

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»

А.Р. Федюнин
Чубаров И.В.

Выполнил:
студент группы 22BBB1
Чубаров И.В.

Приняли:
к.т.н., доцент Федюнин Р.Н.

2024

✓

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

профессор М. А. Митрохин

« ____ » ____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Архитектура распределенных систем

Студенту Губарову Игорю Валерьевичу

Группа 72 ВВВ-1

Тема проекта Система передачи данных сообщений

Исходные данные (технические требования) на проектирование

Разработать программную систему клиент-серверной архитектуры, функционирующую посредством разработанного протокола передачи данных. Функционал включает в себя: индивидуальное взаимодействие на сервере посредством клиента передачу текстовых сообщений между пользователями, передачу файлов.

Функции сервера: Организация взаимодействия между клиентами (передача сообщений, файлов).
Взаимодействие клиента: графический оконный интерфейс пользователя, взаимодействие с сервером.

Требования к языку программирования: C++
Допускается расширение функционала системы: авторизация пользователей, передача информации, поддержка и СУБД для хранения информации в пользователе.

Объем работы по курсу

Объем работы по курсу

1. Расчетная часть

- 1) Разработка протокола взаимодействия программ
- 2) Разработка программ
- 3) Следствие преобразования и протоколов

2. Графическая часть

Не менее 5 UML диаграмм, описывающих работу системы

3. Экспериментальная часть

Тестирование системы программ в локальной вычислительной сети.

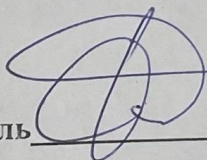
Срок выполнения проекта по разделам

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

Дата выдачи задания "24" сентября

Дата защиты проекта " "

Руководитель



Сердюк Р.Н.

Задание получил

"24" сентября

20__ г.

Студент

Чударов Чударов И.В.

Содержание

Введение	5
Основная часть	6
1. Постановка задачи.....	6
2. Теоритические сведения.....	7
3. Выбор решения.....	8
4. Описание Протокола.....	9
5. Описание программы.....	12
6. Руководство пользователя.....	18
Заключение	20
Список используемых источников.....	22
Приложение А.	23
Листинг программы клиента.....	23
Листинг программы сервера	35
Приложение В.....	53

Введение

На сегодняшний день технологии передачи данных и эффективные коммуникации стали неотъемлемой частью жизни общества. В условиях бурного прогресса в IT-сфере мессенджеры прочно укрепились в качестве ключевого инструмента для общения, предоставляя пользователям функционал для мгновенного обмена текстовыми сообщениями, файлами и медиа-контентом.

Целью данной курсовой работы является разработка программного обеспечения для обмена текстовыми сообщениями и файлами в пределах локальной сети. Ключевая задача проекта — создание интуитивно понятного и многофункционального мессенджера, отвечающего основным запросам пользователей на коммуникацию. Разрабатываемое приложение призвано оптимизировать взаимодействие между пользователями и предложить стабильный канал для передачи данных.

В рамках проекта будет проанализирован ряд вопросов, связанных с созданием приложения: архитектурные решения, поддержка многопользовательского режима, система регистрации и проверки подлинности пользователей, а также механизмы обработки и пересылки файлов. Приоритетными направлениями станут разработка эргономичного пользовательского интерфейса и защита передаваемых данных.

Итогом проекта станет работоспособное приложение, обеспечивающее комфортное и защищенное взаимодействие пользователей в локальной сети. Приобретенные знания и навыки в области проектирования и создания мессенджеров могут быть применены не только для последующего развития данной программы, но и для создания схожих продуктов в иных сферах.

Основная часть

1. Постановка задачи

Основной задачей проекта является разработка клиент-серверной системы для передачи текстовых сообщений и файлов в пределах локальной сети. Проектирование предполагает создание специализированного протокола связи, через который клиентские приложения будут взаимодействовать с сервером.

В рамках решения поставленной задачи планируется:

Реализовать клиентскую часть с графическим интерфейсом пользователя (GUI), обеспечивающим функции отправки сообщений и файлов.

Разработать серверное приложение с консольным интерфейсом, которое позволит отслеживать активность и обмен данными между подключенными клиентами.

Обеспечить надежность передачи как текстовой информации, так и файлов.

Технической основой для реализации послужит язык программирования C#, а итоговое приложение будет обладать кроссплатформенностью, то есть сможет функционировать в различных операционных средах.

2. Теоритические сведения

Ключевым условием взаимодействия программ в интернете является использование специальных наборов правил, известных как протоколы. Эти стандарты регламентируют процедуру передачи информации между устройствами, устанавливая четкий алгоритм отправки и получения данных.

Фундаментальную роль в работе глобальной сети играет протокол ТСП/IP, объединяющий два компонента. ТСП (Протокол управления передачей) контролирует процесс коммуникации: разделяет информацию на пакеты, обеспечивает их отправку, проверяет целостность доставки и отвечает за правильную последовательность сборки на стороне приемника. Протокол IP (Интернет-протокол) решает задачу маршрутизации этих пакетов. Каждый из них содержит IP-адреса источника и назначения, что позволяет сетевым маршрутизаторам выбирать для них оптимальный путь следования.

Для функционирования пользовательских приложений, таких как программы для обмена сообщениями или почтовые службы, применяются протоколы более высокого уровня. Они регулируют конкретные процессы, включая передачу данных и процедуры аутентификации, как, например, запрос пароля при входе в почтовый ящик, который выполняется по строго установленному сценарию.

Благодаря слаженной работе всего многообразия протоколов становится возможным согласованное и эффективное взаимодействие между программным обеспечением и оборудованием в глобальной сети, что лежит в основе удобства использования современных интернет-технологий.

3. Выбор решения

Графический интерфейс клиентского приложения был создан с применением технологии WinForms. Данная платформа предлагает эффективные средства для создания программ под ОС Windows, обеспечивая простую интеграцию таких компонентов, как поля ввода, кнопки управления и списки, в функциональную архитектуру мессенджера.

Коммуникация между серверной и клиентской частью организована посредством протокола TCP/IP, обеспечивающего надежную доставку информации за счет системы подтверждений и механизма повторной передачи при возникновении сбоев. Этот протокол нашел широкое применение в таких распространенных сервисах, как HTTP, FTP, SMTP и Telnet. Работа по TCP предполагает предварительное установление соединения, в ходе которого сервер переходит в режим ожидания входящих запросов, а клиент начинает сеанс связи, отправляя синхронизирующий пакет с порядковым номером.

В основе сетевой организации лежит конфигурация типа "звезда", где все клиенты подключаются к центральному узлу в лице сервера. Для защиты передаваемой информации был внедрен механизм проверки подлинности и разграничения прав доступа пользователей, что позволяет управлять доступом к системе и блокировать несанкционированные подключения.

В целях улучшения пользовательского опыта в приложение были добавлены дополнительные возможности, включая индикацию сетевого статуса пользователей.

4. Описание Протокола

Протокол функционирует как единый стандарт взаимодействия, одинаково понимаемый и клиентом, и сервером. Именно через этот набор правил осуществляется их согласованный обмен данными.

Реализация протокола присутствует как в клиентском приложении (файл `client.cs`), так и в серверной части (файл `Program.cs`). Каждая команда, включая `REGISTER`, `LOGIN`, `GET_USERS`, `SEND_MESSAGE`, `GET_MESSAGES`, `NEW_MESSAGE`, `SEND_FILE`, `RECIEVE_FILE` и `CLOSE`, является структурным элементом протокола и выполняет строго отведенную ей роль в процессе обмена сообщениями. Единообразная реализация на обеих сторонах гарантирует стабильность соединения и корректность взаимодействия.

