

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
КАФЕДРА «ИНФОРМАЦИОННЫЕ СИСТЕМЫ»

Направление подготовки 09.03.04
«Программная инженерия»

Утверждаю
Заведующий кафедрой ИС

_____ Р.А.Валиев

_____ г.

КУРСОВАЯ РАБОТА

по дисциплине:

«Программирование»

на тему:

«Разработка прикладной программы на основе объектно-ориентированной технологии»

Автор:
студент группы 2191121

_____ Ибрагимов Р.Х.

Оценка: _____

Руководитель:
к.т.н., доцент кафедры ИС

_____ Е.В. Зубков

Дата защиты:
_____ г.

Набережные Челны
2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
НАБЕРЕЖНОЧЕЛНИНСКИЙ ИНСТИТУТ (ФИЛИАЛ)
ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО АВТОНОМНОГО ОБРАЗОВАТЕЛЬНОГО
УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
КАФЕДРА «ИНФОРМАЦИОННЫЕ СИСТЕМЫ»
Направление подготовки 09.03.04
«Программная инженерия»

Утверждаю
Заведующий кафедрой ИС

_____ Р.А.Валиев

_____ г.

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент

Ибрагимов Руслан Халитович

1 Тема

«Разработка прикладной программы на основе объектно-ориентированной технологии»

2 Срок представления к защите

_____ г.

3 Исходные данные

-алгоритм RSA шифрования

-Visual Studio 2019

-протокол TCP

4 Перечень подлежащих разработке вопросов

-реализовать библиотеку RSA-шифрования

-создать сервера по протоколу TCP

-разработать графический дизайн для приложения

-подключить базу данных для авторизации пользователей

Задание выдано

_____ г.

Е.В. Зубков

Задание принято

_____ г.

Р.Х. Ибрагимов

Введение.

Данный курсовой проект представляет из себя сетевое приложение, который использует протокол TCP для передачи информации по сети. Клиенту представлены окна регистрации и авторизации, а также само окно мессенджера. Для регистрации и авторизации пользователей приложения используется база данных SQLite. Общение проходит между двумя пользователями, сообщения которых шифруется с помощью алгоритма RSA. RSA - криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Раздел 1. Проектирование ПО.

Приложение состоит из 4 проектов.

Первый проект это RSAEncryption, который и представляет шифрование по алгоритму RSA. В классе RSA представлены следующие методы: Encode, Decode, Create, Generate, IsSimple, PutE, PutD. Для работы RSA шифрование необходимы открытый и закрытый ключ. Для их создания были созданы переменные типа данных long, так как используются очень большие числа. С помощью метода Generate, генерируются числа и проверяются методом IsSimple, являются ли эти числа простыми, из большого списка простых чисел случайно выбираются два простых числа p и q . Далее вычисляются числа n (модуль) и функция Эйлера (m). Далее с помощью методов PutE и PutD вычисляются числа e и d . Все эти числа генерируются в методе Create, который вызывается в конструкторе при создании объекта класса RSA. Числа $\{e,n\}$ —это открытый ключ клиента, а $\{d,n\}$ -закрытый. Пользователи обмениваются ключами, шифрование сообщения, которое отправляет клиент, производится с помощью открытого ключа собеседника. Шифрование выполняется методом Encrypt, который принимает текст и открытый ключ собеседника и возвращает зашифрованное сообщение. Расшифровка выполняется с помощью метода Decrypt, который принимает зашифрованный текст и приватный ключ клиента, а возвращает расшифрованный текст. Текст для расшифровки и шифрования передается из другого класса.

Второй проект это ClientsInfo. Данный проект содержит несколько классов. Первый класс UserInfo который содержит уникальный идентификатор id , а также Login клиента. Класс SClients наследуется от класса UserInfo, и предназначен для создания списка клиентов сервера. Этот класс содержит 2 конструктора - по умолчанию и конструктор принимающий следующие параметры: id клиента, логин, а также объект класса TcpClient. Класс ClientsConnected также наследуется от класса UserInfo, и представляет собой класс подключённых клиентов. Следующие классы созданы для рассылки

сообщений клиентам. Класс `MessageContent` содержит следующие поля, `string Text` (сообщение клиента), `DateTime` (время), `string Login` (логин клиента). На основе этого класса создан метод `messageContent`, предназначенный для рассылки сообщений подключённому собеседнику. Класс `Message` содержит следующие поля: текст сообщения, а также дату и время, имеет конструктор для передачи параметров (текст и время), и представляет собой зашифрованное сообщение. Класс `MessageSending` читает подключенного клиента, а также сообщение. Рассылает сообщение.

Третий проект это `Server`, который работает по протоколу TCP. Этот проект представляет из себя консольное приложение и содержит 2 класса `Start` и сам `Server`. В классе сервера описана логика работы данного сервера, создан список клиентов, конструктор для передачи `ip` и `host` для подключения, созданы объекты классов `TcpListener`, `NetworkStream` для прослушивания входящих подключений и создания сетевого потока для чтения и записи. Данный класс также содержит несколько методов `Start`, `ConnectionInformation` и `MessageToClients`. В методе `Start` сервер в бесконечном потоке ждёт новых подключений. Так же он принимает запрос на подключения, добавляет клиентов в список и вызывает методы `ConnectionInformation` и `MessageToClients`, для подключённых клиентов. Данные при передаче на сервер проходят бинарную сериализацию. На сервере проходит десериализация и вывод на консоль. Метод `MessageToClients` рассылает сообщение всем клиентам, которые добавлены в список клиентов. Список клиентов обновляется при каждом перезапуске сервера. Метод `ConnectionInformation` выводит на консоль сервера сообщение о подключении нового клиента. В классе `Start` содержится метод `Main`. В нем создается объект класса `Server`, и через него вызывается метод `Start` для запуска сервера. `Ip` для запуска сервера вычисляется с помощью метода `SearchIP`.

Четвертый проект это `MessengerClient`. Данный проект представляет из себя WPF приложение, которое состоит из 3 окон. Стартовым окном является `AuthWindow`, это окно для авторизации пользователя, которое взаимодействует

с базой данных SQLite. В данном окне 3 кнопки, а также 2 текстовых поля. Текстовые поля предназначены для ввода логина и пароля клиента. Кнопки: «выход» (для закрытия приложения), «войти» и кнопка «Вы ещё не зарегистрированы?». При нажатии кнопки «Вы ещё не зарегистрированы?» происходит переход на окно регистрации. При нажатии на кнопку «войти» выполняется метод Check. Данный метод считывает текстовые поля логина и пароля, проверяет их длину (логин должен быть длиннее 3 символов, а пароль 5 символов). Если проверка пройдена происходит поиск клиента в базе данных. Если такой клиент не найден выводится ошибка, если найден, то пользователя перекидывает на окно мессенджера. Окно регистрации содержит 3 текстовых поля и 3 кнопки («регистрация», «выход», «у вас уже есть аккаунт?»). При регистрации нужно заполнить следующие поля: логин, пароль и повтор пароля. После ввода данных необходимо нажать на кнопку «регистрация». В ней срабатывает метод CheckandRegistr. В этом методе проводятся проверки текстовых полей, а также регистрация пользователя, путём занесения его данных в базу данных. Проверка полей- логин должен быть длиннее 3 символов, пароль длиннее или равен 5 символам, а также поля пароль и повтор пароля должны совпадать. Если проверки пройдены пользователь заносится в базу данных и его перекидывает на окно авторизации. Данное окно взаимодействует с классом ApplicationViewModel благодаря паттерну MVVM(Model-View-ViewModel). Почти все методы. Окно мессенджера содержит следующие кнопки: подключиться, отключиться, отправить, 1,2,3(это кнопки выбора дизайна), закрыть. При нажатии на кнопку присоединиться срабатывает метод Connect, с помощью которого клиент подключается к серверу. Кнопкой отключиться выполняется метод Disconnect, то есть отключение от сервера. Для пользователя предоставлено 3 вида дизайна, выбор которых проходит через кнопки 1,2 и 3. Кнопка закрыть перекидывает клиента на окно авторизации и отключает его от сервера если метод Disconnect ранее не сработал. При нажатии на кнопку отправить срабатывает метод SendMessage. Данный метод передает зашифрованное сообщение на сервер, через который

его получит собеседник. К окнам WPF подключен дизайн «MaterialDesign», который обновляет дизайн практически всех элементов. Так же для обращения к некоторым командам был пакет Prism.Core. А для работы с базой данных SQLite было установлено SQLite.Core. Для работы с базой данных были созданы 2 класса. Первый из этих классов это класс модели User, который содержит все поля из таблицы базы данных. Второй класс это ApplicationContext, который наследуется от класса DBContext. Он содержит в себе список на основе класса модели и подключается к нужной нам базе данных.

Раздел 2. Блок-схемы.

Проект-RSAEncryption.

