МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных

систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: «Компьютерные системы и сети»

на тему: ***«Сетевой чат с поддержкой шифрования сообщений»***

|  |  |
| --- | --- |
| Выполнила: | ст. гр. 10701121 Панасевич А.А. |
| Приняла: | ст. пр. Белова С.В. |

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники (ФИТР)

Кафедра программного обеспечения информационных

систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине: «Компьютерные системы и сети»

на тему: ***«Сетевой чат с поддержкой шифрования сообщений»***

|  |  |
| --- | --- |
| Выполнила: | ст. гр. 10701121 Панасевич А.А. |
| Приняла: | ст. пр. Белова С.В. |

Минск 2023

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 6](#_Toc134521374)

[1 Описание и анализ предметной области. Обзор аналогов 7](#_Toc134521375)

[2 Постановка задачи. Описание требований к ПО 8](#_Toc134521376)

[3 Проектирование ПО 9](#_Toc134521377)

[3.1 Диаграмма вариантов использования 9](#_Toc134521378)

[3.2 Архитектура ПО. Описание клиентской и серверной части 9](#_Toc134521379)

[3.3 Описание протокола взаимодействия клиента и сервера 10](#_Toc134521380)

[3.4 Проектирование интерфейса пользователя 12](#_Toc134521381)

[3.5 Выбор средств разработки 12](#_Toc134521382)

[4 Реализация ПО 13](#_Toc134521383)

[4.1 Реализация серверной части 13](#_Toc134521384)

[4.2 Реализация клиентской части 14](#_Toc134521385)

[4.3 Описание контрольного примера 15](#_Toc134521386)

[5 Руководство пользователя 17](#_Toc134521387)

[ЗАКЛЮЧЕНИЕ 21](#_Toc134521388)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc134521389)

[ПРИЛОЖЕНИЕ A 23](#_Toc134521390)

[ПРИЛОЖЕНИЕ Б 28](#_Toc134521391)

# **ВВЕДЕНИЕ**

Современный прогресс человечества неразрывно связан с развитием сетевых технологий и интернет-сообщества. В настоящее время сетевые чаты становятся все более популярными, предоставляя людям возможность общаться и обмениваться сообщениями в режиме реального времени, независимо от географического расположения.

Целью данного проекта является разработка сетевого чата, позволяющего пользователям общаться между собой в режиме реального времени. Для реализации этой задачи мы используем язык программирования C# в связке с Windows Forms, что позволяет создать удобный и интуитивно понятный пользовательский интерфейс. Также мы используем технологию сокетов операционной системы Windows, которая обеспечивает сетевое взаимодействие между клиентами и сервером.

Сетевой чат предоставляет возможность клиентам подключаться к серверу, обмениваться зашифрованными текстовыми сообщениями, принимать т расшифровывать сообщения от других участников чата. Серверная часть обрабатывает входящие сообщения, распределяет их между подключенными клиентами и осуществляет передачу сообщений в режиме реального времени.

# **1 Описание и анализ предметной области. Обзор аналогов**

Сетевой чат с шифрованием - это сетевое приложение или сервис, который позволяет пользователям общаться между собой в режиме реального времени через компьютерные сети, при этом применяется шифрование для защиты конфиденциальности сообщений.

В таком чате перед отправкой сообщений они проходят процесс шифрования, который преобразует их в непонятный для посторонних набор символов или данные, которые можно понять и прочитать только с помощью специального ключа или пароля. При получении сообщений они расшифровываются обратно в исходный вид.

Шифрование в сетевом чате обеспечивает дополнительный уровень безопасности и конфиденциальности, так как предотвращает чтение или перехват сообщений третьими лицами. Это особенно важно в случаях, когда отправляемая информация является чувствительной или содержит личные данные.

Для реализации шифрования в сетевом чате могут применяться различные алгоритмы и методы шифрования, такие как симметричное или асимметричное шифрование. В зависимости от выбранного метода, обмен ключами, паролями или сертификатами может использоваться для обеспечения безопасного обмена шифрованными сообщениями между пользователями. Один из аналогов сетевого чата приведен ниже на рис.1.1:

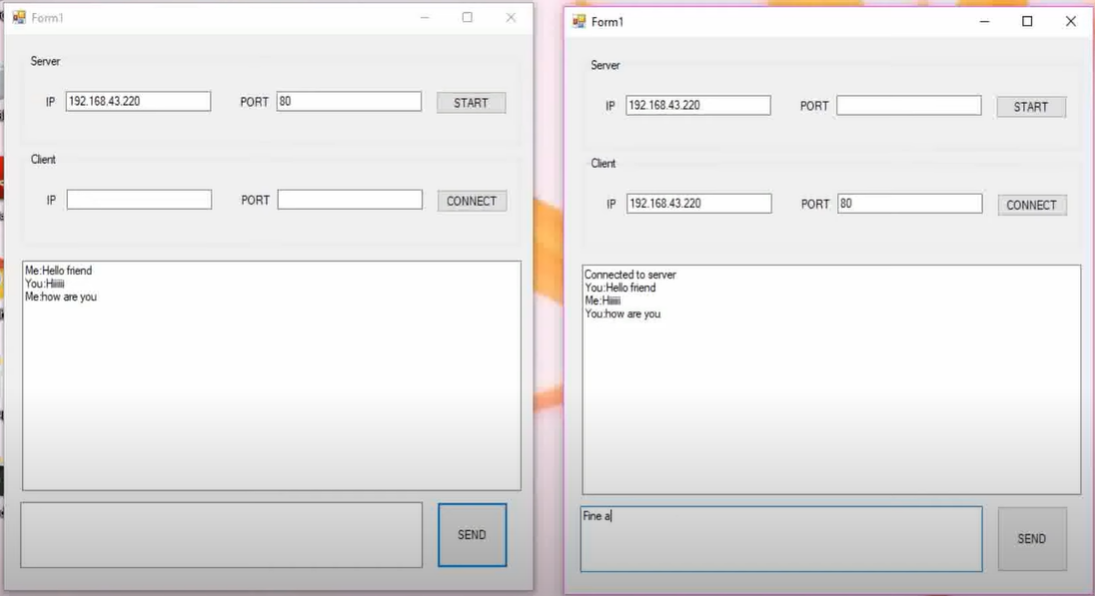


Рисунок 1.1 – Аналог сетевого чата

# **2 Постановка задачи. Описание требований к ПО**

В рамках данного курсового проекта необходимо реализовать сетевой чат с поддержкой шифрования.

Разрабатываемое приложение должно иметь архитектуру клиент-сервер, а также многопользовательский режим.

Клиентская часть должна иметь графический интерфейс. Серверная часть – консольный интерфейс.

Требования к реализации работы:

- операционная система – MS Windows 10;

- среда выполнения – Visual Studio 2022;

- язык программирования – C#;

- сетевое ПО «Чат»;

- графический интерфейс;

- многопользовательский режим.

Взаимодействие клиента и сервера необходимо осуществлять с помощью технологии сокетов.