Далее следует описание функционала ключевых команд.

Команда `SEND_MESSAGE` отвечает за пересылку текстовых сообщений целевому пользователю. Запрос включает логин отправителя, логин получателя и текст сообщения. Сервер обрабатывает этот запрос, извлекая указанные данные. Если адресат находится в сети, сервер сохраняет сообщение в файл истории и немедленно пересылает его получателю в виде команды `NEW_MESSAGE`, содержащей логин отправителя и текст. Если пользователь недоступен, сообщение просто архивируется в истории чата.

Команда `GET_MESSAGES` предоставляет пользователю доступ к истории диалога с другим участником. Клиент формирует запрос, указывая свой логин и логин собеседника. Сервер находит соответствующий файл истории, проверяет его наличие и возвращает содержимое текстовой строкой. В случае отсутствия файла клиенту направляется уведомление об отсутствии переписки. Этот подход позволяет пользователям легко возобновлять предыдущие беседы.

Команда `SEND_FILE` обеспечивает передачу файлов между пользователями. Клиент отправляет на сервер метаданные и содержимое

файла в бинарном виде. Сервер сохраняет полученный файл в хранилищах, связанных как с отправителем, так и с получателем.

Команда RECIEVE_FILE используется сервером для пересылки файла клиенту в ответ на успешное выполнение команды SEND_FILE. После сохранения файла на сервере проверяется статус получателя. Если он онлайн, файл передается ему немедленно; в противном случае файл остается на сервере для последующей доставки. В перспективе функционал можно расширить для организации отложенной отправки пользователям, находившимся в офлайне в момент передачи.

Команда LOGIN предназначена для аутентификации пользователей. Клиент передает логин и пароль, которые сервер сверяет с записями в файле users.txt. При совпадении учетных данных предоставляется доступ к системе.

Команда REGISTER служит для создания новой учетной записи. Клиент отправляет желаемые логин и пароль. Сервер проверяет уникальность логина в файле users.txt. Если дубликат не найден, регистрация считается успешной, и пользователь получает права на вход в мессенджер.

Команда GET_USERS обновляет список контактов в клиентском приложении. По запросу клиента сервер возвращает перечень всех зарегистрированных пользователей с указанием их текущего статуса (в сети/не в сети). Для поддержания актуальности информации этот список автоматически обновляется каждые 5 секунд.

Команда CLOSE выполняет удаление пользователя из списка активных сессий. Она активируется при выходе пользователя из системы.



Рисунок 2 – UML Диаграмма состояний

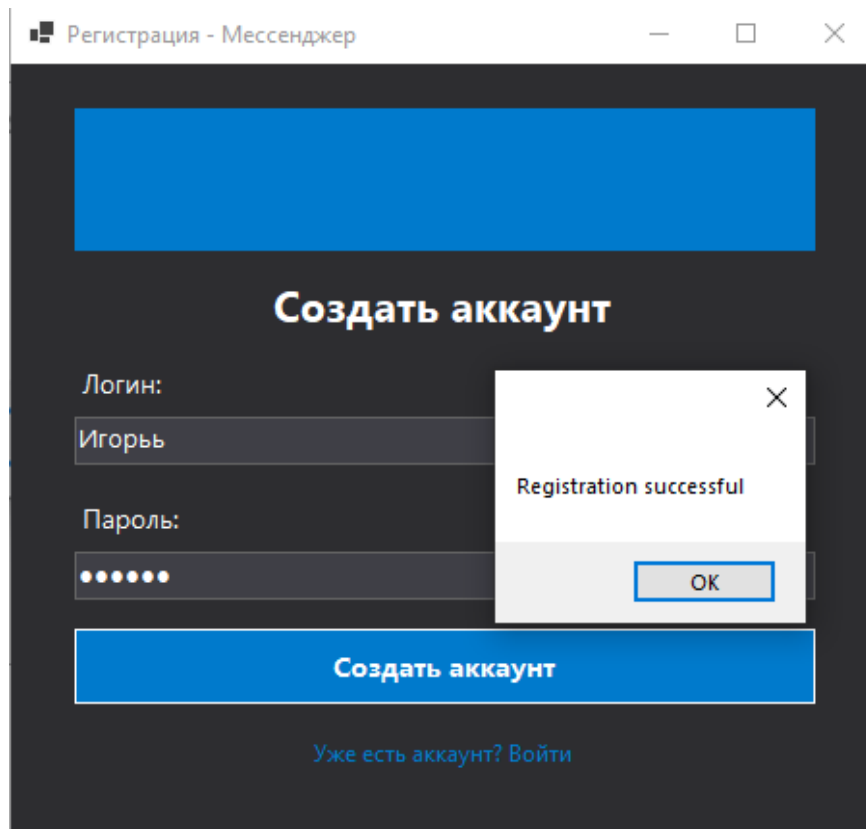


Рисунок 4 – Авторизация и аутентификация

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.