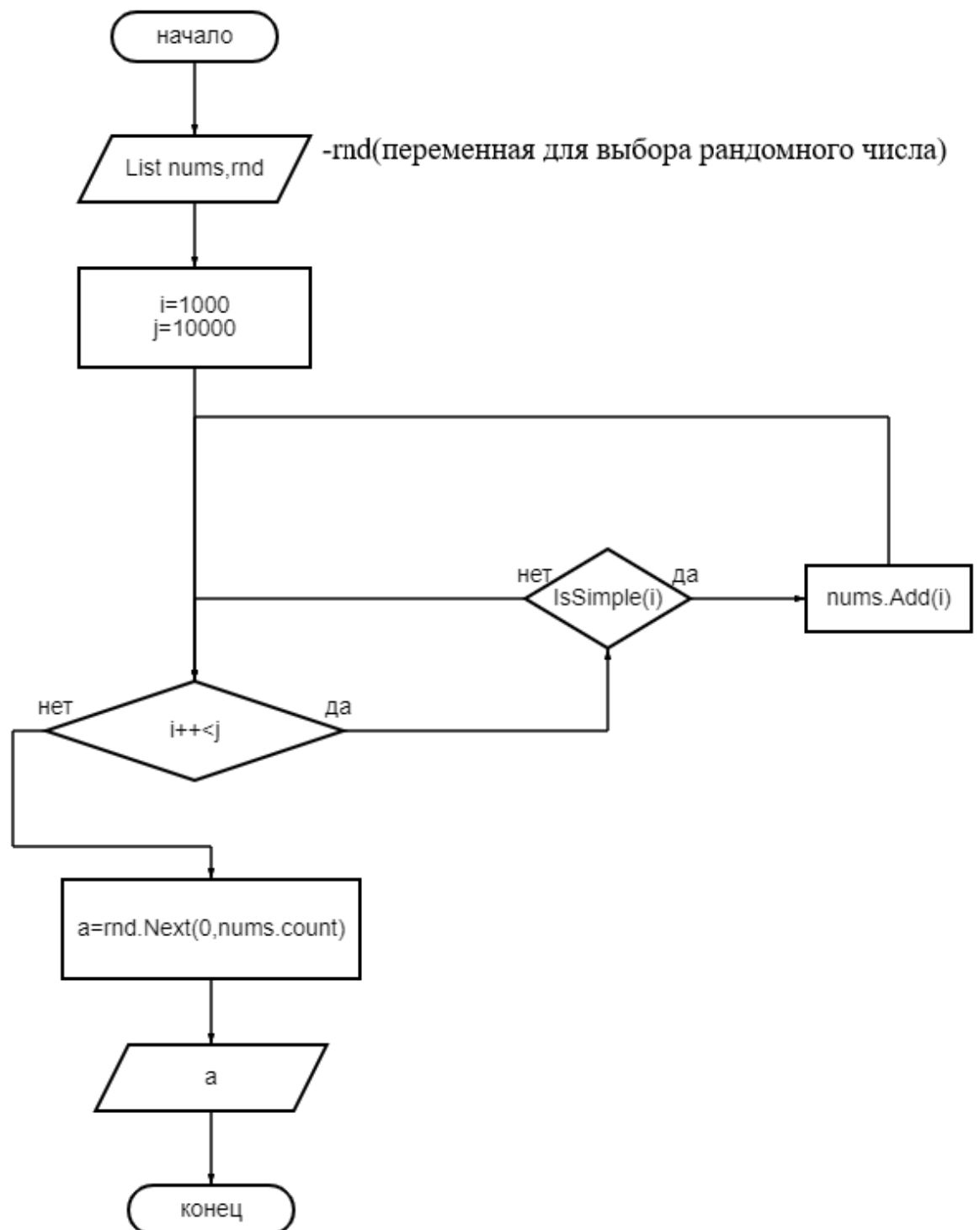


Рисунок 1. Метод Generate, для генерации простого числа

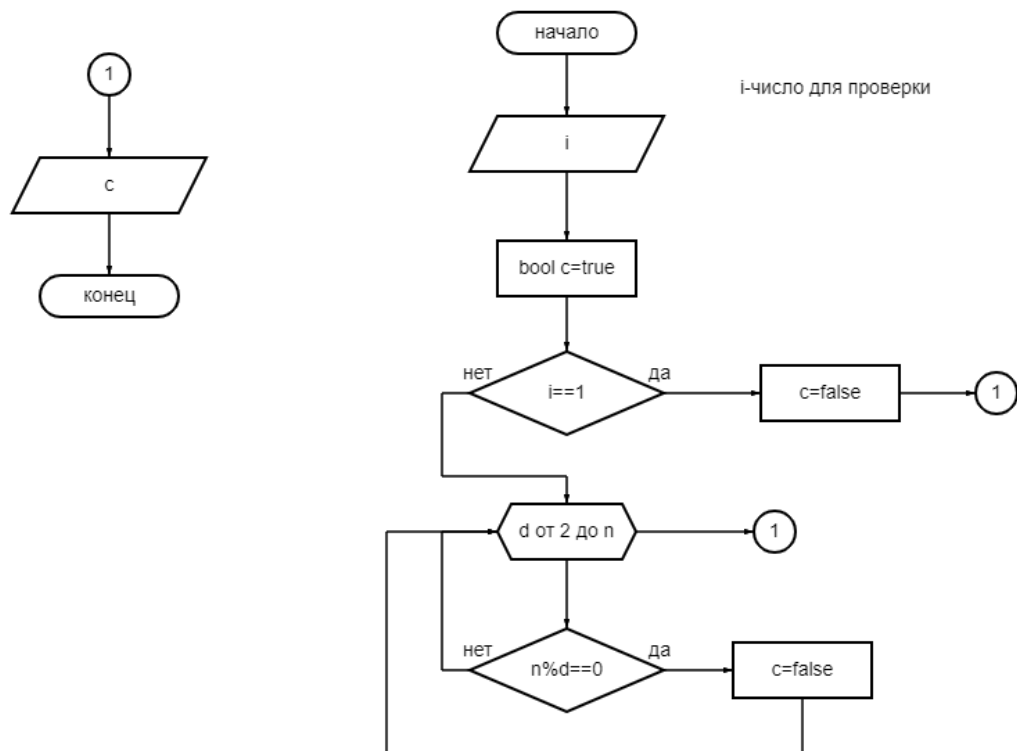


Рисунок 2. Метод IsSimple, для проверки числа, является ли оно простым

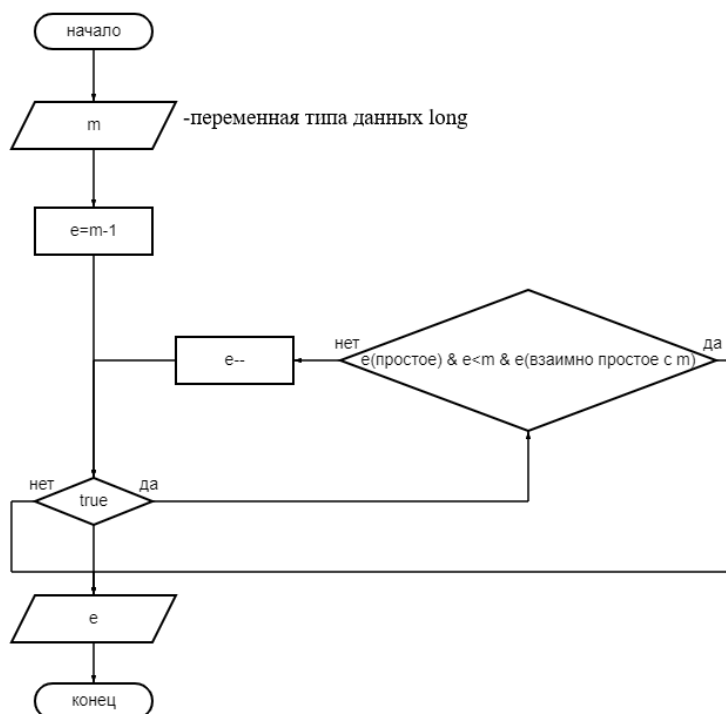


Рисунок 3. Метод PutE, для получения числа e.

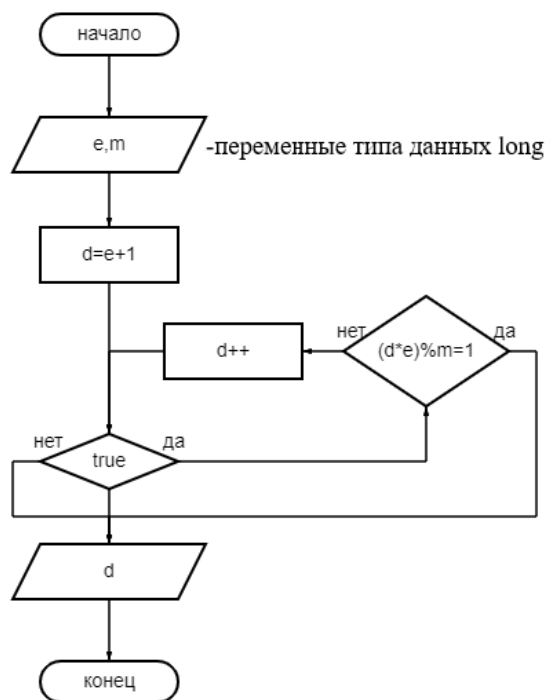


Рисунок 4. Метод PutD, для получения числа d

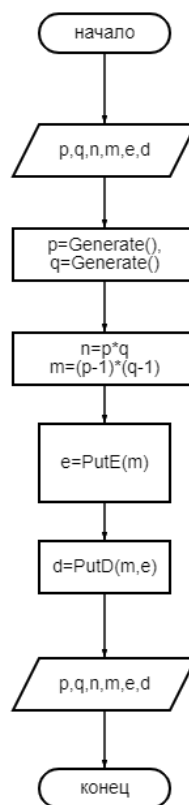


Рисунок 5. Метод Create, запускает получение чисел

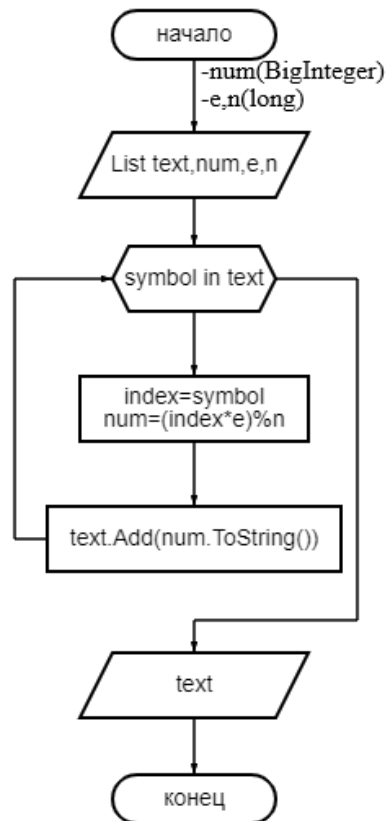


Рисунок 6. Метод Encode, для шифрования сообщения

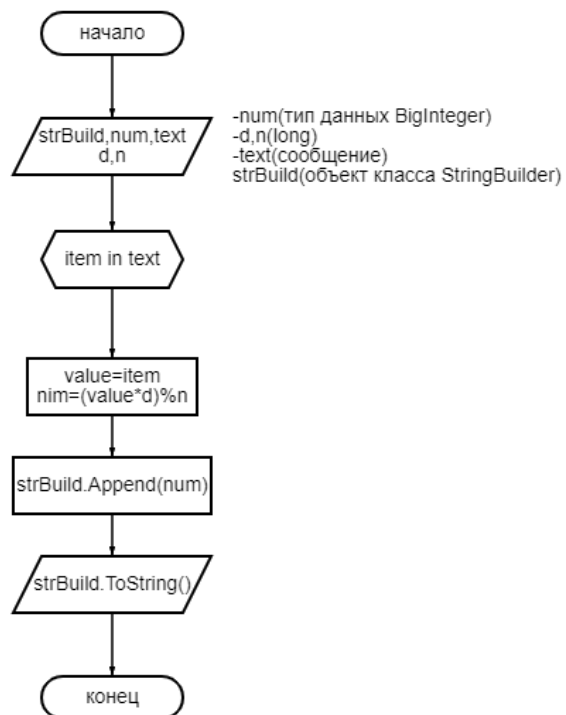


Рисунок 7. Метод Decode, для расшифровки сообщения

Проект-Server.