# **3 Проектирование ПО**

# **3.1 Диаграмма вариантов использования**

Программное обеспечение «Сетевой чат» имеет несколько вариантов использования, полный их перечень представлен ниже на рисунке 3.1.1.

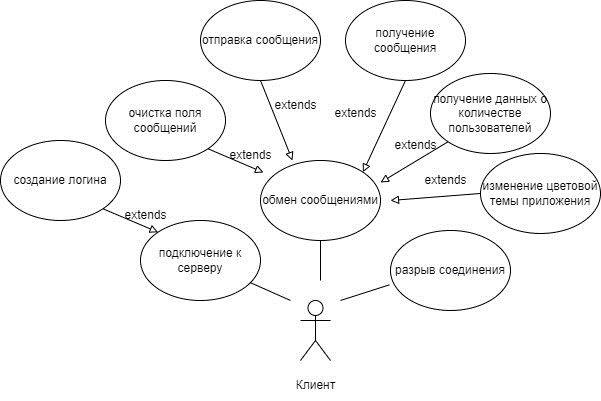


Рисунок 3.1.1 – Диаграмма вариантов использования приложения

# **3.2 Архитектура ПО. Описание клиентской и серверной части**

Архитектура программного обеспечения определяет структуру и организацию программы или системы, включая ее компоненты, их взаимодействие и свойства, которые они предоставляют внешним компонентам. Хорошо спроектированная архитектура программного обеспечения обеспечивает эффективное выполнение задач и удовлетворяет требованиям производительности, безопасности, надежности и другим факторам.

Сетевое приложение является примером распределенной программы, где различные части приложения выполняются на отдельных компьютерах, соединенных в сеть. Каждая часть приложения, называемая компонентом, выполняет определенные функции и взаимодействует с другими компонентами для обмена информацией и достижения общей цели приложения.

Целью хорошо спроектированной архитектуры сетевого приложения является обеспечение надежной и эффективной коммуникации между компонентами, обработка данных в реальном времени и обеспечение безопасности и конфиденциальности при передаче информации по сети. (см. рис. 3.2.1).

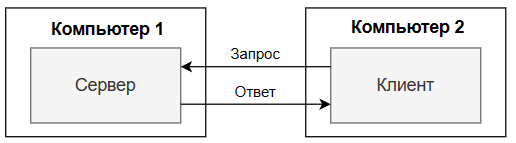


Рисунок 3.2.1 - Взаимодействие частей распределённого приложения

Распределенная программа использует архитектуру клиент-сервер, где различные компоненты приложения взаимодействуют между собой через сеть. Сетевое программное обеспечение, разработанное в рамках текущего курсового проекта, представляет собой многопользовательское приложение, основанное на технологии клиент-сервер. Оно реализует многопоточную модель взаимодействия, где основной поток серверного приложения обрабатывает запросы, а для каждого подключенного клиента создается отдельный поток. Каждый поток существует в течение всего сеанса взаимодействия с клиентом и уничтожается при отключении клиента.

Клиент будет выполнять следующие функции:

1. Ввод данных;
2. Проверка правильности введённых данных;
3. Отправка запросов серверу на выполнение выбранного действия;
4. Принятие ответа от сервера.

Функции сервера:

1. Принятие запроса;
2. Обработка запроса;
3. Отправка ответа;
4. Отображение результата взаимодействия с клиентом.

# **3.3 Описание протокола взаимодействия клиента и сервера**

Для взаимодействия сервера и клиента, после установления связи, в данном приложении используется набор строковых команд, которые вычленяются из переданной строки информации.   
 Для клиента существуют команды:

1. “U” – получение количества подключившихся пользователей;
2. “M” – получение сообщения от других пользователей (либо при случае, когда пользователь единственный в чате, от себя) через сервер.

Для сервера существуют команды:

1. “L” – получение логина пользователя и отправка в ответ количества пользователей онлайн;
2. “M” – получение сообщения и его дальнейшая передача другим пользователям.
3. “E” – получение сообщения о том, что пользователь покинул чат.

# **3.4 Проектирование интерфейса пользователя**

Интерфейс пользователя должен содержать поля для ввода сообщения, вывода диалога чата, кнопок, повышающих удобность использования программы, и кнопки для отправки сообщения. Схема клиентского интерфейса представлена ниже на рисунке 3.4.1.

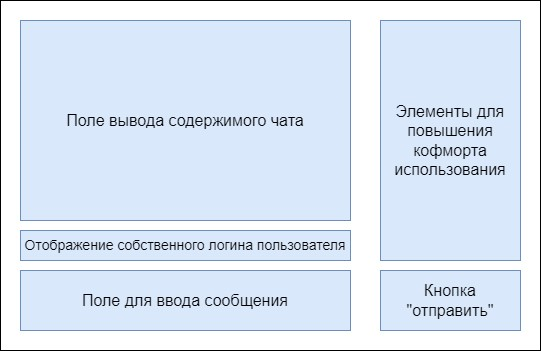


Рисунок 3.4.1 – Схема пользовательского интерфейса

# **3.5 Выбор средств разработки**

Операционная система: Windows 10;

Язык программирования: C#;

Среда разработки: Visual Studio 2022.

Для разработки клиентской части программы был выбран интерфейс Windows Forms среды Visual Studio 2022, а серверная часть имеет консольный интерфейс. Visual Studio 2022 - это интегрированная среда разработки для языка программирования C#.

# **4 Реализация ПО**

# **4.1 Реализация серверной части**

UML-диаграмма классов серверной части представлена ниже на рисунке 4.1.1.

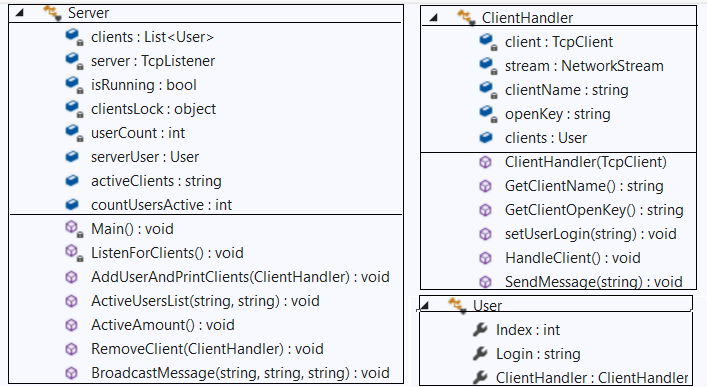


Рисунок 4.1.1 - UML-диаграмма классов серверной части программы

Класс Server отвечает за установление связи с клиентом, его добавление и удаление из списка клиентов класса Users, класс ClientHandler обрабатывает запросы клиента.