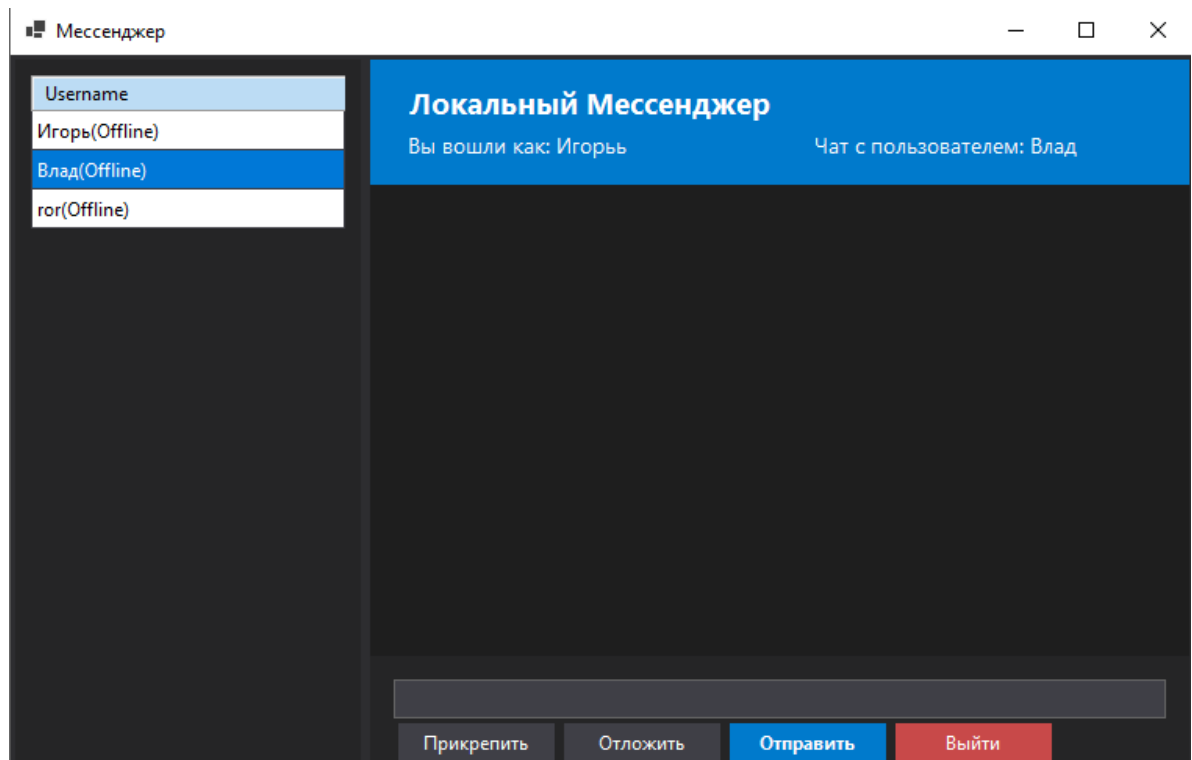


Рисунок 5 – Главное окно

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

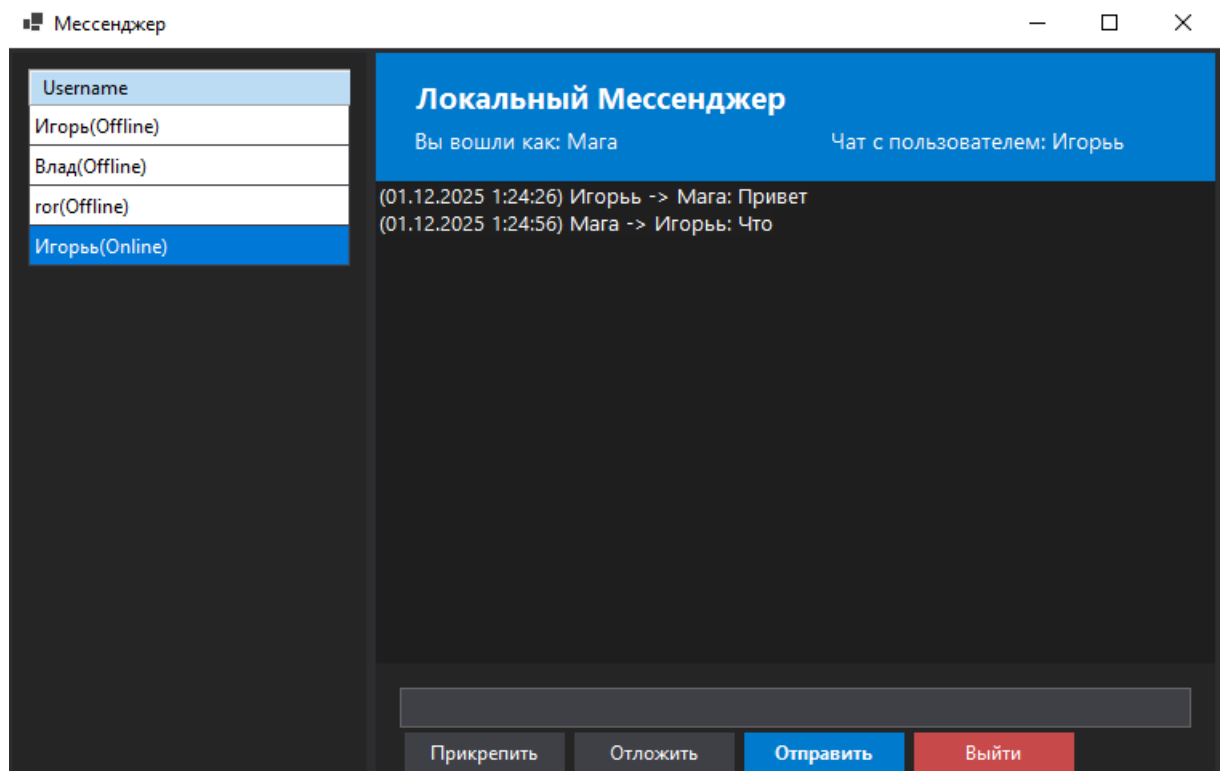


Рисунок 6 – Окно отправителя

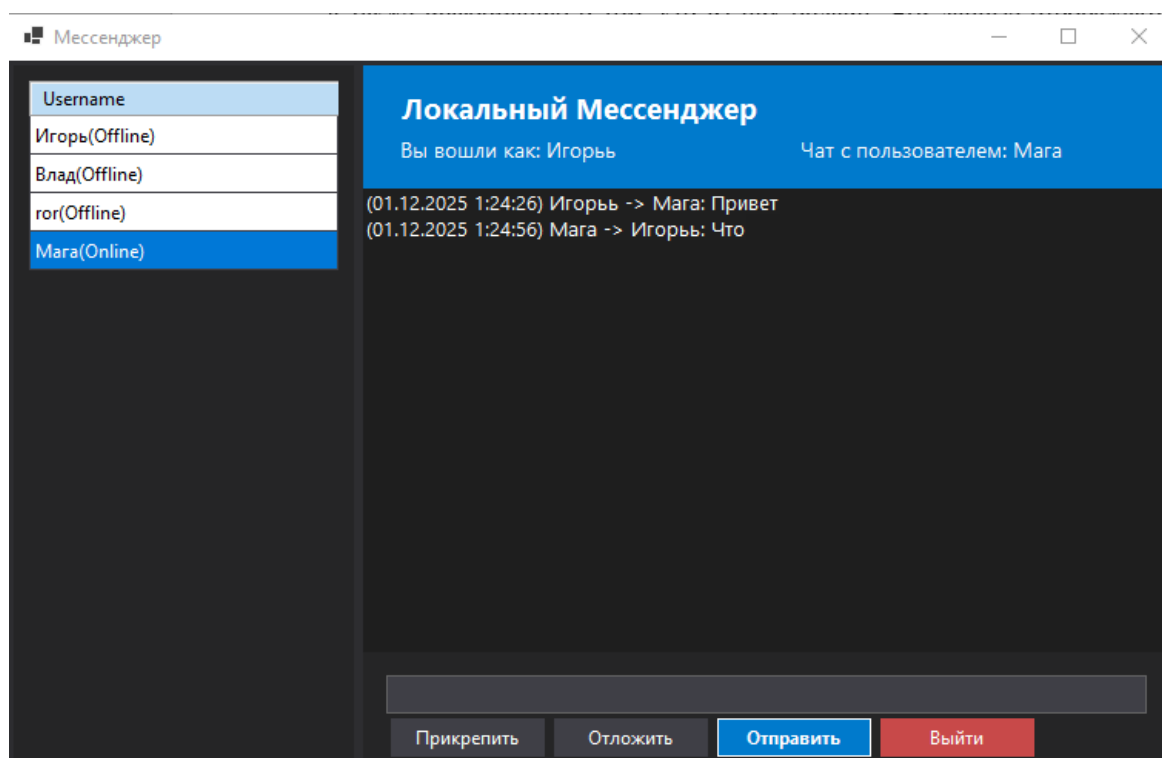


Рисунок 7 – Окно получателя

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

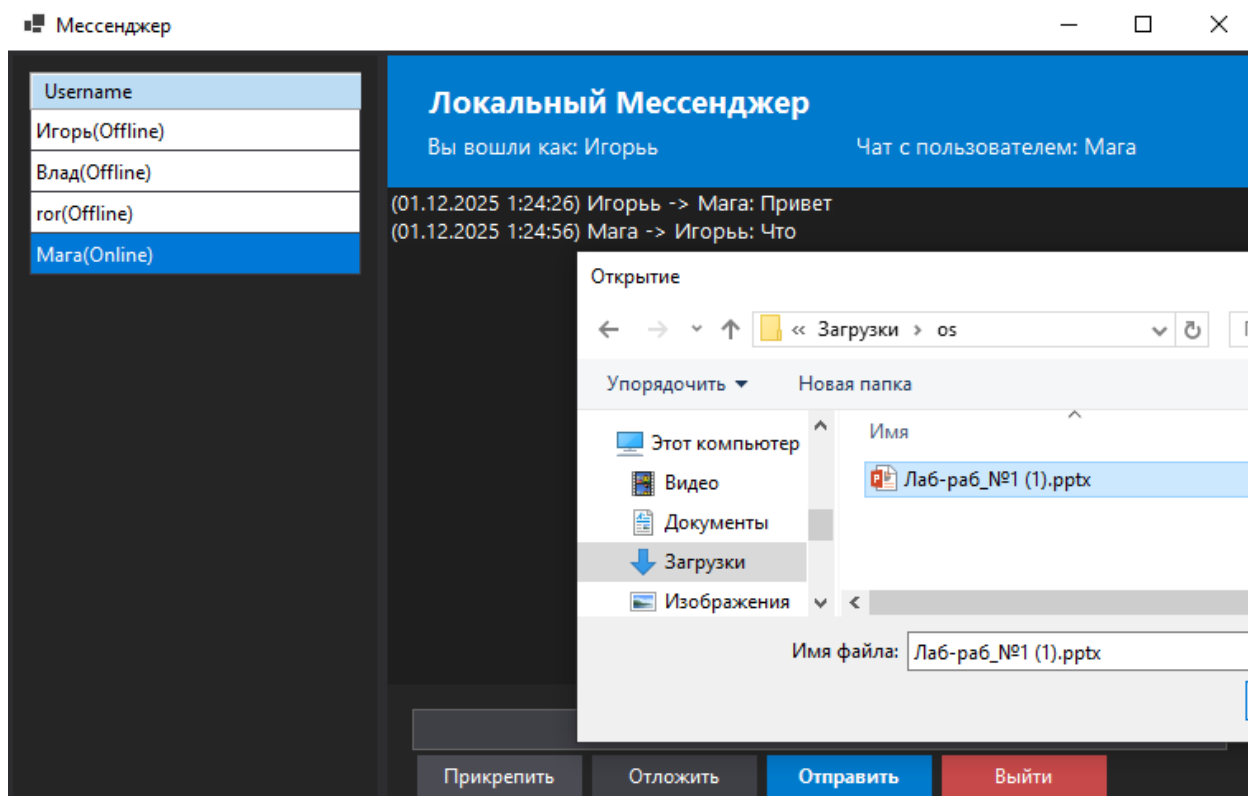


Рисунок 8 – Выбор файла для отправки

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

