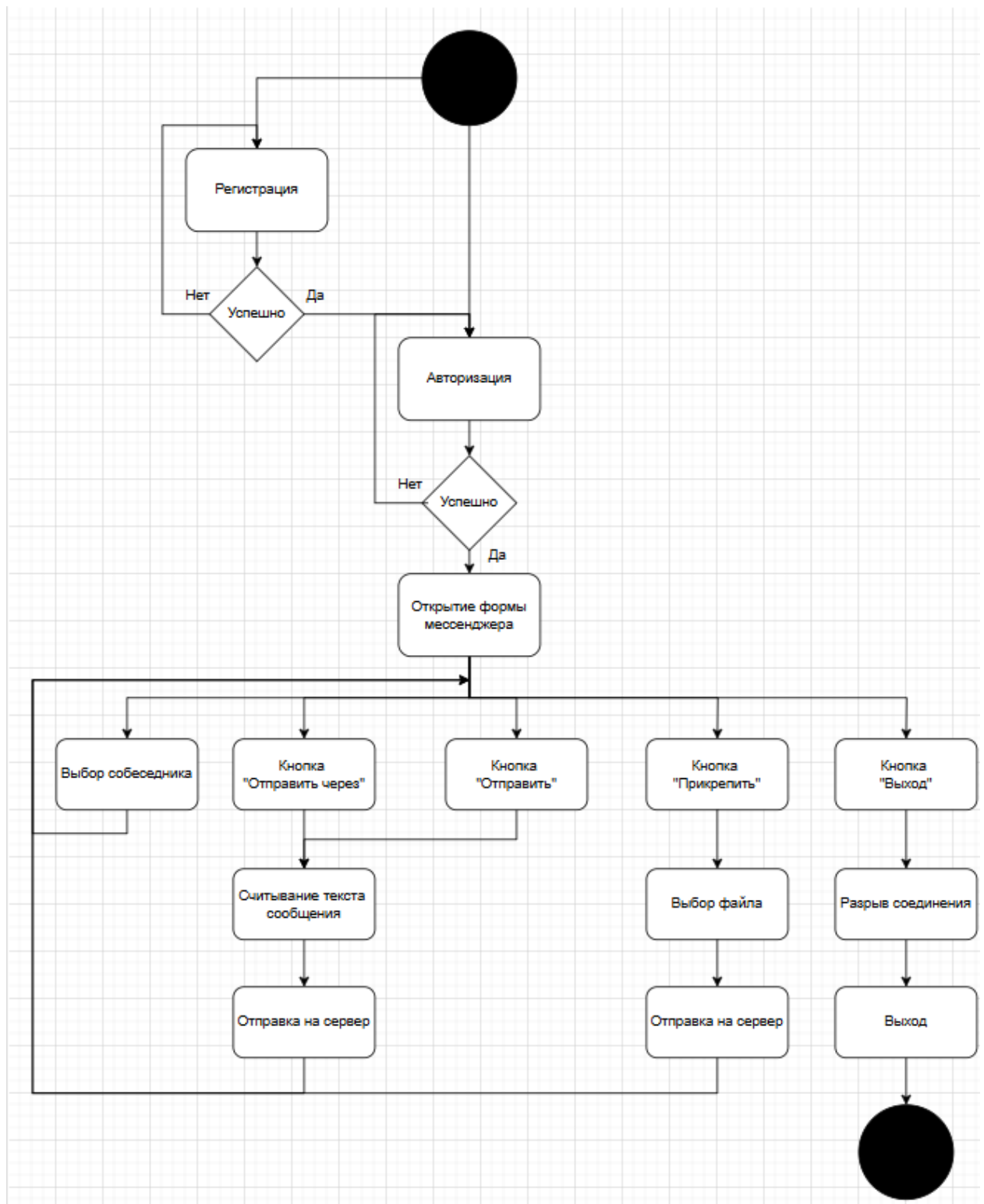


Рисунок 23 – UML - диаграмма состояний клиента