# **4.2 Реализация клиентской части**

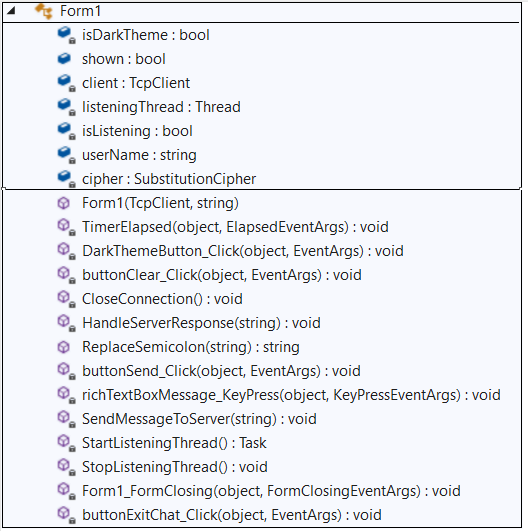
UML-диаграмма классов клиентской части представлена ниже на рисунке 4.2.1.

Рисунок 4.2.1 - UML-диаграмма классов клиентской части программы

Form1 это по сути главный класс клиентского приложения, он отвечает за интерфейс, отправку сообщений серверу, прослушку сообщений от сервера, осуществляет методы шифрования сообщений. На рис. 4.2.2 представлен его интерфейс:

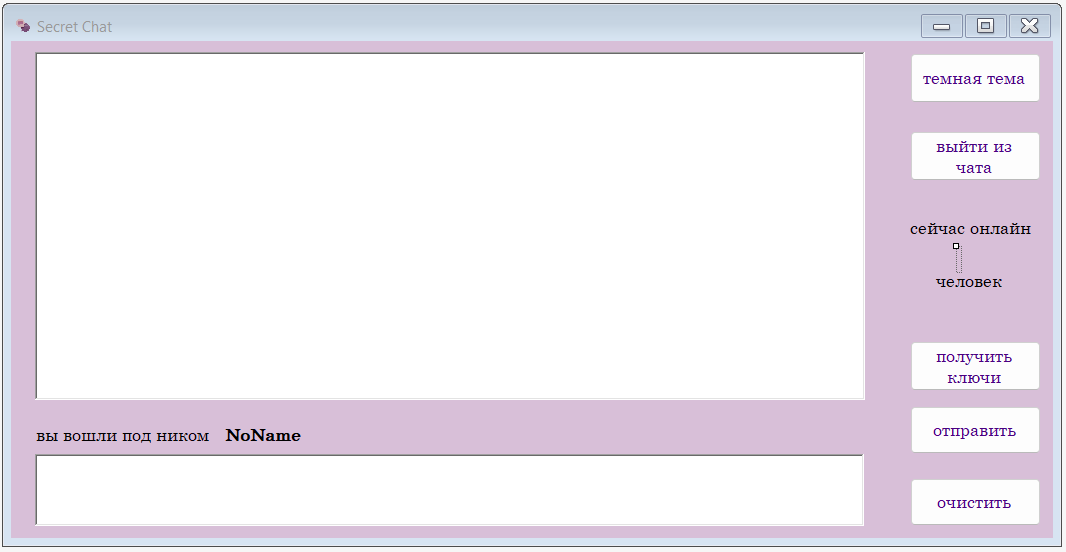


Рисунок 4.2.2 – Главное окно клиентской части

# **4.3 Описание контрольного примера**

Работа приложения была протестирована на контрольном примере. После запуска приложения появилось окно подключения к серверу(рис. 4.3.1).

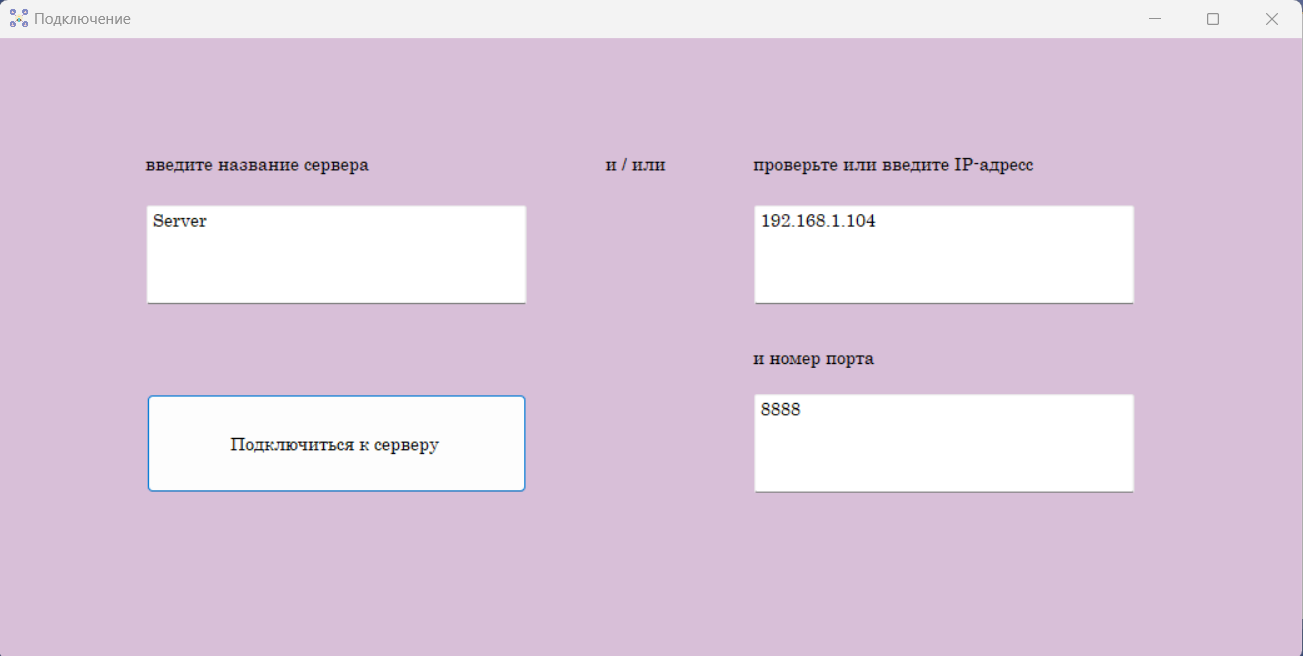


Рисунок 4.3.1 – Результат выполнения контрольного примера для клиентской части программы

После нажатия на кнопку «Подключиться к серверу» появилось окно ввода логина, изображённое на рисунке 4.3.2, а в серверной части программы появилась запись о подключении клиента. (рис. 4.3.3)