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.

Если 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 3 секунд сообщение будет автоматически отправлено.

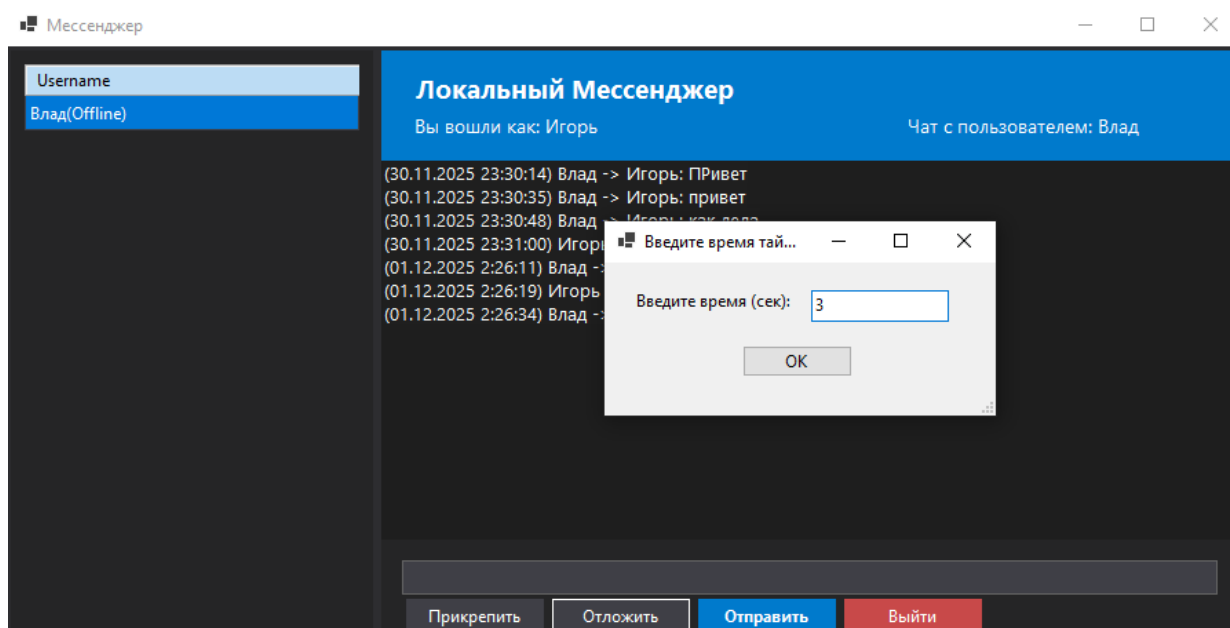


Рисунок 9– Отправка сообщения с таймером

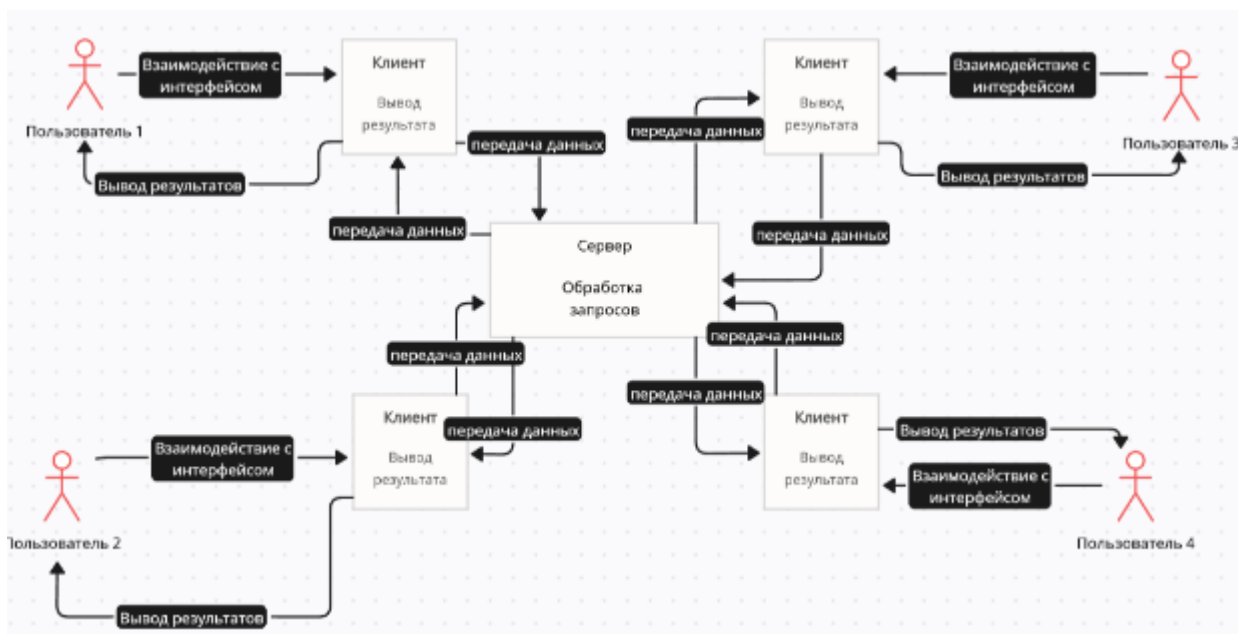


Рисунок 14 – UML - диаграмма развёртывания

UML-диаграмма состояний сервера, отображающая последовательность событий, происходящих на сервере в процессе обработки команд, поступающих от клиентов.