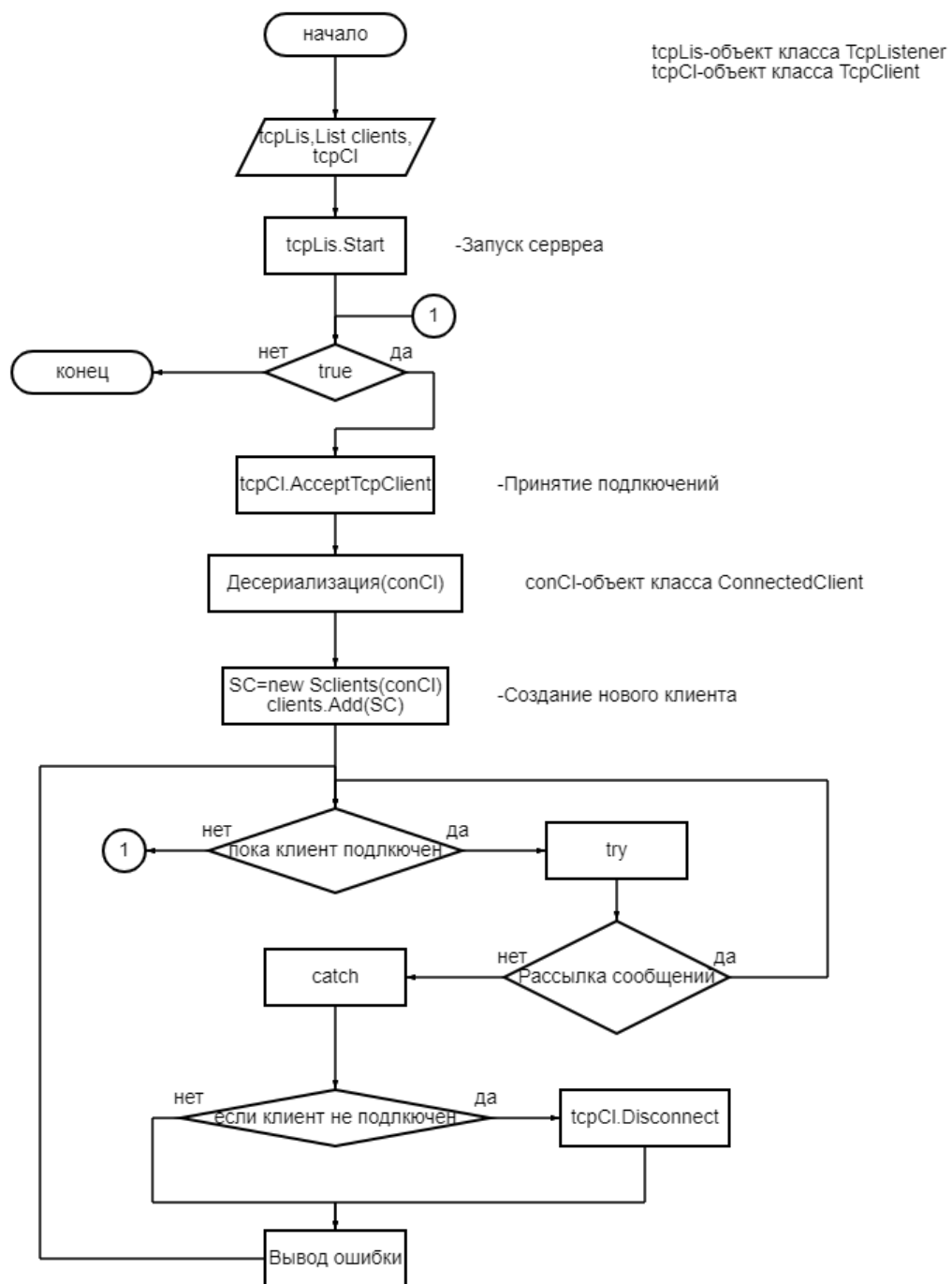


Рисунок 8. Метод Start, для запуска сервера

Проект-MessengerClient

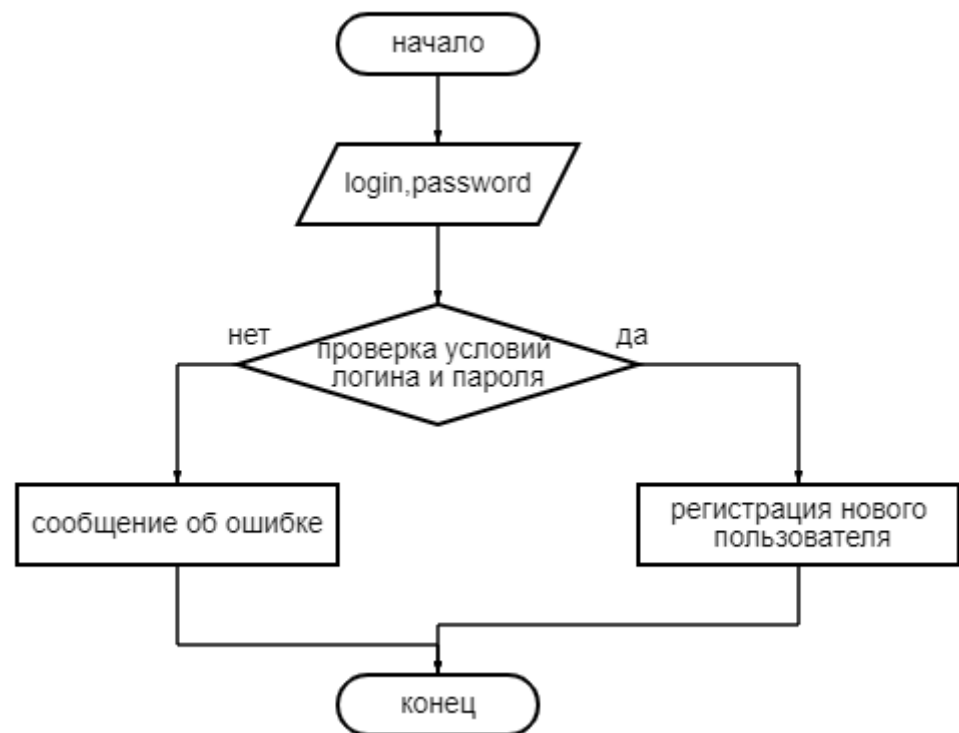


Рисунок 9. Метод CheckRegistr, для проверки условий логина и пароля и регистрации пользователя