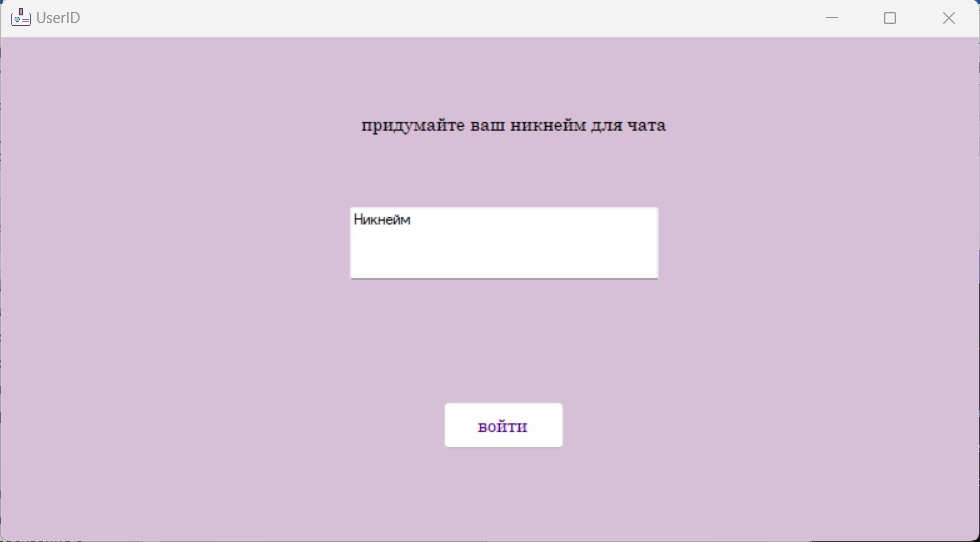


Рисунок 4.3.2 – Результат выполнения контрольного примера для клиентской части программы

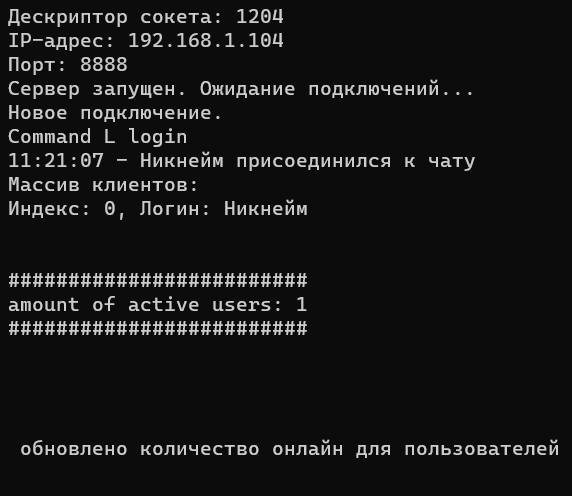


Рисунок 4.3.3 – Результат выполнения контрольного примера для серверной части программы

# **5 Руководство пользователя**

Перед запуском клиентской части программы необходимо запустить сервер для этого необходимо вызвать файл ChatServer.exe. После вызова появится окно, представленное ниже на рисунке 5.1.

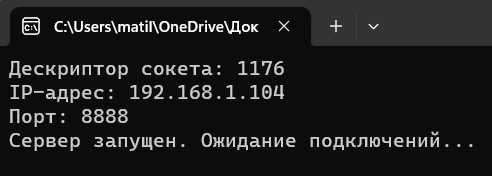


Рисунок 5.1 – Окно сервера до подключения клиентов

Для запуска клиентской части приложения необходимо вызвать файл ChatClient.exe. После вызова приложения появляется окно подключения к серверу (рис. 5.2).

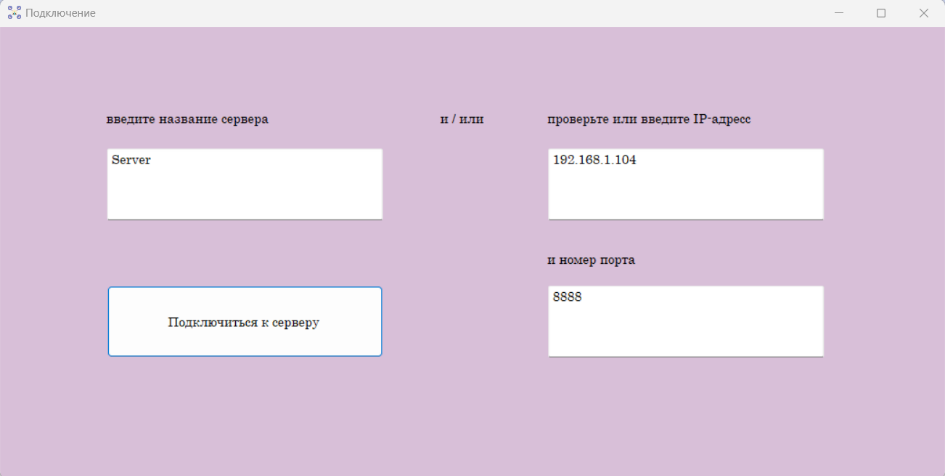


Рисунок 5.2 – Окно для подключения к серверу

После успешного подключения к серверу появляется окно ввода никнейма/логина пользователя, представленное ниже на рисунке 5.3.

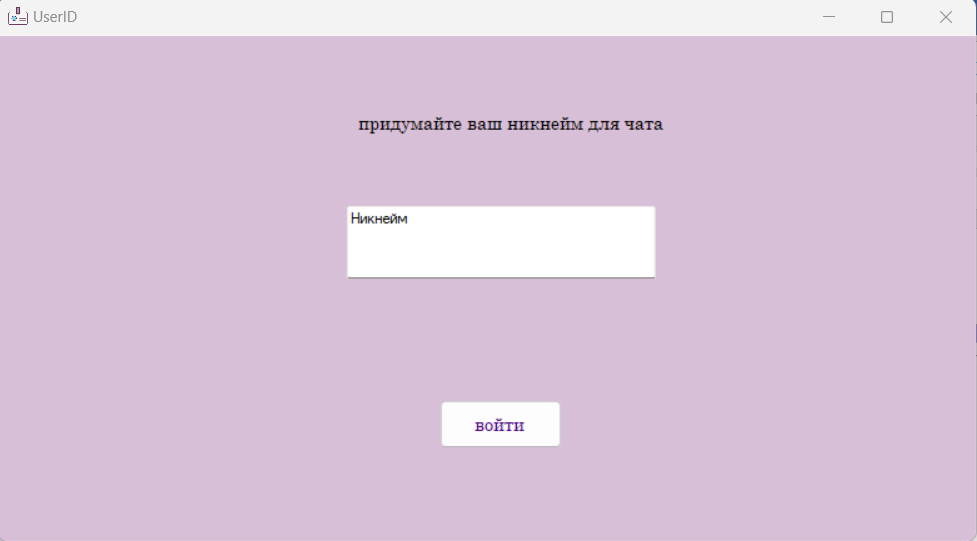


Рисунок 5.3 – Окно клиентской части для ввода никнейма

После этого следует присоединение к чату и открывается главная форма приложения (рис. 5.4):



Рисунок 5.4 – Главное окно клиентской части

Приложение предоставляет возможность менять цветовую тему интерфейса (рис. 5.5):