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.

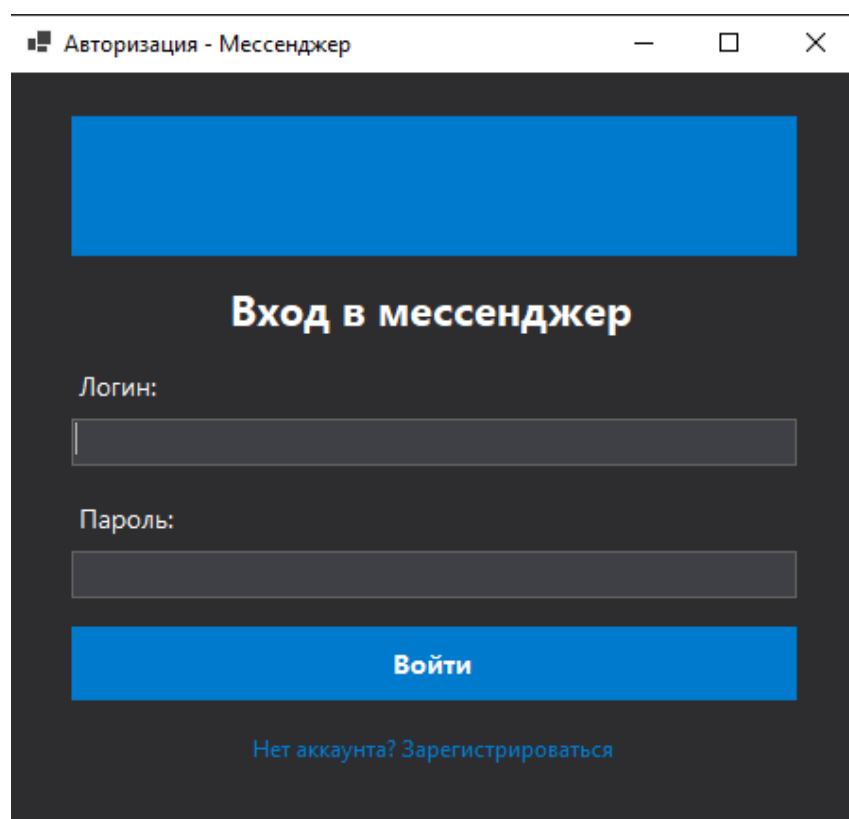


Рисунок 10 – Форма аутентификации и авторизации

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

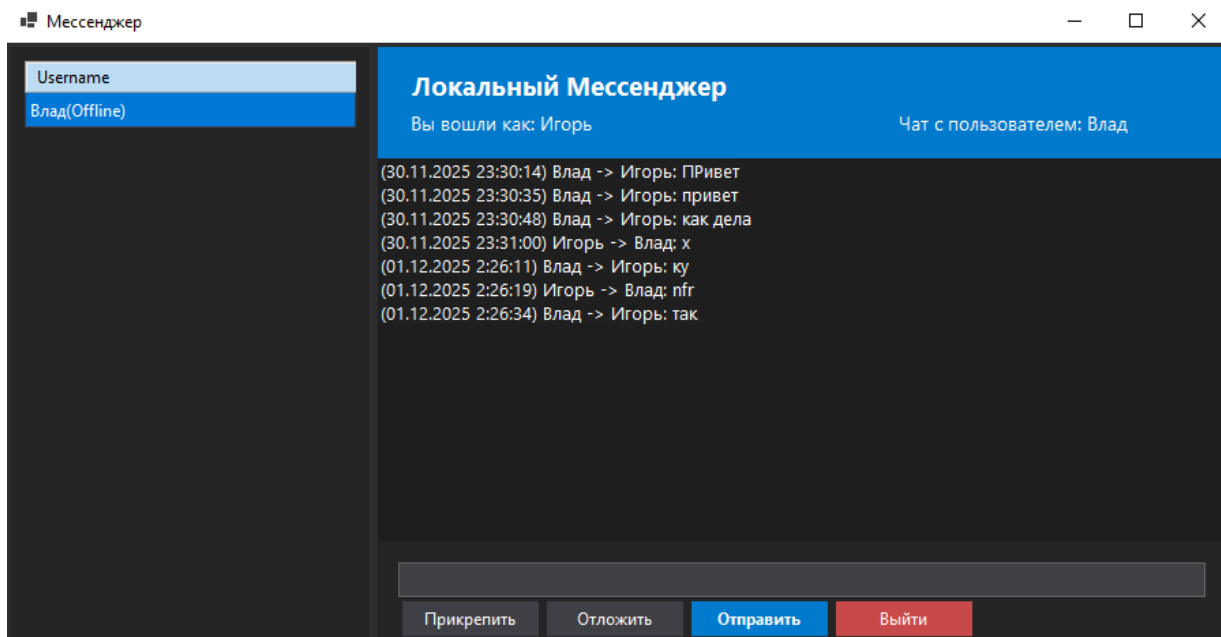


Рисунок 11 – Выбор пользователя и ввод сообщения

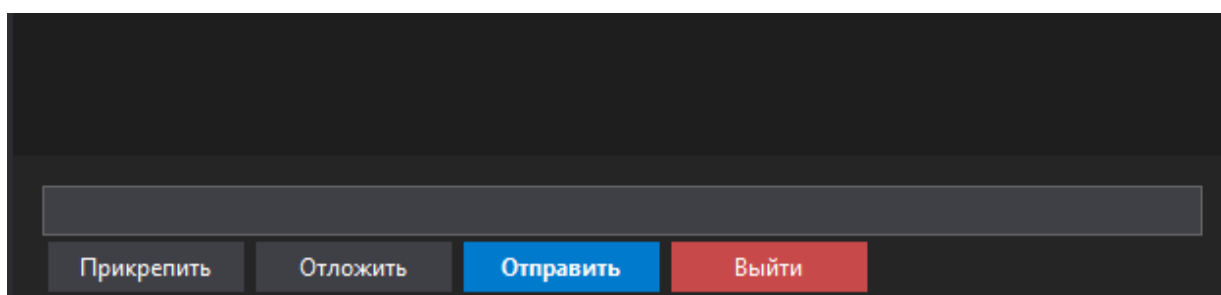


Рисунок 12 – Ввод

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 представлена UML-диаграмма сценариев клиента.

```
[ЛОГ] Исходная дата и время: 01.12.2025 2:26:34
[ЛОГ] Сообщение добавлено с датой: 01.12.2025 2:26:34
Сообщение: System.Collections.Generic.List`1[Server.Program+Message]
[ЛОГ] Сообщений загружено: 8
[ЛОГ] Сообщение: Отправитель: Влад, Получатель: Игорь, Сообщение: Привет, Дата: 30.11.2025 23:30:14
[ЛОГ] Сообщение: Отправитель: Влад, Получатель: Игорь, Сообщение: привет, Дата: 30.11.2025 23:30:35
[ЛОГ] Сообщение: Отправитель: Влад, Получатель: Игорь, Сообщение: как дела, Дата: 30.11.2025 23:30:48
[ЛОГ] Сообщение: Отправитель: Игорь, Получатель: Влад, Сообщение: х, Дата: 30.11.2025 23:31:00
[ЛОГ] Сообщение: Отправитель: Влад, Получатель: Чат, Сообщение: -, Дата: 01.12.2025 2:25:55
[ЛОГ] Сообщение: Отправитель: Влад, Получатель: Игорь, Сообщение: ку, Дата: 01.12.2025 2:26:11
[ЛОГ] Сообщение: Отправитель: Игорь, Получатель: Влад, Сообщение: nfr, Дата: 01.12.2025 2:26:19
[ЛОГ] Сообщение: Отправитель: Влад, Получатель: Игорь, Сообщение: так, Дата: 01.12.2025 2:26:34
[ЛОГ] Количество сообщений, полученных от клиента: 7
```