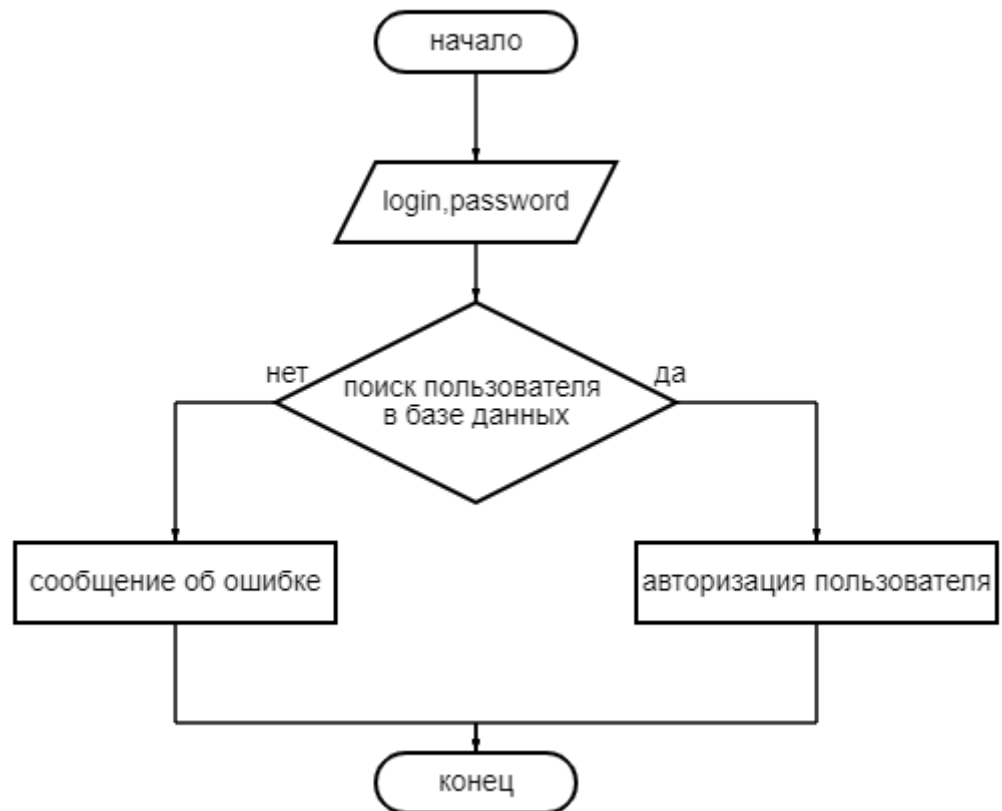


Рисунок 10. Метод Check, для поиска пользователя в базе данных и его авторизации

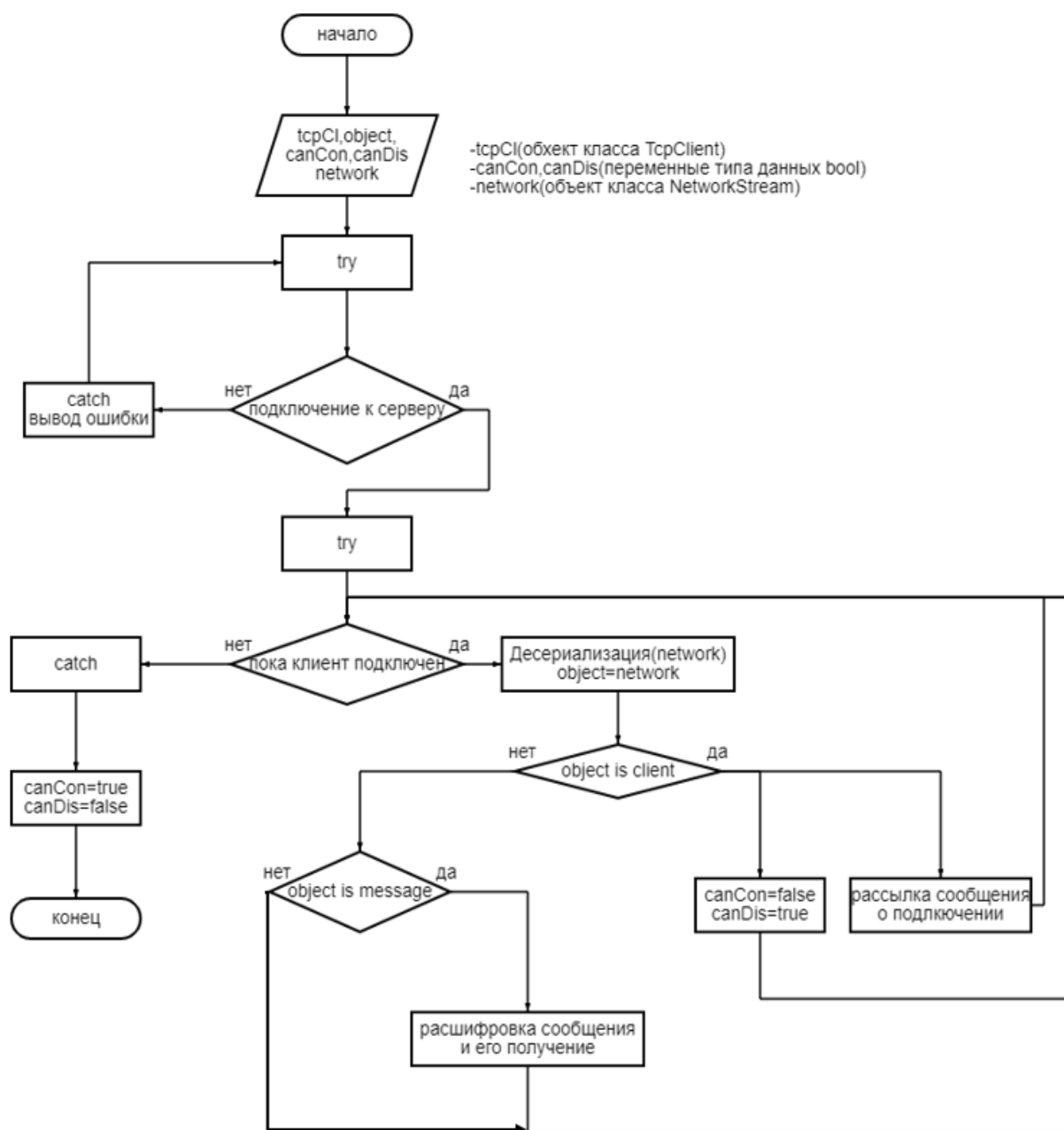


Рисунок 11. Метод Connect, для подключения к серверу

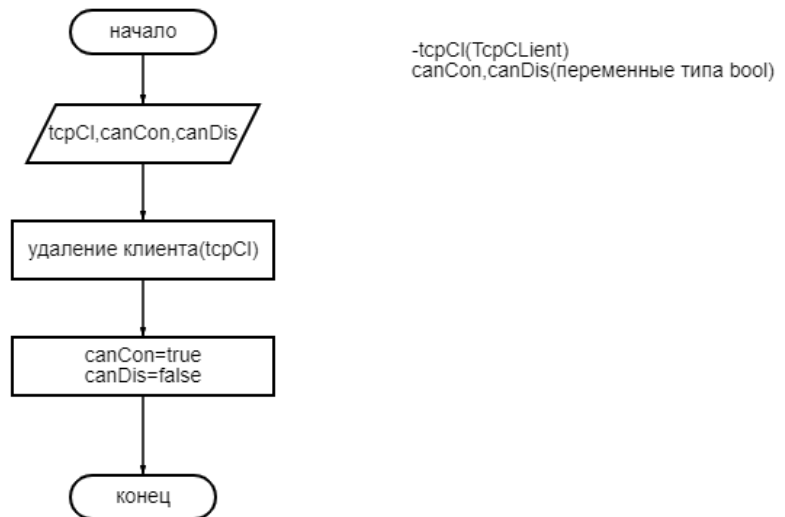


Рисунок 12. Метод Disconnect, для отключения от сервера

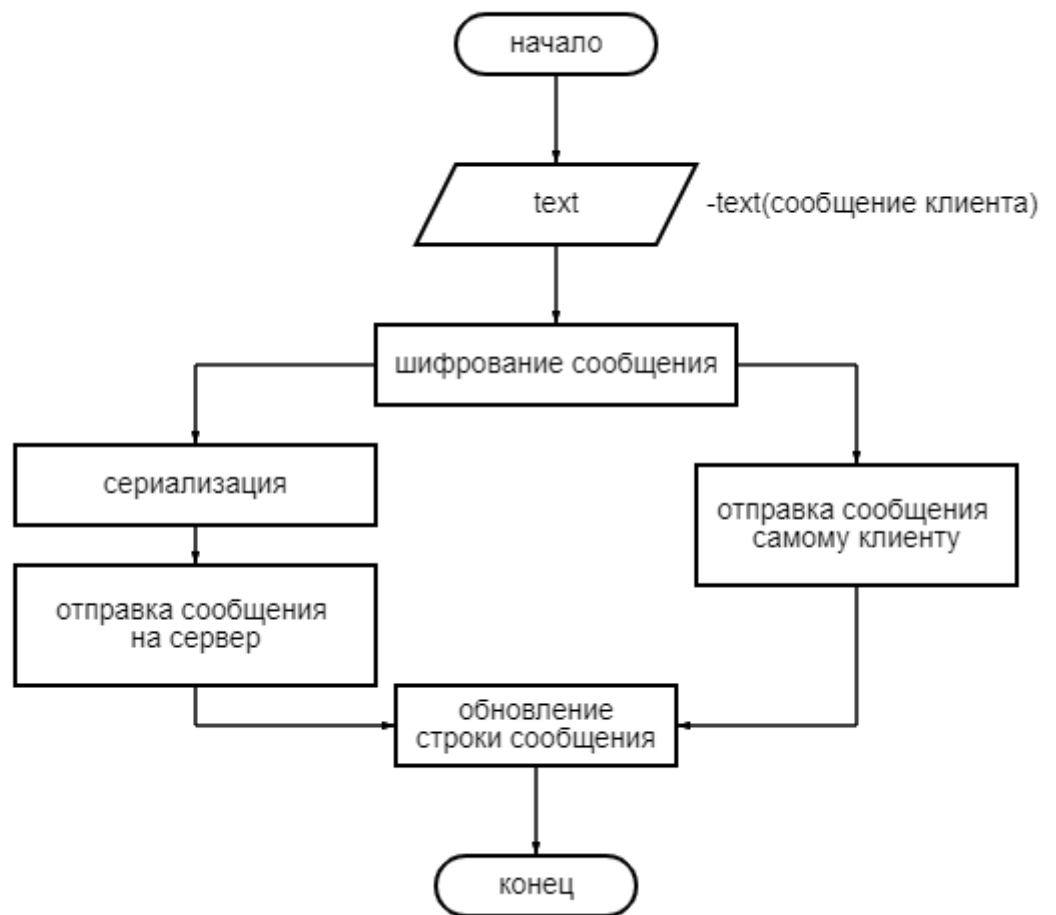


Рисунок 13. Метод SendMessage, для отправки сообщения

Раздел 3. Листинг программы.

```
Проект 1, RSAEncryption.  
Using System;  
using System.Collections.Generic;  
using System.Numerics;  
using System.Text;  
  
namespace RSAEncryption  
{  
    public class RSA  
    {  
        //Рандомная переменная для выбора рандомного числа из списка простых чисел  
        static Random random = new Random();  
        //переменные для создания ключей и шифрования  
        public long p;  
        public long q;  
        public long e;  
        public long n;  
        public long d;  
        public long m;  
        //запуск получения чисел и ключей  
        public RSA()  
        {  
            Create();  
        }  
        public void Create()  
        {  
            //p и q простые числа  
            //Генерация p  
            p = Generate();  
            //Генерация q  
            q = Generate();  
            //Получение n(Модуль произведения)  
            n = p * q;  
            //Получение m(Функция Эйлера)  
            m = (p - 1) * (q - 1);  
            //Получение числа e  
            e = PutE(m);  
            //Получение числа d  
            d = PutD(m, e);  
        }  
        //Метод шифрования для передачи клиенту
```

```

public string[] Encrypt(string text, long E, long N)
{
    return Encode(text, E, N);
}
//Метод расшифровки для передачи клиенту
public string Decrypt(string[] data)
{
    return Decode(data, d, n);
}
//Метод для шифровки
private string[] Encode(string text, long e, long n)
{
    //Создание массива с текстом
    List<string> enText = new List<string>();
    BigInteger num;
    //Шифрование с помощью открытого ключа и запись в массив
    foreach (char symbol in text)
    {
        //Умножение на x, символ каждый раз имеет новую комбинацию
чисел
        int index = symbol;
        //Шифрование по методу RSA
        num = BigInteger.ModPow(index, e, n);
        //Преобразование в строку и добавление текста в список
        enText.Add(num.ToString());
    }
    //Копируем массив
    return enText.ToArray();
}
//Метод для расшифровки
private string Decode(string[] enText, long d, long n)
{
    //Изменяемая строка символов
    StringBuilder stringBuild = new StringBuilder();
    BigInteger num;
    //Расшифровка текста
    foreach (string item in enText)
    {
        BigInteger value = new BigInteger(Convert.ToInt64(item));
        num = BigInteger.ModPow(value, d, n);
        //Добавление расшифрованного текста посимвольно
        stringBuild.Append((char)num);
    }
    //Результат расшифрованного текста в строковой форме
    return stringBuild.ToString();
}

```

```

}
//Функция для генерации простых чисел
public static long Generate()
{
    //Массив для простых чисел
    List<long> simpleNums = new List<long>();
    //числовой диапазон для чисел
    int startPosition = 1000;
    int endPosition = 10000;
    //перебор всех чисел из диапазона
    while (startPosition++ < endPosition)
    {
        //если оно простое то добавляем его в список
        if (IsSimple(startPosition))
            simpleNums.Add(startPosition);
    }
    //функция возвращает 1 простое число из списка
    return simpleNums[random.Next(0, simpleNums.Count)];
}
//Проверка простое ли число
public static bool IsSimple(long n)
{
    if (n == 1)
    {
        return false;
    }
    for (int d = 2; d * d <= n; d++)
    {
        if (n % d == 0)
            return false;
    }
    return true;
}
//Получение числа e для открытого ключа
private long PutE(long m)
{
    long e = m - 1;
    while (true)
    {
        //Проверка условий для числа e
        if (IsSimple(e) && e < m && BigInteger.GreatestCommonDivisor(new
        BigInteger(e), new BigInteger(m)) == BigInteger.One)
            break;
        e--;
    }
}

```

```

        return e;
    }
    //Получение числа d для приватного ключа
    private long PutD(long m, long e)
    {
        long d = e + 1;
        while (true)
        {
            //Проверка условий для числа d
            if ((d * e) % m == 1)
                break;
            d++;
        }
        return d;
    }
}
}

```

```

    Проект 2, ClientsInfo.
    Using System;
    using System.Net.Sockets;
[Serializable]
    public class UserInfo
    {
        //Уникальный идентификатор
        public Guid ID { get; set; }
        //Имя клиента
        public string Login { get; set; }
    }
[Serializable]
    //Подключенные клиенты
    public class ClientsConnected : UserInfo
    {
        public ClientsConnected(string login)
        {
            Login = login;
        }
    }
    public class Sclients : UserInfo
    {
        //Конструктор по умолчанию
        public Sclients()
        {

```

```

    }
    //У каждого клиента свой логин(имя), и свой уникальный
идентификатор(id)
    public Sclients(TcpClient tcpClient,Guid id,string login)
    {
        TcpClient = tcpClient;
        Login = login;
        ID = id;
    }
    //Объект для клиентского подключения к сети
    public TcpClient TcpClient;
    }
[Serializable]
    //Класс для обмена сообщениями
    public class MessageSending
    {
        //Чтение подключенных клиентов
        public ClientsConnected clientsConnected { get; }
        //Чтение сообщения
        public Messages messages { get; }
        public MessageSending(Messages messages,ClientsConnected
clientsConnected)
        {
            this.clientsConnected = clientsConnected;
            this.messages = messages;
        }

    }

[Serializable]
    //Зашифрованное сообщение
    public class Messages
    {
        //Текст сообщения
        public string [] text { get; set; }
        //Дата и время отправки сообщения
        public DateTime dateTime { get; set; }
        public Messages(string [] text, DateTime dateTime)
        {
            this.text = text;
            this.dateTime = dateTime;
        }
    }
}

```

```

//Содержание сообщения для клиентов
public class MessageContent
{
    //Текст сообщения
    public string Text { get; set; }
    //Дата и время отправки сообщения
    public DateTime Time { get; set; }
    //Логин отправителя
    public string Login { get; set; }
}

}

public static class News
{
    //Возвращает текст подключенного клиента, передает логин и время отправки
    public static MessageContent messageContent (this MessageSending messageSending,string text)
    {
        return new MessageContent() { Text = text, Login = messageSending.clientsConnected.Login, Time = messageSending.messages.dateTime };
    }
}

```

Проект 3, Server.

```

Using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Net.Sockets;
using ClientsInfo;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading;
using System.Net;

namespace Server
{
    public class Server
    {
        //Объект для прослушивание подключений от TCP-клиентов
        static TcpListener tcpListener;
        //Список клиентов сервера
        static List<Sclients> sClients;

        public Server(IPAddress host, int port)

```

```

{
    //Ожидание входящих попыток подключения, с заданным ip и port
    tcpListener = new TcpListener(host, 5050);
    //Выделение памяти для списка клиентов
    sClients = new List<Sclients>();
    //Вывод сообщения о запуске сервера на консоль
    Console.WriteLine("Сервер создан!");
    Console.WriteLine("Host:{0} Port:{1}", host, port);
}
//Метод старта сервера
public void Start()
{
    //Запуск сервера
    tcpListener.Start();
    //Вывод сообщения на консоль
    Console.WriteLine("Начало работы сервера!");
    //Ожидание подключений в бесконечном цикле
    while (true)
    {
        //Получение объекта для взаимодействия с подключенными
клиентами
        TcpClient client = tcpListener.AcceptTcpClient();
        //Запуск различных задач, выполняющихся независимо друг от друга
        Task.Factory.StartNew(() =>
        {
            //Сетевой поток для чтения и записи
            NetworkStream stream = client.GetStream();
            //Объект для сериализации и десериализации объектов
            BinaryFormatter binaryFormatter = new BinaryFormatter();
            //Объект класса с содержанием данных клиента
            Sclients SC = new Sclients();
            //Десериализация потока данных подключенных клиентов
            ClientsConnected conClient = binaryFormatter.Deserialize(stream) as
ClientsConnected;
            //Передача данных подключенного клиента
            SC = new Sclients(client, conClient.ID, conClient.Login);
            //Добавление клиента в список подключенных клиентов
            sClients.Add(SC);
            //Вывод сообщения о подключении на консоль
            ConnectionInformation(conClient);
            //Рассылка сообщения клиентам
            MessageToClients(conClient);
            //Цикл для подключенных клиентов
            while (client.Client.Connected)
            {