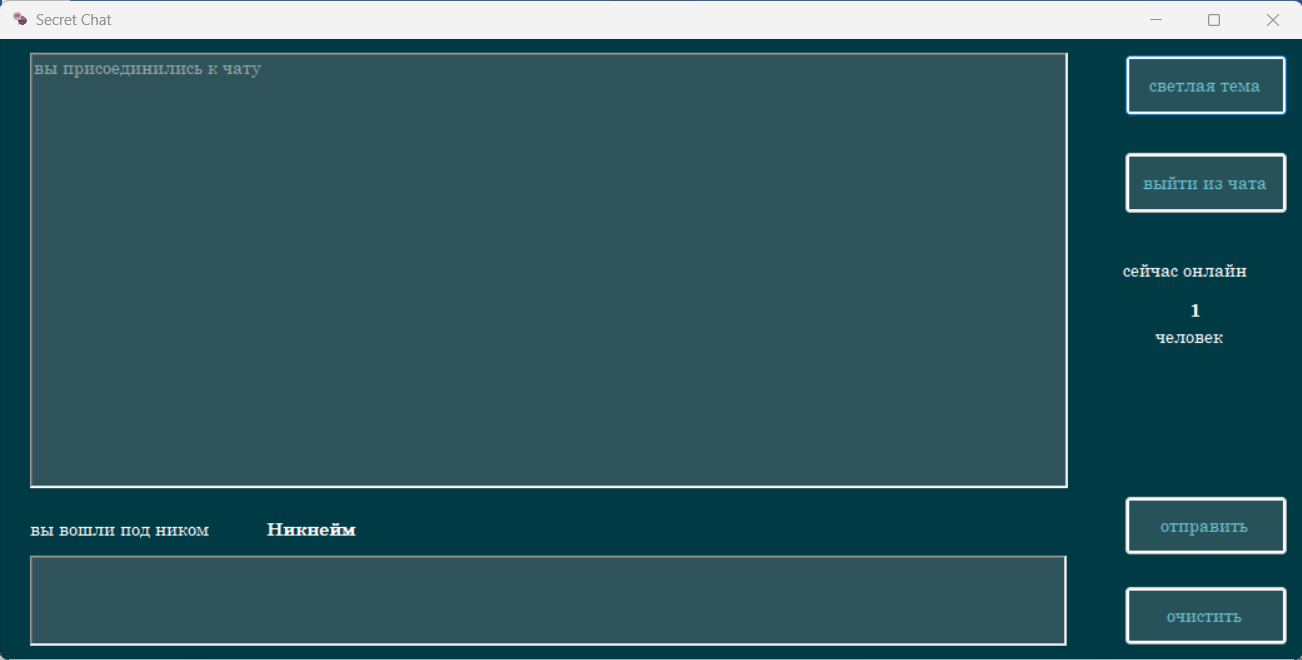


Рисунок 5.5 – Окно клиентской части с темной темой

После написания и отправки сообщения, оно появляется в окне с чатом, как на рисунке 5.6:

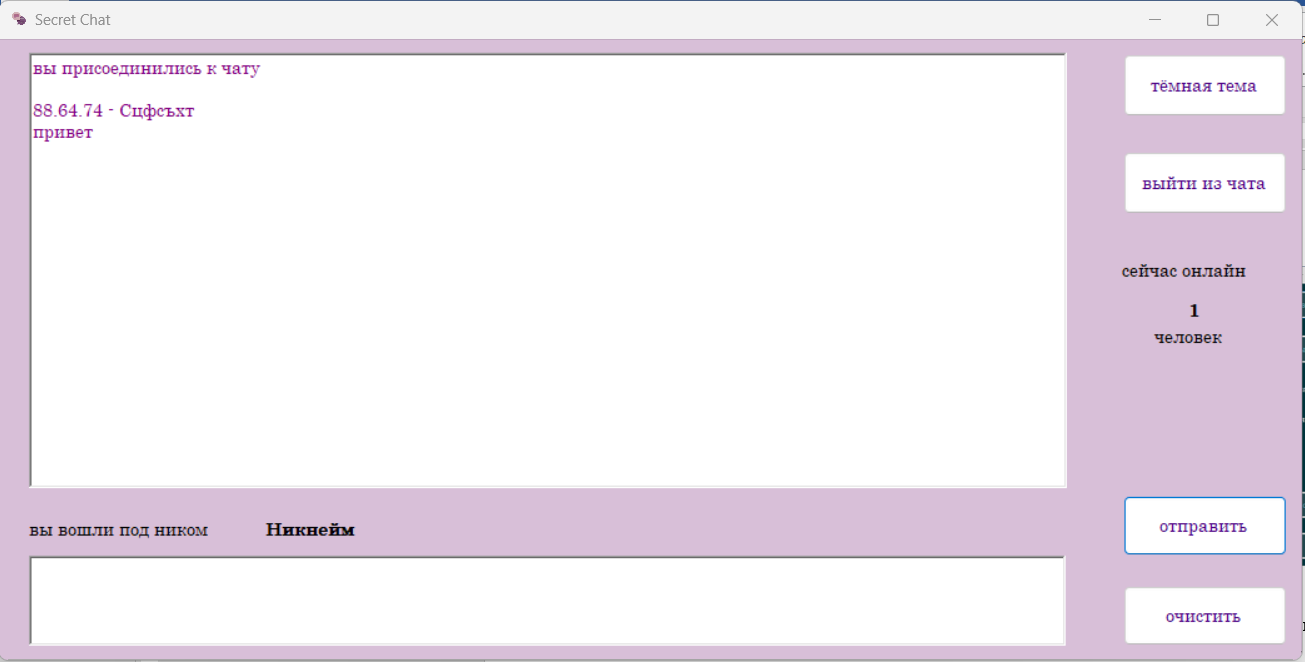


Рисунок 5.6 – Окно клиентской части с новым сообщением

Все сообщения с командами, посылаемые клиентом отображены в серверной части приложения. Интерфейс серверной части программы представлен ниже на рисунке 5.7.

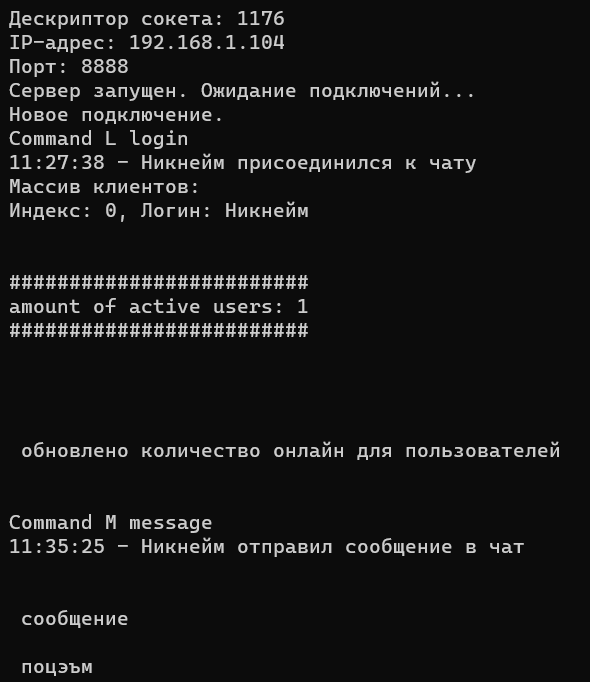


Рисунок 5.7 – Интерфейс серверной части программы

# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения данного курсового проекта были применены знания, полученные при изучении дисциплины "Компьютерные системы и сети". Были изучены основы клиент-серверного взаимодействия и использование средств C# для разработки многопоточных клиент-серверных приложений на основе сокетов. В результате был получен опыт создания полноценного сетевого приложения.