Рисунок 13 – Работа сервера

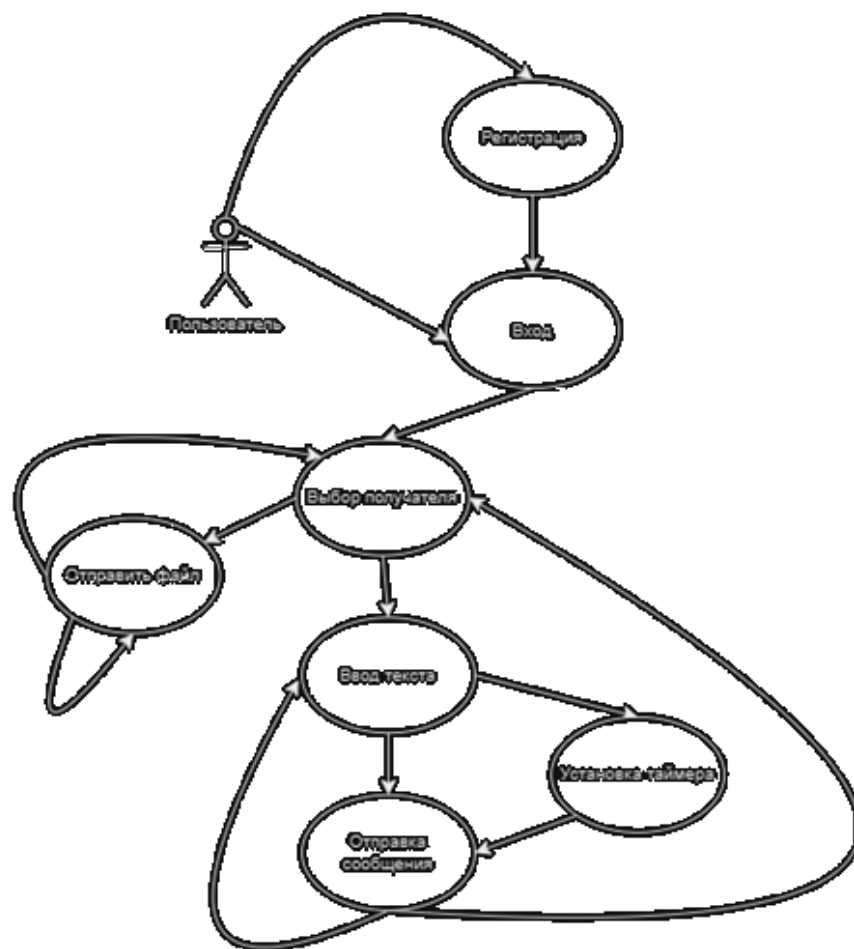


Рисунок 14 – UML-Диаграмма сценариев

Заключение

В ходе данного курсового проекта был создан мессенджер, гарантирующий продуктивное и защищенное общение пользователей в пределах локальной сети. Ключевым результатом стало построение клиент-серверного решения, обеспечивающего обмен текстовыми сообщениями и файлами через протокол TCP/IP, гарантирующий устойчивую связь.

В процессе создания приложения успешно решен ряд значимых задач: спроектирована архитектура системы, разработан специализированный протокол передачи данных, внедрены процедуры авторизации и

аутентификации. Существенное внимание было уделено обеспечению комфортного взаимодействия конечных пользователей с программой.

Клиентское приложение, созданное на платформе WinForms, получило интуитивно понятный интерфейс, а серверная часть с консольным управлением предоставила инструменты для администрирования и отслеживания состояния системы, что повысило гибкость эксплуатации всего программного комплекса.

Разработанный протокол обмена данными составил основу для корректной передачи сообщений и файлов между клиентами. Его отказоустойчивость обеспечена механизмами обнаружения ошибок и повторной отправки информации, что критически важно для бесперебойного функционирования системы в среде локальной сети.

Приобретенный опыт создания мессенджера с поддержкой текстовой и файловой коммуникации может быть использован для последующего развития данного продукта, а также при реализации похожих решений в других областях. В перспективе приложение может быть усовершенствовано за счет добавления поддержки мультимедийных сообщений, усовершенствованных методов защиты информации и расширенного функционала для системных администраторов.

В итоге, проделанная работа позволила получить ценные знания и практические умения в области создания многопользовательских приложений на основе клиент-серверной архитектуры, а также углубить понимание принципов передачи данных и сетевых взаимодействий.

Список используемых источников

1. Официальная документация Microsoft по .NET и C#:
<https://learn.microsoft.com/ru-ru/>

Приложение А.

Листинг программы клиента

Avtorisation.Designer.cs

```
namespace client
{
    partial class Avtorisation
    {
        private System.ComponentModel.IContainer components = null;
        private System.Windows.Forms.TableLayoutPanel tableLayoutPanel;
        private System.Windows.Forms.Label lblTitle;
        private System.Windows.Forms.Label lblUser;
        private System.Windows.Forms.Label lblPass;
        private System.Windows.Forms.TextBox textBox2;
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Panel panel1;
        private System.Windows.Forms.PictureBox pictureBox1;

        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null)) { components.Dispose(); }
            base.Dispose(disposing);
        }

        private void InitializeComponent()
        {
            tableLayoutPanel = new System.Windows.Forms.TableLayoutPanel();
            panel1 = new System.Windows.Forms.Panel();
            pictureBox1 = new System.Windows.Forms.PictureBox();
            lblTitle = new System.Windows.Forms.Label();
            lblUser = new System.Windows.Forms.Label();
            textBox2 = new System.Windows.Forms.TextBox();
            lblPass = new System.Windows.Forms.Label();
            textBox1 = new System.Windows.Forms.TextBox();
            button1 = new System.Windows.Forms.Button();
            button2 = new System.Windows.Forms.Button();
            tableLayoutPanel.SuspendLayout();
            panel1.SuspendLayout();
        }
    }
}
```

```

        ((System.ComponentModel.ISupportInitialize)pictureBox1).BeginInit();
        SuspendLayout();
        //
        // tableLayoutPanel
        //
        tableLayoutPanel.BackColor = System.Drawing.Color.FromArgb(45, 45,
48);
        tableLayoutPanel.ColumnCount = 1;
        tableLayoutPanel.ColumnStyles.Add(new
System.Windows.Forms.ColumnStyle(System.Windows.Forms.SizeType.Percent,
100F));
        tableLayoutPanel.Controls.Add(panel1, 0, 0);
        tableLayoutPanel.Controls.Add(lblTitle, 0, 1);
        tableLayoutPanel.Controls.Add(lblUser, 0, 2);
        tableLayoutPanel.Controls.Add(textBox2, 0, 3);
        tableLayoutPanel.Controls.Add(lblPass, 0, 4);
        tableLayoutPanel.Controls.Add(textBox1, 0, 5);
        tableLayoutPanel.Controls.Add(button1, 0, 6);
        tableLayoutPanel.Controls.Add(button2, 0, 7);
        tableLayoutPanel.Dock = System.Windows.Forms.DockStyle.Fill;
        tableLayoutPanel.Location = new System.Drawing.Point(0, 0);
        tableLayoutPanel.Name = "tableLayoutPanel";
        tableLayoutPanel.Padding = new System.Windows.Forms.Padding(30, 20,
30, 20);
        tableLayoutPanel.RowCount = 8;
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
80F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
50F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
30F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
40F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
30F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
40F));