```

```

try
{
    //Десериализация потока данных
    MessageSending messageSending =
binaryFormatter.Deserialize(stream) as MessageSending;
    //Вывод сообщения клиента на консоль(зашифрованное)
    Console.WriteLine($"[{messageSending.clientsConnected.Login}]
“ + $”{string.Join(“”, messageSending.messages.text)}”);
    //Рассылка сообщения клиентам
    MessageToClients(messageSending);
}
//Вывод ошибок
catch (Exception e)
{
    //Определение состояния клиента
    if (client.Client.Poll(0, SelectMode.SelectRead))
    {
        //Проверка подключен ли клиент
        if (client.Client.Connected == false)
        {
            //Отключение клиента(true-сокеты могут быть повторно
использованы для подключения)
            client.Client.Disconnect(true);
        }
    }
    //Изменение цвета текста в консоли
    Console.ForegroundColor = ConsoleColor.Red;
    //Вывод ошибки на консоль
    Console.WriteLine(e.Message);
    //Изменение цвета текста в консоли
    Console.ForegroundColor = ConsoleColor.White;
    //Завершение потока
    Thread.CurrentThread.Abort();
}
}
});
}

}

//Метод рассылки сообщения клиентам
private static void MessageToClients(object msg)
{
    //Объект для сериализации и десериализации объектов
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    //Перебор всех клиентов

```



```

        foreach(Sclients client in sClients.ToArray())
        {
            if(client.TcpClient.Connected)
                //Сериализация
                binaryFormatter.Serialize(client.TcpClient.GetStream(), msg);
        }
    }

    //Отправка сообщения о подключении клиента
    private static void ConnectionInformation (ClientsConnected clientsConnected)
    {
        //Изменение цвета текста в консоли
        Console.ForegroundColor = ConsoleColor.Green;
        //Вывод информации о подключении на консоль
        Console.WriteLine($»Новое подключение: {clientsConnected.Login}»);
        //Изменение цвета текста в консоли
        Console.ForegroundColor = ConsoleColor.White;
    }
}

public class Start
{
    static void Main(string[] args)
    {
        //Запуск сервера
        Server server = new Server(SearchIP(),5050);
        server.Start();
    }
    //определение ip
    public static IPAddress SearchIP()
    {
        IPEndPoint host = Dns.GetHostEntry(Dns.GetHostName());
        foreach (var ip in host.AddressList)
            if (ip.AddressFamily == AddressFamily.InterNetwork)
                return ip;

        throw new Exception(“Ошибка”);
    }
}

```

Проект 4, MessengerClient

ApplicationViewModel.cs

```

using System;
using System.Collections.ObjectModel;
using System.Threading.Tasks;
using Prism.Commands;
using System.ComponentModel;
using RSAEncryption;
using ClientsInfo;
using System.Net;
using System.Net.Sockets;
using System.Windows.Input;
using System.Windows;
using System.Runtime.Serialization.Formatters.Binary;
using System.Windows.Threading;

namespace MessengerClient
{
    //Паттерн MVVM
    public class ApplicationViewModel : INotifyPropertyChanged
    {
        //Конструктор для установления IP и порта для прослушивания
        public ApplicationViewModel(IPAddress host, int port)
        {
            //Возвращает данный объект в виде строки
            Host = host.ToString();
            Port = port;
            //Выделение памяти под сообщения
            Messages = new ObservableCollection<MessageContent>();
            //Связь команд и методов
            SendMessageCommand = new DelegateCommand(SendMessage);
            ConnectCommand = new DelegateCommand(Connect);
            DisconnectCommand = new DelegateCommand(Disconnect);
        }
        #region objects
        RSA rSA = new RSA();
        TcpClient tcp;
        NetworkStream network;
        ClientsConnected clients;
        //Коллекция, которая оповещает внешние объекты об изменении
        public ObservableCollection<MessageContent> Messages { get; set; }
        public string Host { get; set; }
        //Имя клиента при подключении
        public string Login { get; set; } = AuthWindow.username;
        public int Port { get; set; }
    }
}

```

```

#endregion

#region Keys
//Ключи клиента
public long ClientE => rSA.e;
public long ClientN => rSA.n;
//Ключи собеседника в виде свойств
public long FriendE { get => friendE; set { friendE = value;
OnPropertyChanged(); } }
public long FriendN { get => friendN; set { friendN = value;
OnPropertyChanged(); } }
//Ключи собеседника
long friendE;
long friendN;
#endregion

#region Text
//Текст который вводит клиент
public string Text
{
    get => _text; set
    {
        _text = value;
        OnPropertyChanged();
    }
}
private string _text;
#endregion

#region bool
//Условия для активации кнопки подключения
public bool Connected => FriendE != 0 && FriendN != 0 && CanConnect;
//Переменная для определения возможности подключения
bool CanConnect = true;
//Переменная для определения возможности отключения
public bool CanDisconnect { get; set; } = false;
#endregion

#region Command
//Команда для подключения к серверу
public ICommand ConnectCommand { get;set; }
//Метод подключения
private void Connect()
{
    //Создание нового клиента

```

```

tcp = new TcpClient();

try
{
    //Попытка подключения к серверу
    tcp.Connect(Host, Port);
}
//Ошибки
catch (Exception e)
{
    //Вывод ошибки
    MessageBox.Show(e.Message, "Сервер отключен!",
    MessageBoxButton.OK, MessageBoxImage.Error);
    return;
}
//Сетевой поток для чтения и записи
network = tcp.GetStream();
//Объявление клиента подключенным
clients = new ClientsConnected(Login);
//Запуск вложенных задач
Task.Factory.StartNew(() =>
{
    try
    {
        //Объект для сериализации и десериализации объектов
        BinaryFormatter bf = new BinaryFormatter();
        //Сериализация потока чтения/записи сервера и клиентов
        bf.Serialize(network, clients);
        //Пока клиент подключен
        while (tcp.Connected)
        {
            //Десериализация
            object obj = bf.Deserialize(network);
            //Проверка является ли объект подключенным клиентом
            if (obj is ClientsConnected conClient)
            {
                //Изменение возможностей подключения и отключения
                CanConnect = false;
                CanDisconnect = true;
                //Рассылает уведомление синхронно в потоке
                Application.Current.Dispatcher.Invoke(() =>
                {
                    //Сообщение о подключении
                    Messages.Add(new MessageContent() { Text = $"Новое
подключение: {conClient.Login}", Time = DateTime.Now });

```

```

    });
    //Обновление свойств
    OnPropertyChanged();
}
//Проверяет объект является ли он сообщением
else if (obj is MessageSending msg)
{
    //Пропускает расшифровку сообщения для самого клиента
    if (msg.clientsConnected.Login == clients.Login)
        continue;
    //Расшифровка сообщения в потоке
    Application.Current.Dispatcher.Invoke(() =>
    {
        //Расшифровка
        string text = rSA.Decrypt(msg.messages.text);
        //Рассылка сообщения
        Messages.Add(msg.messageContent(text));
    }, DispatcherPriority.Background);
}
}
}
catch (Exception e)
{
    //Сообщение об отключении
    Application.Current.Dispatcher.Invoke(() =>
    {
        Messages.Add(new MessageContent() { Text = "Отключение от
сервера", Time = DateTime.Now });
    });
    //Сообщение об ошибке
    Console.WriteLine(e.Message);
    //Изменение возможностей подключения и отключения
    CanConnect = true;
    CanDisconnect = false;
    //Обновление свойств
    OnPropertyChanged();
}
}, TaskCreationOptions.LongRunning);
}
//Команда для отключения от сервера
public ICommand DisconnectCommand { get; set; }
//Метод отключения
private void Disconnect()
{
    //Удаление клиента

```

```

tcp?.Close();
//Изменение возможностей подключения и отключения
CanConnect = true;
CanDisconnect = false;
//Обновление свойств
OnPropertyChanged();

}
//Команда отправки сообщения
public ICommand SendMessageCommand { get; set; }
//Метод отправки сообщения
private void SendMessage()
{
    try
    {
        //Шифрование сообщения с помощью ключа собеседника
        string [] text = rSA.Encrypt(Text, FriendE, FriendN);
        //Отправка сообщения самому себе
        Dispatcher.CurrentDispatcher.Invoke(() =>
        {
            Messages.Add(new MessageContent() { Login = clients.Login, Text =
Text, Time = DateTime.Now });
        });
        //Отправка зашифрованного сообщения на консоль
        MessageSending message = new MessageSending(new Messages(text,
DateTime.Now), clients);
        new BinaryFormatter().Serialize(network, message);
        //Обновляет строку сообщения
        Text = string.Empty;
        OnPropertyChanged(nameof(Text));
    }
    catch (Exception e)
    {
        MessageBox.Show(e.Message, "Ошибка отправки сообщения",
MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }
}
}
#endregion

//Метод для обновления свойств(паттерн MVVM)
public void OnPropertyChanged(string name = "")
{
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));
}

```

```

        public event PropertyChangedEventHandler PropertyChanged;

    }
}

```

MessengerWindow.xaml.cs

```

using System.Windows;
using System.Windows.Input;
using System.Collections.Specialized;
using Server;
using System.Windows.Media;

```

namespace MessengerClient

```

{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            //Переход к последнему сообщению

```

```

Loaded+=(s,e)=>((INotifyCollectionChanged)Items.ItemsSource).CollectionChange
d+=(q,z)=>Scroll.ScrollToEnd();

```

```

        DataContext = new ApplicationViewModel(Start.SearchIP(),5050);

```

```

    }

```

//Заккрытие окна приложения

```

    public void CloseWindow(object sender, RoutedEventArgs e)
    {

```

```

        AuthWindow authWindow = new AuthWindow();
        authWindow.Show();
        this.Close();

```

```

    }

```

//Захват мышкой и передвижение по экрану

```

    public void DrivePanel(object sender, MouseButtonEventArgs e)
    {
        this.DragMove();
    }

```

//Дизайн по умолчанию

```

    private void FirstD_Click(object sender, RoutedEventArgs e)
    {

```

```

        //Переменная для конвертации цвета

```

```

var bc = new BrushConverter();
//Изменение цвета панелей
HorizontalPanel.Background = (Brush)bc.ConvertFrom("#FF563096");
VerticalPanel.Background = (Brush)bc.ConvertFrom("#FF563096");
//Изменение цвета кнопок
Dis.Background = (Brush)bc.ConvertFrom("#FF5A3096");
Con.Background = (Brush)bc.ConvertFrom("#FF5A3096");
First.Background = (Brush)bc.ConvertFrom("#FF673AB7");
Second.Background = (Brush)bc.ConvertFrom("#FF673AB7");
Theard.Background = (Brush)bc.ConvertFrom("#FF673AB7");
First.BorderBrush = (Brush)bc.ConvertFrom("#FF673AB7");
Second.BorderBrush = (Brush)bc.ConvertFrom("#FF673AB7");
Theard.BorderBrush = (Brush)bc.ConvertFrom("#FF673AB7");
//Изменение цвета заднего фона
Main.Background = (Brush)bc.ConvertFrom("#FF2F2B42");
//Изменение цвета кнопки отправки
Send.Background = (Brush)bc.ConvertFrom("#FF5A3096");
Send.BorderBrush = (Brush)bc.ConvertFrom("#FF673AB7");
//Изменение цвета текстового поля
MessageBox.Background = (Brush)bc.ConvertFrom("#FF393251");
//Изменение цвета кнопки закрыть
Close.Background = (Brush)bc.ConvertFrom("#FF2F2B42");
Close.BorderBrush = (Brush)bc.ConvertFrom("#FF673AB7");
}
//Второй вариант дизайна
private void SecondD_Click(object sender, RoutedEventArgs e)
{
    //Переменная для конвертации цвета
    var bc = new BrushConverter();
    //Изменение цвета панелей
    HorizontalPanel.Background=(Brush)bc.ConvertFrom("#000033");
    VerticalPanel.Background = (Brush)bc.ConvertFrom("#000033");
    //Изменение цвета кнопок
    Dis.Background = (Brush)bc.ConvertFrom("#000033");
    Con.Background = (Brush)bc.ConvertFrom("#000033");
    First.Background = (Brush)bc.ConvertFrom("#000033");
    Second.Background = (Brush)bc.ConvertFrom("#000033");
    Theard.Background = (Brush)bc.ConvertFrom("#000033");
    First.BorderBrush = (Brush)bc.ConvertFrom("#3399ff");
    Second.BorderBrush = (Brush)bc.ConvertFrom("#3399ff");
    Theard.BorderBrush = (Brush)bc.ConvertFrom("#3399ff");
    //Изменение цвета заднего фона
    Main.Background = (Brush)bc.ConvertFrom("#000000");
    //Изменение цвета кнопки отправки
    Send.Background = (Brush)bc.ConvertFrom("#000033");