Разработанный проект представляет собой сетевое ПО, позволяющее нескольким клиентам подключаться к одному сетевому чату и обмениваться сообщениями в реальном времени. Серверная часть обладает функциональностью приема, отправки и обработки данных, а клиентская часть способна подключаться к серверу, отправлять запросы и получать ответы.

В процессе разработки и тестирования программы были исправлены выявленные ошибки и реализована обработка исключительных ситуаций, которые могли привести к неработоспособности приложения.

Приложение обладает интуитивно понятным, простым и эффективным интерфейсом. В будущем возможно его расширение, например, добавление функциональности асимметричного шифрования для повышения уровня безопасности.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Creating a Substitution Cipher with C# [Электронный ресурс]. – 2023. – Режим доступа: https://www.c-sharpcorner.com/uploadfile/b942f9/creating-a-substitution-cipher-with-C-Sharp/
2. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 2-е изд. / В.Г.Олифер, Н.А.Олифер. — СПб: «Питер», 2003. — 864с.

# **ПРИЛОЖЕНИЕ A**

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—1070112120–ДО-2023-01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Панасевич |  |  | Архитектура приложения | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **1** | **5** |
| Консульт. | | Белова |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—1070112120–ДО–2023-02 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Панасевич |  |  | Диаграмма вариантов использования | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **2** | **5** |
| Консульт. | | Белова |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—1070112120–ДО–2023-03 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Панасевич |  |  | Диаграмма классов клиентской части | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **3** | **5** |
| Консульт. | | Белова |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—1070112120–ДО–2023-04 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Панасевич |  |  | Диаграмма классов серверной части | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **4** | **5** |
| Консульт. | | Белова |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

**Графическая часть**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—1070112120–ДО–2023-05 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Панасевич |  |  | Интерфейс пользователя | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | Д |  | **5** | **5** |
| Консульт. | | Белова |  |  | 1 40 01 01 БНТУ  г.Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | |  |  |  |

# **ПРИЛОЖЕНИЕ Б**

**Листинг клиентской и серверной части**

**Enter.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Threading;

using System.Windows.Forms;

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

namespace Chat\_Client

{

public partial class Enter : Form

{

private Dictionary<string, int> serverDictionary;

IPAddress serverIP = Dns.GetHostEntry(Dns.GetHostName()).AddressList.FirstOrDefault(ip => ip.AddressFamily == AddressFamily.InterNetwork);

private TcpClient client = new TcpClient();

public Enter()

{

InitializeComponent();

// Создание словаря для хранения серверов и портов

serverDictionary = new Dictionary<string, int>();

serverDictionary.Add("Server", 8888);

textBoxServerPort.Text = "8888";

// Получение IP-адреса сервера

//textBoxServerIP.Text = "127.0.0.1";

textBoxServerIP.Text = serverIP.ToString();

}

private void textBoxChooseServer\_TextChanged(object sender, EventArgs e)

{

string serverName = textBoxChooseServer.Text;

if (serverDictionary.ContainsKey(serverName))

{

int port = serverDictionary[serverName];

textBoxServerPort.Text = port.ToString();

// Получение IP-адреса сервера

textBoxServerIP.Text = serverIP.ToString();

}

else

{

textBoxServerPort.Text = string.Empty; // Очищаем, если сервер не найден

textBoxServerIP.Text = string.Empty; // Очищаем, если сервер не найден

}

}

private void buttonConnect\_Click(object sender, EventArgs e)

{

// Получение IP-адреса сервера из textBox2

IPAddress serverIP;

if (IPAddress.TryParse(textBoxServerIP.Text, out serverIP))

{

// Получение порта сервера

int serverPort;

if (int.TryParse(textBoxServerPort.Text, out serverPort))

{

try

{

// Подключение к серверу

client.Connect(serverIP, serverPort);

MessageBox.Show("Подключено к серверу.", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

// Создание экземпляра формы UserID

UserID userIDForm = new UserID(client);

userIDForm.Show();

this.Hide();

}

catch (Exception ex)

{

MessageBox.Show("Не удалось подключиться к серверу: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("Введите корректный номер порта.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("Введите корректный IP-адрес сервера.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

**UserID.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net.Sockets;

using System.Security.Cryptography;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using MyRSALibrary\_v1;

namespace Chat\_Client

{

public partial class UserID : Form

{

// Добавьте поле для хранения имени пользователя

private string userName;

private TcpClient client;

private string combinedMessage;

private string command;

public UserID(TcpClient \_client)

{

InitializeComponent();

this.client = \_client;

}

private void buttonID\_Click(object sender, EventArgs e)

{

// Получение имени пользователя из textBoxUserID

userName = textBoxID.Text;

command = "L";

//combinedMessage = $"{command};{userName};{openKey}";

combinedMessage = $"{command};{userName}";

try

{

// Получение потока сети из объекта TcpClient

NetworkStream stream = client.GetStream();

// Преобразование сообщения в массив байтов

byte[] buffer = Encoding.UTF8.GetBytes(combinedMessage);

// Отправка сообщения на сервер

stream.Write(buffer, 0, buffer.Length);

stream.Flush();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при отправке логина: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

// Создание экземпляра формы Form1 с передачей имени пользователя

Form1 form1 = new Form1(client, userName);

form1.Show();

// Закрытие формы

this.Close();

}

private void textBoxID\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == '\r')

{

buttonID.PerformClick();

}

}

}

}

**Form1.cs**

using System;

using System.Drawing;

using System.Drawing.Text;

using System.Net.Sockets;

using System.Reflection;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Security.Cryptography.RSACryptoServiceProvider;

using MyRSALibrary\_v1;

using System.Security.Cryptography;

using System.Timers;

using MySubstitutionCipherLibrary;

namespace Chat\_Client

{

public partial class Form1 : Form

{

private bool isDarkTheme = false;

public bool shown = false;

private TcpClient client;

private Thread listeningThread;

private bool isListening = false;

private string userName;

SubstitutionCipher cipher = new SubstitutionCipher();

public Form1(TcpClient \_client, string \_userName)

{

InitializeComponent();

buttonGetKeys.Visible = false;

this.client = \_client;

userName = \_userName;

labelNickname.Text = userName;

richTextBoxChat.Text = "вы присоединились к чату\n";

// Запуск прослушивания сервера при загрузке формы

\_ = StartListeningThread();

}

private void TimerElapsed(object sender, ElapsedEventArgs e)

{

// Отправка сообщения "U;" на сервер

SendMessageToServer("U;");

}

private void DarkThemeButton\_Click(object sender, EventArgs e)

{

if (isDarkTheme)

{

// Возврат к светлой теме

this.BackColor = Color.Thistle;

this.ForeColor = Color.Black;

DarkThemeButton.Text = "тёмная тема";

// Изменение цвета кнопок обратно на светлый

foreach (Control control in this.Controls)