```

```

        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
45F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Absolute,
45F));
        tableLayoutPanel.RowStyles.Add(new
System.Windows.Forms.RowStyle(System.Windows.Forms.SizeType.Percent,
100F));
        tableLayoutPanel.Size = new System.Drawing.Size(450, 400);
        tableLayoutPanel.TabIndex = 0;
        //
        // panel1
        //
        panel1.BackColor = System.Drawing.Color.FromArgb(0, 122, 204);
        panel1.Controls.Add(pictureBox1);
        panel1.Dock = System.Windows.Forms.DockStyle.Fill;
        panel1.Location = new System.Drawing.Point(33, 23);
        panel1.Name = "panel1";
        panel1.Size = new System.Drawing.Size(384, 74);
        panel1.TabIndex = 8;
        //
        // pictureBox1
        //
        pictureBox1.Dock = System.Windows.Forms.DockStyle.Fill;
        pictureBox1.Location = new System.Drawing.Point(0, 0);
        pictureBox1.Name = "pictureBox1";
        pictureBox1.Size = new System.Drawing.Size(384, 74);
        pictureBox1.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.CenterImage;
        pictureBox1.TabIndex = 0;
        pictureBox1.TabStop = false;
        //
        // lblTitle
        //
        lblTitle.Dock = System.Windows.Forms.DockStyle.Fill;
        lblTitle.Font = new System.Drawing.Font("Segoe UI", 16F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point);
        lblTitle.ForeColor = System.Drawing.Color.White;
        lblTitle.Location = new System.Drawing.Point(33, 100);
        lblTitle.Name = "lblTitle";
        lblTitle.Size = new System.Drawing.Size(384, 50);
        lblTitle.TabIndex = 0;
        lblTitle.Text = "Вход в мессенджер";

```

```

lblTitle.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
//
// lblUser
//
lblUser.Dock = System.Windows.Forms.DockStyle.Fill;
lblUser.Font = new System.Drawing.Font("Segoe UI", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
lblUser.ForeColor = System.Drawing.Color.White;
lblUser.Location = new System.Drawing.Point(33, 150);
lblUser.Name = "lblUser";
lblUser.Size = new System.Drawing.Size(384, 30);
lblUser.TabIndex = 1;
lblUser.Text = "Логин:";
lblUser.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
//
// textBox2
//
textBox2.BackColor = System.Drawing.Color.FromArgb(63, 63, 70);
textBox2.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
textBox2.Dock = System.Windows.Forms.DockStyle.Fill;
textBox2.Font = new System.Drawing.Font("Segoe UI", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
textBox2.ForeColor = System.Drawing.Color.White;
textBox2.Location = new System.Drawing.Point(33, 183);
textBox2.Name = "textBox2";
textBox2.Size = new System.Drawing.Size(384, 25);
textBox2.TabIndex = 2;
//
// lblPass
//
lblPass.Dock = System.Windows.Forms.DockStyle.Fill;
lblPass.Font = new System.Drawing.Font("Segoe UI", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
lblPass.ForeColor = System.Drawing.Color.White;
lblPass.Location = new System.Drawing.Point(33, 220);
lblPass.Name = "lblPass";
lblPass.Size = new System.Drawing.Size(384, 30);
lblPass.TabIndex = 3;
lblPass.Text = "Пароль:";
lblPass.TextAlign = System.Drawing.ContentAlignment.MiddleLeft;
//
// textBox1
//
textBox1.BackColor = System.Drawing.Color.FromArgb(63, 63, 70);

```

```

textBox1.BorderStyle = System.Windows.Forms.BorderStyle.FixedSingle;
textBox1.Dock = System.Windows.Forms.DockStyle.Fill;
textBox1.Font = new System.Drawing.Font("Segoe UI", 10F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
textBox1.ForeColor = System.Drawing.Color.White;
textBox1.Location = new System.Drawing.Point(33, 253);
textBox1.Name = "textBox1";
textBox1.Size = new System.Drawing.Size(384, 25);
textBox1.TabIndex = 4;
textBox1.UseSystemPasswordChar = true;
//
// button1
//
button1.BackColor = System.Drawing.Color.FromArgb(0, 122, 204);
button1.Dock = System.Windows.Forms.DockStyle.Fill;
button1.FlatAppearance.BorderSize = 0;
button1.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
button1.Font = new System.Drawing.Font("Segoe UI", 10F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point);
button1.ForeColor = System.Drawing.Color.White;
button1.Location = new System.Drawing.Point(33, 293);
button1.Name = "button1";
button1.Size = new System.Drawing.Size(384, 39);
button1.TabIndex = 5;
button1.Text = "Войти";
button1.UseVisualStyleBackColor = false;
button1.Click += button1_Click;
//
// button2
//
button2.BackColor = System.Drawing.Color.Transparent;
button2.Dock = System.Windows.Forms.DockStyle.Fill;
button2.FlatAppearance.BorderSize = 0;
button2.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
button2.Font = new System.Drawing.Font("Segoe UI", 9F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point);
button2.ForeColor = System.Drawing.Color.FromArgb(0, 122, 204);
button2.Location = new System.Drawing.Point(33, 338);
button2.Name = "button2";
button2.Size = new System.Drawing.Size(384, 39);
button2.TabIndex = 6;
button2.Text = "Нет аккаунта? Зарегистрироваться";
button2.UseVisualStyleBackColor = false;
button2.Click += button2_Click;

```

```

//
// Avtorisation
//
ClientSize = new System.Drawing.Size(450, 400);
Controls.Add(tableLayoutPanel);
Name = "Avtorisation";
StartPosition = System.Windows.Forms.FormStartPosition.CenterScreen;
Text = "Авторизация - Мессенджер";
tableLayoutPanel.ResumeLayout(false);
tableLayoutPanel.PerformLayout();
panel1.ResumeLayout(false);
((System.ComponentModel.ISupportInitialize)pictureBox1).EndInit();
ResumeLayout(false);
}
}
}

```

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.listBox1.ItemsAdd(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

```

```

senderUsername);
    string saveDirectory = Path.Combine("Downloads",
    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();
}
}
}

```

Program.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
    }
}
}

```

RequestCode.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";
    }
}

```

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";
    }
}

```

ServerApp

Form1.Designer.cs

```
namespace Server
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
        otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(800, 450);
            this.Text = "Form1";
        }

        #endregion
    }
}
```

```

        #endregion
    }
}

```

Program.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($"{{RequestCodes.NEW_MESSAGE}}:{{username}}:{{me
ssage}}");
                await recipientClient.GetStream().WriteAsync(data, 0,
data.Length);
                Console.WriteLine($"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
                });
            }
        }
        catch { }
    }
}

```

```

        });
        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_P
ASSWORD}";
    }

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

```

```

        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_PASS
WORD}}";
        }

        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; }
}
}

```

RequestName.cs