```



```

Send.BorderBrush = (Brush)bc.ConvertFrom("#3399ff");
//Изменение цвета текстового поля
MessageBox.Background = (Brush)bc.ConvertFrom("#333333");
//Изменение цвета кнопки закрыть
Close.Background = (Brush)bc.ConvertFrom("#000033");
Close.BorderBrush = (Brush)bc.ConvertFrom("#3399ff");

}
//Третий вариант дизайна
private void TheardD_Click(object sender, RoutedEventArgs e)
{
    //Переменная для конвертации цвета
    var bc = new BrushConverter();
    //Изменение цвета панелей
    HorizontalPanel.Background = (Brush)bc.ConvertFrom("#660033");
    VerticalPanel.Background = (Brush)bc.ConvertFrom("#660033");
    //Изменение цвета кнопок
    Dis.Background = (Brush)bc.ConvertFrom("#660033");
    Con.Background = (Brush)bc.ConvertFrom("#660033");
    First.Background = (Brush)bc.ConvertFrom("#660033");
    Second.Background = (Brush)bc.ConvertFrom("#660033");
    Theard.Background = (Brush)bc.ConvertFrom("#660033");
    First.BorderBrush = (Brush)bc.ConvertFrom("#FFFFFFFF");
    Second.BorderBrush = (Brush)bc.ConvertFrom("#FFFFFFFF");
    Theard.BorderBrush = (Brush)bc.ConvertFrom("#FFFFFFFF");
    //Изменение цвета заднего фона
    Main.Background = (Brush)bc.ConvertFrom("#FF150115");
    //Изменение цвета кнопки отправки
    Send.Background = (Brush)bc.ConvertFrom("#660033");
    Send.BorderBrush = (Brush)bc.ConvertFrom("#FFFFFFFF");
    //Изменение цвета текстового поля
    MessageBox.Background = (Brush)bc.ConvertFrom("#333333");
    //Изменение цвета кнопки закрыть
    Close.Background = (Brush)bc.ConvertFrom("#660033");
    Close.BorderBrush = (Brush)bc.ConvertFrom("#FFFFFFFF");
}
}
}

```

AuthWindow.xaml.cs

```

using MessengerClient.DBMS;
using System.Linq;
using System.Windows;
using System.Windows.Input;

```

```

namespace MessengerClient
{
    /// <summary>
    /// Логика взаимодействия для AuthWindow.xaml
    /// </summary>
    public partial class AuthWindow : Window
    {
        //Поля для логина клиента
        public static string username = null;
        public AuthWindow()
        {
            InitializeComponent();
        }
        //Захват мышкой и перемещение по экрану
        private void Grid_MouseDown(object sender, MouseButtonEventArgs e)
        {
            this.DragMove();
        }
        //Закрытие приложения
        private void Exit_Click(object sender, RoutedEventArgs e)
        {
            this.Close();
        }
        //Переход к другому окну
        private void Auth_Click(object sender, RoutedEventArgs e)
        {
            //Получение логина
            string login = Login.Text.Trim();
            //Получение пароля
            string pass = Pass.Password.Trim();
            //Проверка пользователя
            Check(login, pass);
        }
        //Авторизация пользователя
        public void Check(string login,string pass)
        {
            //Проверка условий логина
            if (login.Length <= 3)
            {
                Login.ToolTip = "Имя должно быть длинее(минимум 3 буквы)";
            }
            //Проверка условий пароля
            else if (pass.Length < 5)

```

```

    {
        Pass.ToolTip = "Пароль должен быть длиннее";
    }
    //Если поля корректны
    else
    {
        Login.ToolTip = "";
        Pass.ToolTip = "";
        //Первоначальное значение пользователя
        User authUser = null;
        using (ApplicationContext applicationContext = new ApplicationContext())
        {
            //Поиск пользователя в БД
            authUser = applicationContext.Users.Where(cl => cl.Login == login &&
cl.Password == pass).FirstOrDefault();
        }
        //Если пользователь найден
        if (authUser != null)
        {
            //Передача имени авторизованного пользователя
            username = authUser.Login;
            //переход на другое окно
            new ApplicationViewModel(Server.Start.SearchIP(), 5050);
            MainWindow mainWindow = new MainWindow();
            mainWindow.Show();
            this.Close();
        }
        //Сообщение об ошибке
        else
            MessageBox.Show("Вы ввели что-то некорректно");
    }
}
//Переход на другое окно
private void Reg_Swipe_Click(object sender, RoutedEventArgs e)
{
    RegistrationWindow registration = new RegistrationWindow();
    registration.Show();
    this.Close();
}
}
}

```

RegistrationWindow.xaml.cs
using MessengerClient.DBMS;

```
using System.Windows;
using System.Windows.Input;
```

```
namespace MessengerClient
{
```

```
    /// <summary>
```

```
    /// Логика взаимодействия для RegistrationWindow.xaml
```

```
    /// </summary>
```

```
    public partial class RegistrationWindow : Window
```

```
    {
```

```
        ApplicationContext applicationContext;
```

```
        public RegistrationWindow()
```

```
        {
```

```
            InitializeComponent();
```

```
            applicationContext = new ApplicationContext();
```

```
        }
```

```
        //Захват мышкой и перемещение
```

```
        private void Grid_MouseDown(object sender, MouseButtonEventArgs e)
```

```
        {
```

```
            this.DragMove();
```

```
        }
```

```
        //Выход из приложения
```

```
        private void Exit_Click(object sender, RoutedEventArgs e)
```

```
        {
```

```
            this.Close();
```

```
        }
```

```
        //Переход к другому окну если аккаунт уже есть
```

```
        private void Swipe_Auth_Click(object sender, RoutedEventArgs e)
```

```
        {
```

```
            AuthWindow auth = new AuthWindow();
```

```
            auth.Show();
```

```
            this.Close();
```

```
        }
```

```
        //Кнопка регистрации
```

```
        private void Reg_Click(object sender, RoutedEventArgs e)
```

```
        {
```

```
            //Получение логина
```

```
            string login = Login.Text.Trim();
```

```
            //Получение пароля
```

```
            string pass1 = Pass1.Password.Trim(); ;
```

```
            //Повтор пароля
```

```
            string pass2 = Pass2.Password.Trim(); ;
```

```
            CheckandRegistr(login, pass1, pass2);
```

```

    }
    //Проверка полей и регистрация в приложении
    public void CheckandRegistr(string login,string pass1,string pass2)
    {
        //Проверка условий логина
        if (login.Length <= 3)
        {
            Login.ToolTip = "Имя должно быть длинее(минимум 3 буквы)";
            MessageBox.Show("Проверьте введенные данные");
        }
        //Проверка условий пароля
        else if (pass1.Length < 5)
        {
            Pass1.ToolTip = "Пароль должен быть длинее 5 символов";
            MessageBox.Show("Проверьте введенные данные");
        }
        //Проверка на совпадение паролей
        else if (pass1 != pass2)
        {
            Pass2.ToolTip = "Пароли не совпадают";
            MessageBox.Show("Проверьте введенные данные");
        }
        //Если все поля корректны
        else
        {
            Login.ToolTip = "";
            Pass1.ToolTip = "";
            Pass2.ToolTip = "";
            //Регистрация нового пользователя
            User user = new User(login, pass1);
            //Добавление клиента в список
            applicationContext.Users.Add(user);
            //Сохранение изменений
            applicationContext.SaveChanges();
            //Сообщение об успешной регистрации
            MessageBox.Show("Регистрация пройдена успешно!");
            //переход на другое окно
            AuthWindow authWindow = new AuthWindow();
            authWindow.Show();
            this.Close();
        }
    }
}
}
}

```

DBMS

//Класс модели

```
class User
{
    //Id клиента(уникальный)
    public int id { get; set; }
    //Поля логина и пароля
    private string login, password;
    public string Login
    {
        get => login;set
        {
            login = value;
        }
    }
    public string Password
    {
        get => password; set
        {
            password = value;
        }
    }
    //Конструктор по умолчанию
    public User () { }
    //Конструктор для передачи логина и пароля
    public User(string login,string password)
    {
        this.login = login;
        this.password = password;
    }
}
```

//Класс контекст

```
class ApplicationContext :DbContext
{
    public DbSet<User> Users { get; set; }
    public ApplicationContext() : base("Default") { }
}
```

Раздел 4. Руководство пользователя.

Для начала работы с приложением должен быть запущен сервер.

Скриншот запущенного сервера на рисунке 14.

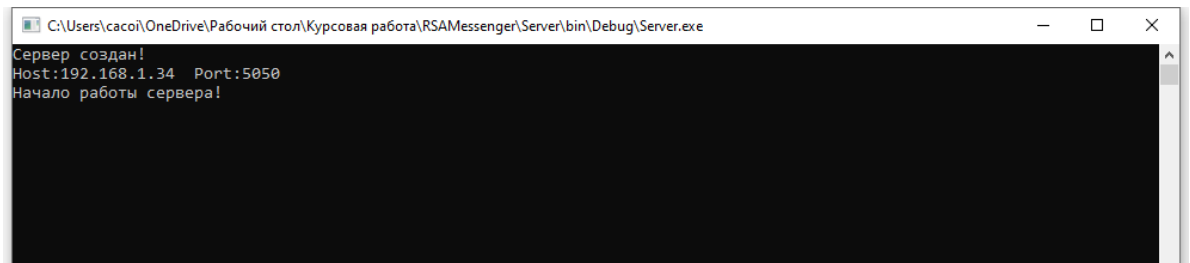


Рисунок 14.

После того как сервер был запущен пользователи могут пользоваться чатом.

При запуске приложения открывается окно авторизации (рисунок 15).