{

if (control is Button)

{

Button button = (Button)control;

button.BackColor = Color.White;

button.ForeColor = Color.Indigo;

}

else if (control is RichTextBox)

{

RichTextBox richTextBox = (RichTextBox)control;

richTextBox.BackColor = Color.White;

richTextBox.ForeColor = Color.Purple;

}

}

}

else

{

// Переход к темной теме

this.BackColor = Color.FromArgb(0, 58, 68);

this.ForeColor = Color.White;

DarkThemeButton.Text = "светлая тема";

// Изменение цвета кнопок на темный

foreach (Control control in this.Controls)

{

if (control is Button)

{

Button button = (Button)control;

button.BackColor = Color.FromArgb(39, 82, 90);

button.ForeColor = Color.FromArgb(108, 193, 208);

}

else if (control is RichTextBox)

{

RichTextBox richTextBox = (RichTextBox)control;

richTextBox.BackColor = Color.FromArgb(47, 83, 90);

richTextBox.ForeColor = Color.FromArgb(146, 167, 171);

}

}

}

// Инвертирование состояния темы

isDarkTheme = !isDarkTheme;

}

private void buttonClear\_Click(object sender, EventArgs e)

{

richTextBoxMessage.Text = string.Empty;

}

private void CloseConnection()

{

try

{

// Закрытие потока сети и клиента

client.GetStream().Close();

client.Close();

}

catch (Exception ex)

{

// Обработка исключений, возникающих при закрытии соединения

MessageBox.Show("Ошибка при закрытии соединения с сервером: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void HandleServerResponse(string response)

{

if (InvokeRequired)

{

Invoke((Action)(() => HandleServerResponse(response)));

return;

}

richTextBoxChat.AppendText(response + Environment.NewLine);

richTextBoxChat.ScrollToCaret();

}

public static string ReplaceSemicolon(string message)

{

if (message.Contains(";"))

{

message = message.Replace(";", ".");

}

return message;

}

private void buttonSend\_Click(object sender, EventArgs e)

{

// Получение текста из richTextBoxMessage

string message = richTextBoxMessage.Text;

message = ReplaceSemicolon(message);

message = cipher.Encrypt(message);

// Отправка сообщения на сервер

SendMessageToServer(message);

}

//поиск слова при нажатии клавиши 'Enter'

private void richTextBoxMessage\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar == '\r')

{

buttonSend.PerformClick();

}

}

private void SendMessageToServer(string \_message)

{

try

{

string command = "M";

string message = \_message;

// Combine the command and encrypted message

string combinedMessage = $"{command};{message}";

// Convert the combined message to bytes

byte[] buffer = Encoding.UTF8.GetBytes(combinedMessage);

// Get the network stream from the TcpClient

NetworkStream stream = client.GetStream();

// Send the encrypted message to the server

stream.Write(buffer, 0, buffer.Length);

stream.Flush();

// Clear the richTextBoxMessage after sending the messages

richTextBoxMessage.Text = string.Empty;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при отправке сообщения: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private async Task StartListeningThread()

{

string command, message;

try

{

// Получение потока сети из объекта TcpClient

NetworkStream stream = client.GetStream();

// Бесконечный цикл для прослушивания сервера

while (true)

{

byte[] buffer = new byte[client.ReceiveBufferSize];

int bytesRead = await stream.ReadAsync(buffer, 0, buffer.Length);

if (bytesRead == 0)

{

// Сервер отключился

break;

}

string response = Encoding.UTF8.GetString(buffer, 0, bytesRead);

// Разделение сообщения на команду и текст

string[] parts = response.Split(';');

command = parts[0];

switch (command)

{

case "U":

message = parts[1];

// Обработка ответа сервера

label4.Text = message;

break;

case "M":

//buttonGetKeys\_Click(sender, new EventArgs());

message = parts[1];

message = cipher.Decrypt(message);

// Обработка ответа сервера

HandleServerResponse(message);

break;

default:

break;

}

// Очистка буфера

Array.Clear(buffer, 0, buffer.Length);

}

// Выполняем необходимые действия после отключения от сервера

CloseConnection();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при получении сообщения от сервера: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void StopListeningThread()

{

// Устанавливаем флаг прослушивания в false

isListening = false;

// Дожидаемся завершения потока

if (listeningThread != null && listeningThread.IsAlive)

{

listeningThread.Join();

listeningThread = null;

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

try

{

// Получение потока сети из объекта TcpClient

NetworkStream stream = client.GetStream();

// Преобразование сообщения в массив байтов

byte[] buffer = Encoding.UTF8.GetBytes($"E;{userName}");

// Отправка сообщения на сервер

stream.Write(buffer, 0, buffer.Length);

stream.Flush();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при отправке логина: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

buttonExitChat\_Click(sender, e);

}

private void buttonExitChat\_Click(object sender, EventArgs e)

{

// Остановка прослушивания при закрытии формы

StopListeningThread();

// Завершение работы приложения

Application.Exit();

}

}

}  
   
**Class1.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace MySubstitutionCipherLibrary

{

public class SubstitutionCipher

{

private Dictionary<char, char> encryptionKey;

private Dictionary<char, char> decryptionKey;

public SubstitutionCipher()

{

Dictionary<char, char> encryptionKey = new Dictionary<char, char>()

{

// Пары букв и цифр, слева по возрастанию, справа по убыванию

};

this.encryptionKey = encryptionKey;

this.decryptionKey = encryptionKey.ToDictionary(pair => pair.Value, pair => pair.Key);

}

public string Encrypt(string text)

{

StringBuilder encryptedText = new StringBuilder();

foreach (char c in text)

{

if (encryptionKey.ContainsKey(c))

{

char encryptedChar = encryptionKey[c];

encryptedText.Append(encryptedChar);

}

else

{

encryptedText.Append(c);

}

}

return encryptedText.ToString();

}

public string Decrypt(string encryptedText)

{

StringBuilder decryptedText = new StringBuilder();

foreach (char c in encryptedText)

{

if (decryptionKey.ContainsKey(c))

{

char decryptedChar = decryptionKey[c];

decryptedText.Append(decryptedChar);

}

else

{

decryptedText.Append(c);

}

}

return decryptedText.ToString();

}

}

}

**Program.cs**using System;

using System.Collections.Generic;

using System.Globalization;

using System.Net;

using System.Net.Sockets;

using System.Security.Cryptography;

using System.Text;

using System.Threading;

using MyRSALibrary\_v1;

class Server

{

private static readonly List<User> clients = new List<User>();

private static TcpListener server;

private static bool isRunning = true;

private static readonly object clientsLock = new object();

private static int userCount = 0; // Счетчик пользователей

public static User serverUser; // Declaring serverUser variable

public static string activeClients="";

public static int countUsersActive;

static void Main()

{

// Задайте IP-адрес и порт для прослушивания

//IPAddress ipAddress = IPAddress.Parse("127.0.0.1");