```

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

namespace Server
{
    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";
    }
}

```

ResponseNames.cs

```

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

namespace Server
{
    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";
    }
}

```

}

}

Приложение В.

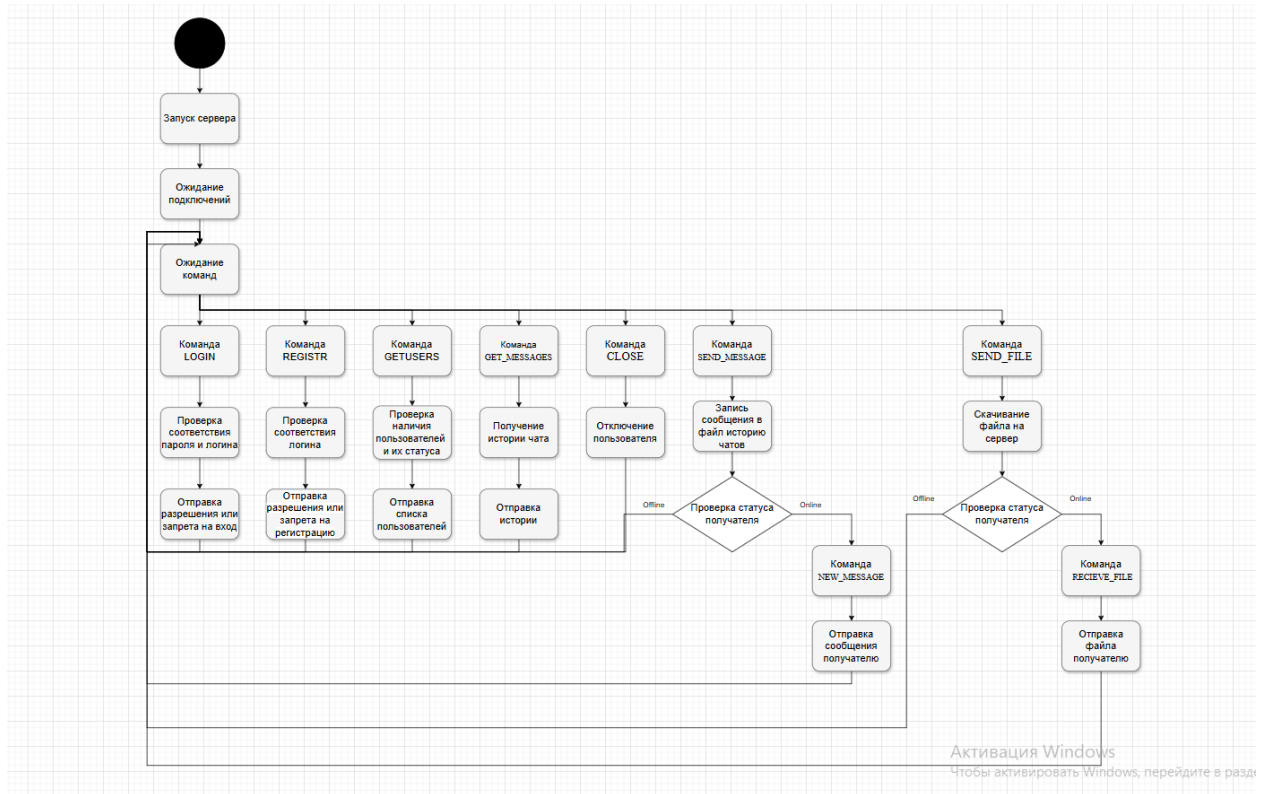


Рисунок 15 – UML - диаграмма состояний сервера

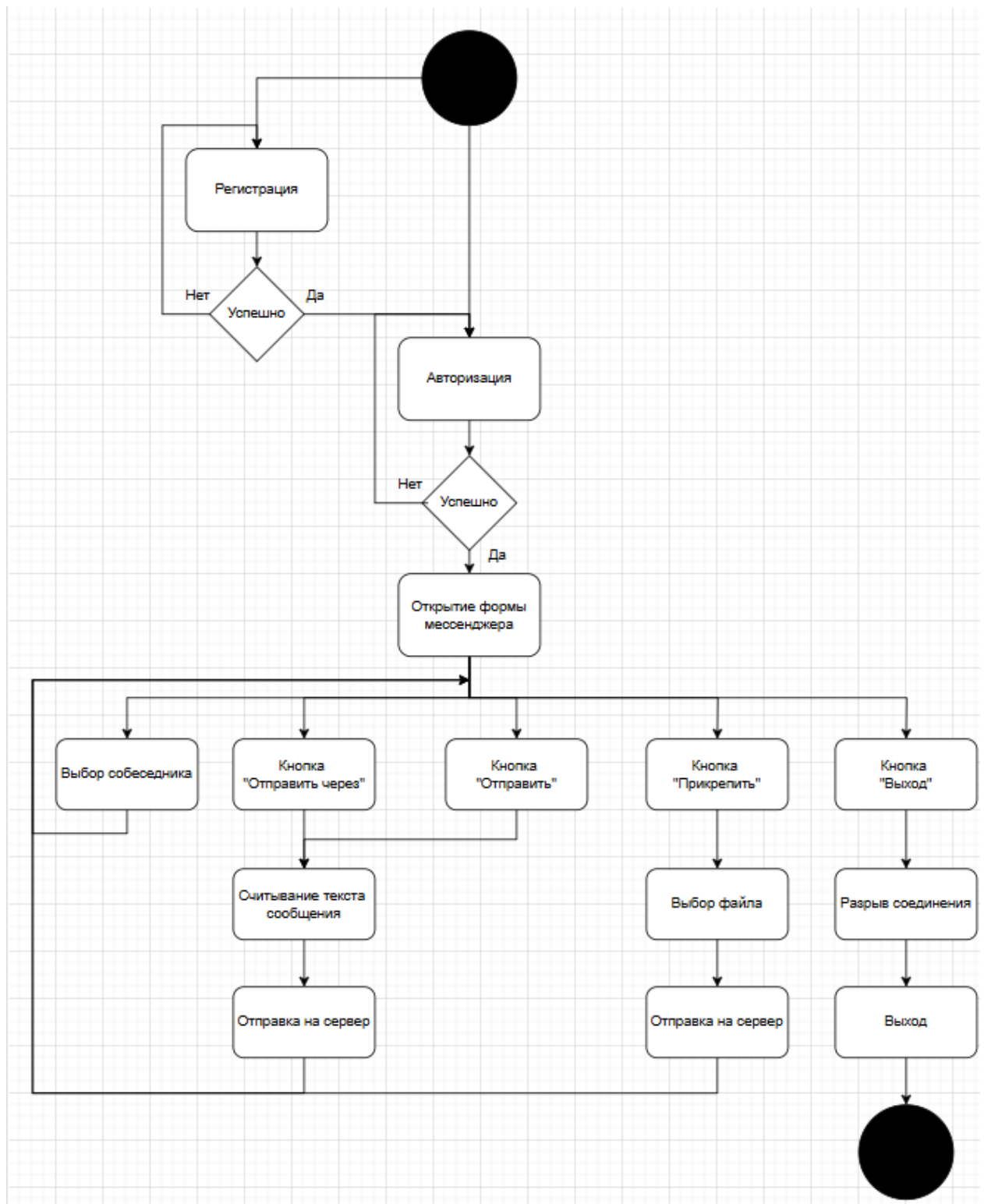


Рисунок 16 – UML - диаграмма состояний клиента