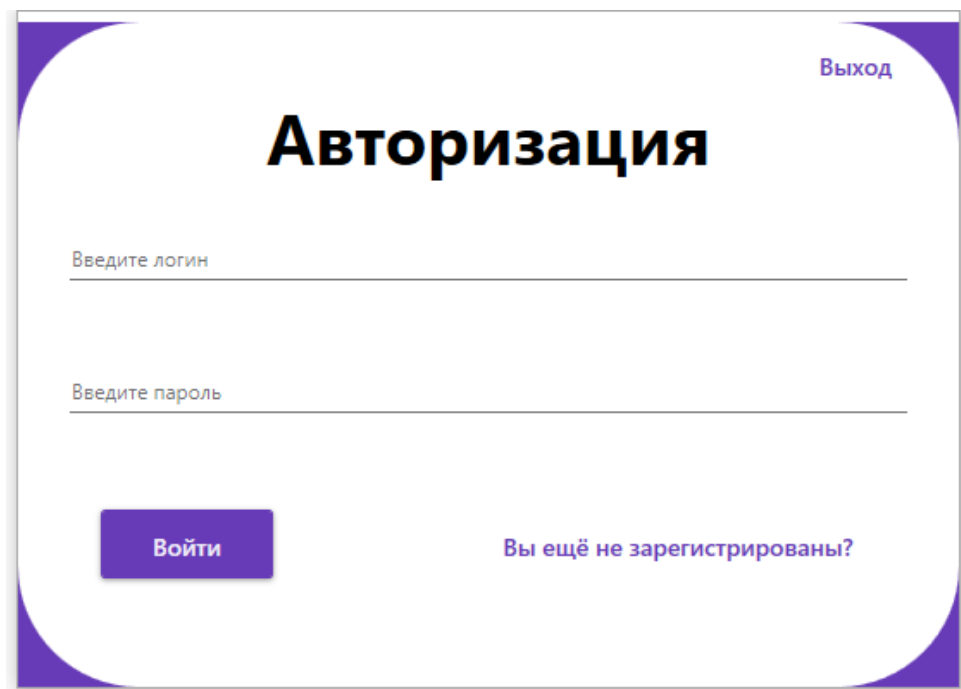
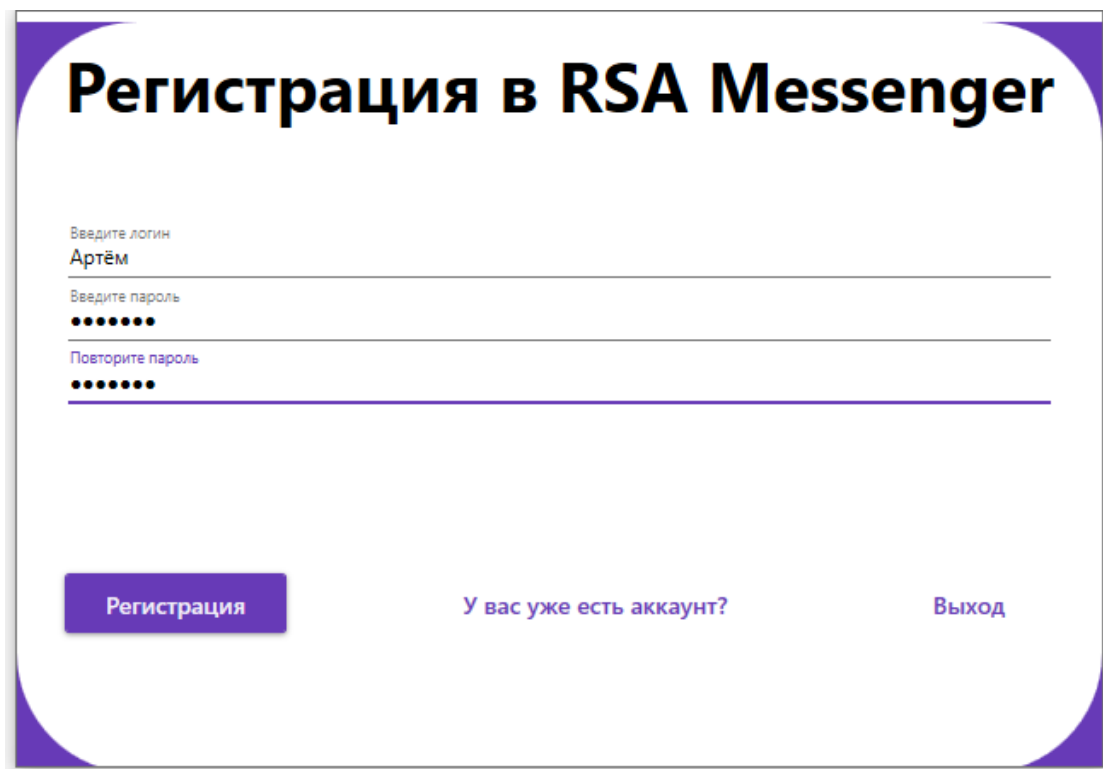


Рисунок 15.

В том случае, если у клиента нет аккаунта в данном приложении, он может зарегистрировать его нажав на кнопку «Вы еще не зарегистрированы?», после этого пользователя перекинет на другое окно.



The image shows a registration window titled "Регистрация в RSA Messenger". It features three input fields: a login field with the placeholder "Введите логин" and the text "Артём", a password field with the placeholder "Введите пароль" and masked dots, and a repeat password field with the placeholder "Повторите пароль" and masked dots. At the bottom, there are three buttons: "Регистрация" (Registration), "У вас уже есть аккаунт?" (Do you already have an account?), and "Выход" (Exit).

Регистрация в RSA Messenger

Введите логин
Артём

Введите пароль
••••••••

Повторите пароль
••••••••

Регистрация У вас уже есть аккаунт? Выход

Рисунок 16. Окно регистрации

Каждое поле имеет свою подсказку, которые высветятся если пользователь введет некорректные данные, например, логин должен быть длиннее 3 символов. В случае успешной регистрации высветится окно с соответствующим сообщением (рисунок 17).

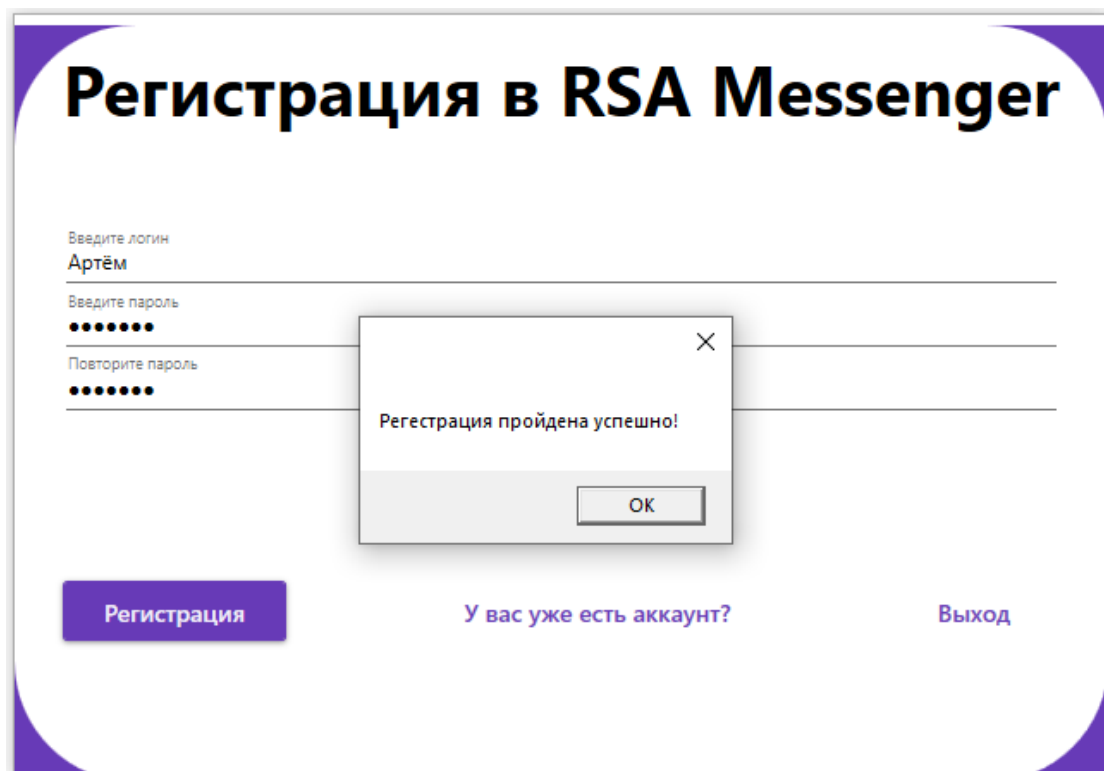


Рисунок 17.

После успешной регистрации клиента перекидывает на окно авторизации. Теперь новый пользователь может ввести свои данные в окне авторизации. Если будет допущена ошибка высветится следующее сообщение (рисунок 18).

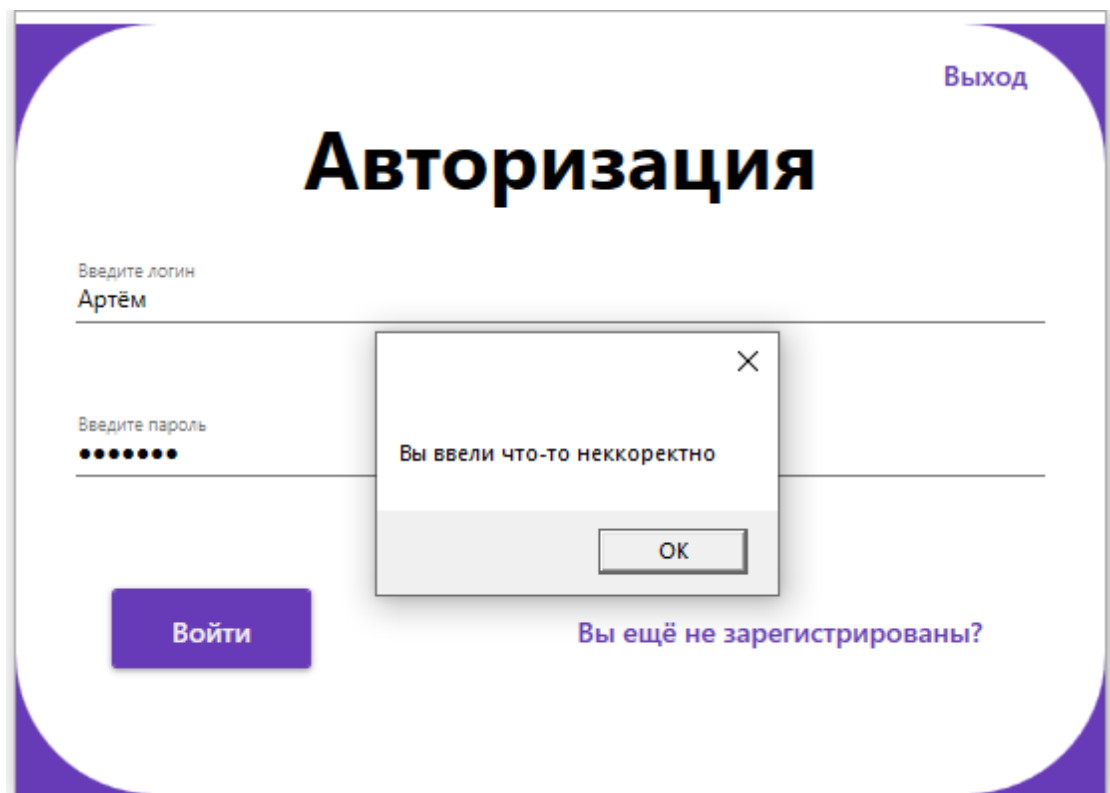


Рисунок 18.

Все зарегистрированные пользователи хранятся в базе данных SQLite.

Таблица: **Users**

	id	login	password
Фи...	Фильтр	Фильтр	Фильтр
1	1	TestUser	12345
2	2	TestUser2	qwert
3	3	TestUsers	123qwe
4	4	Ruslan	ruslanpass
5	5	Adel	adelpass
6	6	12345	12345
7	7	TestUser3	123qwe
8	8	Артём	yaartem

1 - 8 из 8

Перейти к: 1

Рисунок 19. Таблица с пользователями из базы данных.

После успешной авторизации открывается окно мессенджера (рисунок 20).