IPAddress ipAddress = Dns.GetHostEntry(Dns.GetHostName()).AddressList.FirstOrDefault(ip => ip.AddressFamily == AddressFamily.InterNetwork);

int port = 8888;

server = new TcpListener(ipAddress, port);

// Запуск прослушивания

server.Start();

// Вывод информации о слушающем сокете

IPEndPoint iPEndPoint = (IPEndPoint)server.Server.LocalEndPoint;

Console.WriteLine("Дескриптор сокета: {0}\nIP-адрес: {1}\nПорт: {2}",

server.Server.Handle, iPEndPoint.Address, iPEndPoint.Port);

Console.WriteLine("Сервер запущен. Ожидание подключений...");

// Прослушивание подключений в отдельном потоке

Thread listenThread = new Thread(ListenForClients);

listenThread.Start();

// Обработка команд администратора в основном потоке

while (true)

{

string command = Console.ReadLine();

if (command.ToLower() == "exit")

{

isRunning = false;

break;

}

}

// Завершение работы сервера

server.Stop();

Console.WriteLine("Сервер остановлен.");

}

// Метод для прослушивания подключений клиентов

static void ListenForClients()

{

while (isRunning)

{

TcpClient tcpClient = server.AcceptTcpClient();

Console.WriteLine("Новое подключение.");

// Создание нового обработчика клиента

ClientHandler clientHandler = new ClientHandler(tcpClient);

// Запуск обработки клиента в отдельном потоке

Thread clientThread = new Thread(clientHandler.HandleClient);

clientThread.Start();

}

}

public static void AddUserAndPrintClients(ClientHandler clientHandler)

{

// Создание нового пользователя

User newUser = new User();

newUser.Index = userCount;

newUser.Login = clientHandler.GetClientName();

//newUser.openKey = clientHandler.GetClientOpenKey();

newUser.ClientHandler = clientHandler;

clients.Add(newUser);

userCount++;

// Вывод массива клиентов на консоль

if (clients.Count > 0)

{

Console.WriteLine("Массив клиентов:");

activeClients = "";

foreach (var user in clients)

{

Console.WriteLine($"Индекс: {user.Index}, Логин: {user.Login}");

ActiveUsersList(user.Index.ToString(), user.Login.ToString());

//Console.WriteLine($"Индекс: {user.Index}, Логин: {user.Login}, Публичный ключ: {user.openKey}");

//ActiveUsersList(user.Index.ToString(), user.openKey);

}

}

else

{

Console.WriteLine("Массив клиентов пуст.");

}

}

public static void ActiveUsersList(string index, string login)

{

activeClients += $";{index};{login}";

Console.WriteLine($"\n\n#########################\namount of active users: {clients.LongCount()}\n#########################\n\n");

ActiveAmount();

}

public static void ActiveAmount()

{

countUsersActive = clients.Count;

}

// Метод для удаления клиента из списка при отключении

public static void RemoveClient(ClientHandler clientHandler)

{

lock (clientsLock)

{

User userToRemove = clients.Find(user => user.Login == clientHandler.GetClientName());

if (userToRemove != null)

{

clients.Remove(userToRemove);

}

}

}

// Метод для рассылки сообщения всем клиентам

public static void BroadcastMessage(string command, string message, string senderName)

{

string formattedMessage;

if (!string.IsNullOrEmpty(senderName))

{

formattedMessage = $"{command};\n{DateTime.Now.ToString("HH/mm/ss")} - {senderName}\n{message}";

}

else formattedMessage = $"{command};{message}";

lock (clientsLock)

{

foreach (var client in clients)

{

ClientHandler clientHandler = client.ClientHandler;

clientHandler.SendMessage(formattedMessage);

if (!string.IsNullOrEmpty(senderName))

{

Console.WriteLine($"{DateTime.Now.ToString("HH:mm:ss")} - {senderName} отправил сообщение в чат");

Console.WriteLine($"\n\n сообщение\n\n {message} \n\n");

}

else Console.WriteLine($"\n\n обновлено количество онлайн для пользователей \n\n");

}

}

}

}

class ClientHandler

{

private TcpClient client;

private NetworkStream stream;

private string clientName;

private string openKey;

public User clients;

public ClientHandler(TcpClient tcpClient)

{

client = tcpClient;

}

public string GetClientName()

{

return clientName;

}

public string GetClientOpenKey()

{

return openKey;

}

public void setUserLogin(string message)

{

clientName = message;

Console.WriteLine($"{DateTime.Now.ToString("HH:mm:ss")} - {clientName} присоединился к чату");

// Отправка сообщения о подключении пользователя другим клиентам

Server.BroadcastMessage("M",$"присоединился к чату.\n", clientName);

}

// Метод для обработки клиента

public void HandleClient()

{

stream = client.GetStream();

while (true)

{

byte[] buffer = new byte[client.ReceiveBufferSize];

string command = "\0", cmnd = "";

string message = string.Empty, number = string.Empty, openKey = string.Empty;

try

{

int bytesRead = stream.Read(buffer, 0, buffer.Length);

if (bytesRead == 0)

{

// Клиент отключился

break;

}

// Преобразование принятых данных в строку

string receivedMessage = Encoding.UTF8.GetString(buffer, 0, bytesRead);

// Разделение сообщения на номер и текст

string[] parts = receivedMessage.Split(';');

command = parts[0];

switch (command)

{

case "L":

Console.WriteLine("Command " + command + " login");

message = parts[1];

//openKey = parts[2];

setUserLogin(message);

//SendMessage($"OpenKServer;{Server.serverUser.openKey}");

Server.AddUserAndPrintClients(this); // Вызов функции после обработки команды "L"

cmnd = "U";

Server.BroadcastMessage(cmnd,$"{Server.countUsersActive}","");

break;

case "M":

Console.WriteLine("Command " + command + " message");

message = parts[1];

cmnd = "M";

Server.BroadcastMessage(cmnd, message, clientName);

break;

//case "U":

// Console.WriteLine("Command " + command + " sending amount of users to client");

// //openKey = parts[2];

// SendMessage($"U;{Server.countUsersActive}");

// break;

case "E":

Console.WriteLine("Command " + command + " exit");

message = parts[1];

Console.WriteLine($"{DateTime.Now.ToString("HH:mm:ss")} - {message} покинул чат");

break;

default:

Console.WriteLine("Command " + command + " doesn't exist in this context");

break;

}

// Очистка буфера

Array.Clear(buffer, 0, buffer.Length);

catch

{

// Ошибка чтения данных, клиент отключается

break;

}

}

// Закрытие соединения

stream.Close();

client.Close();

// Удаление клиента из списка

Server.RemoveClient(this);

}

// Метод для отправки сообщения клиенту

public void SendMessage(string message)

{

//string {<message>=... <key>=key}

byte[] buffer = Encoding.UTF8.GetBytes(message);

try

{

stream.Write(buffer, 0, buffer.Length);

}

catch (IOException ex)

{

Console.WriteLine("Ошибка при записи данных: " + ex.Message);

}

}

}

class User

{

public int Index { get; set; }

public string Login { get; set; }

public ClientHandler ClientHandler { get; set; }

}