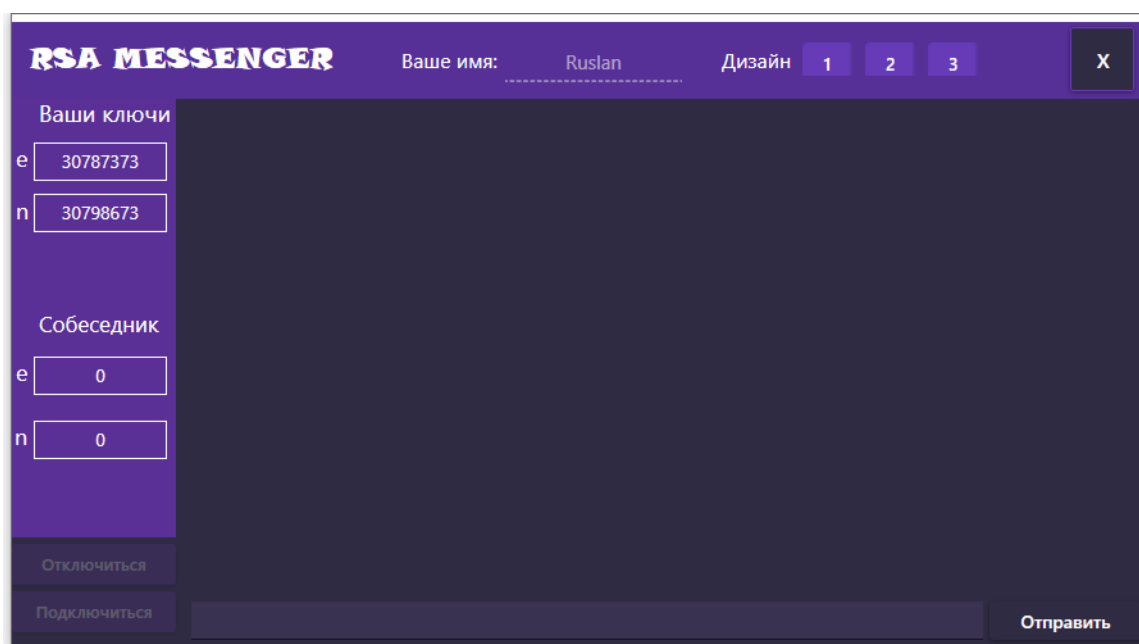


Рисунок 20.

Пользователю на выбор предоставлено 3 вида дизайна, которые можно в любой момент поменять (дизайн №1 запускается по умолчанию).

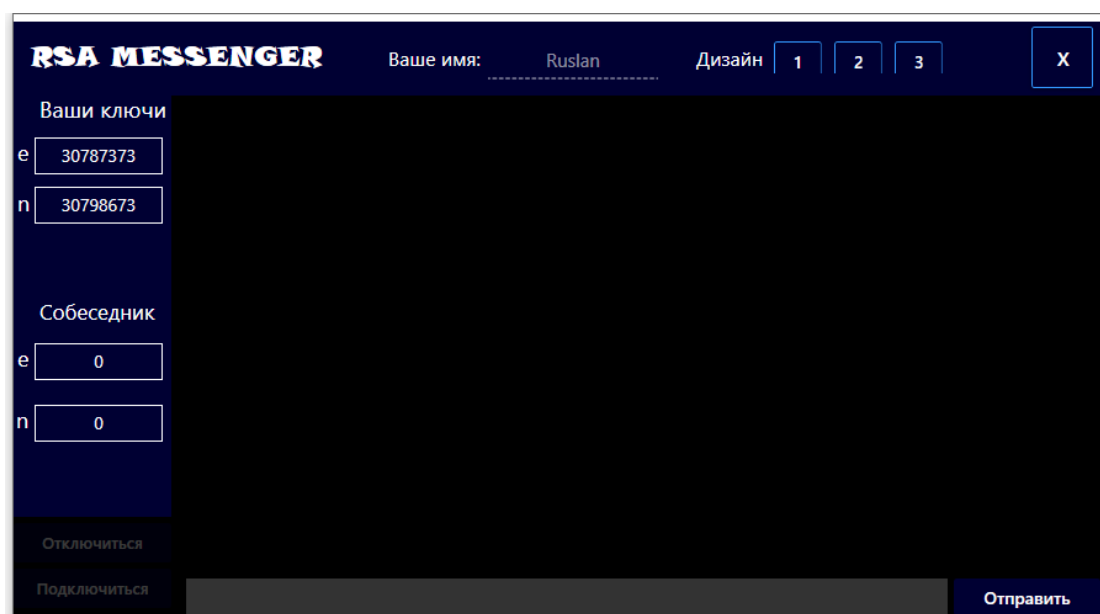


Рисунок 21. Дизайн №2.

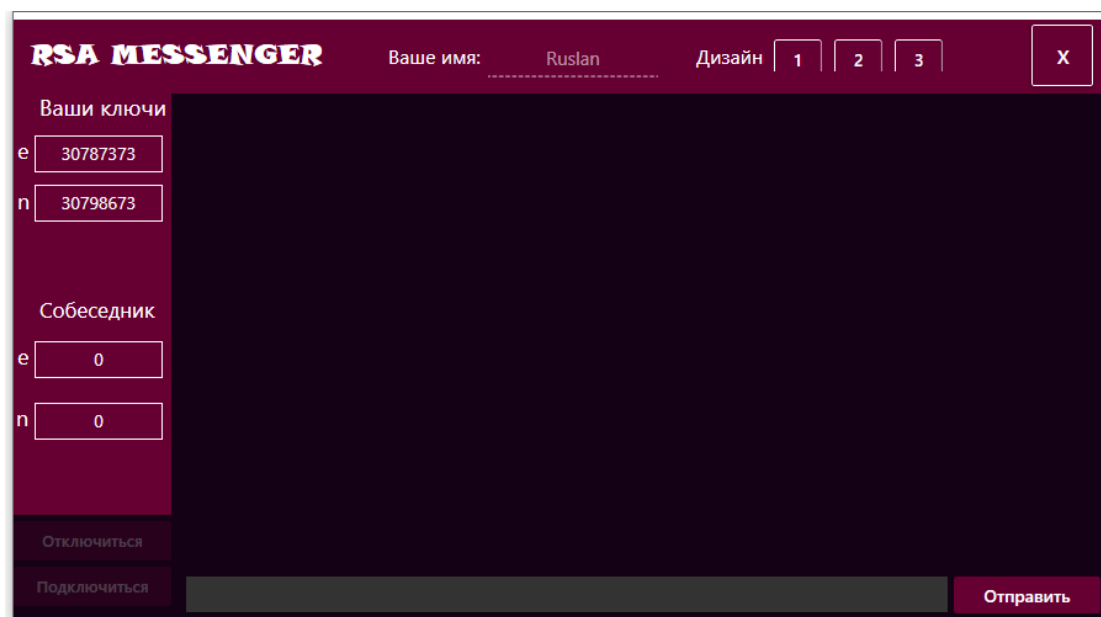


Рисунок 22. Дизайн №3.

Для того чтобы начать диалог с другим пользователем, собеседникам необходимо обменяться публичными ключами, после чего нажать на кнопку подключиться. Пример на рисунках 23 и 24.

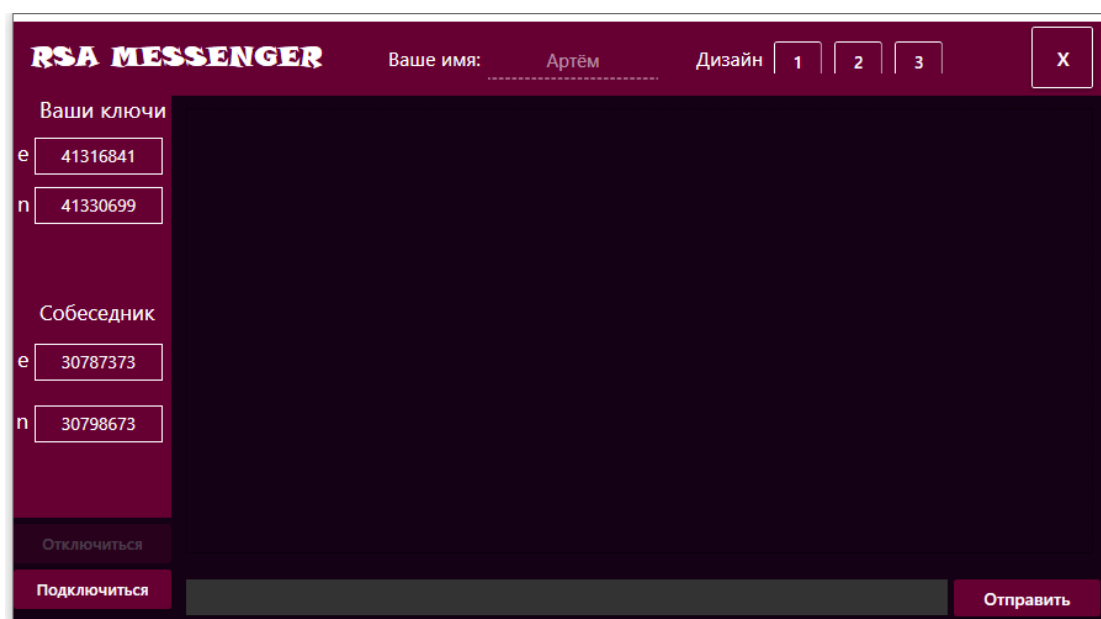


Рисунок 23.

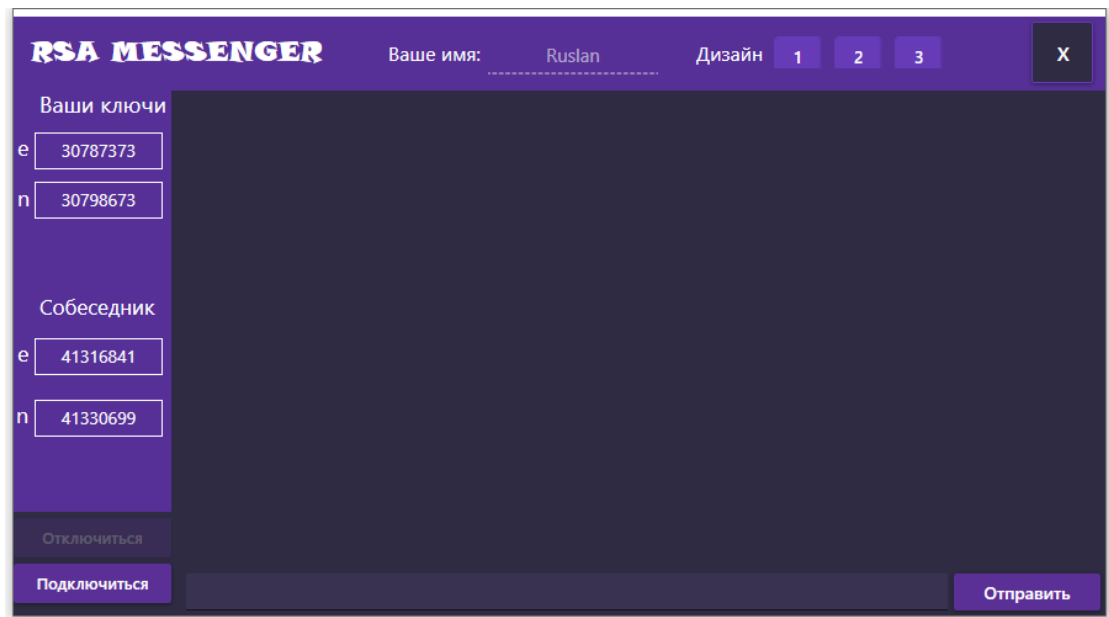


Рисунок 24.

Если ключи введены неправильно, соединение с сервером будет разорвано. При успешном подключении, соответствующее сообщение высветиться на консоли сервера, а также на экранах пользователей чата.

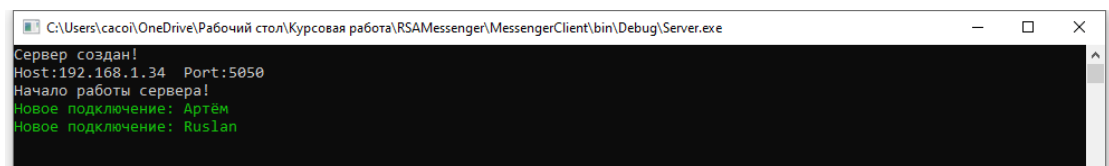


Рисунок 25. Сервер при подключении клиентов

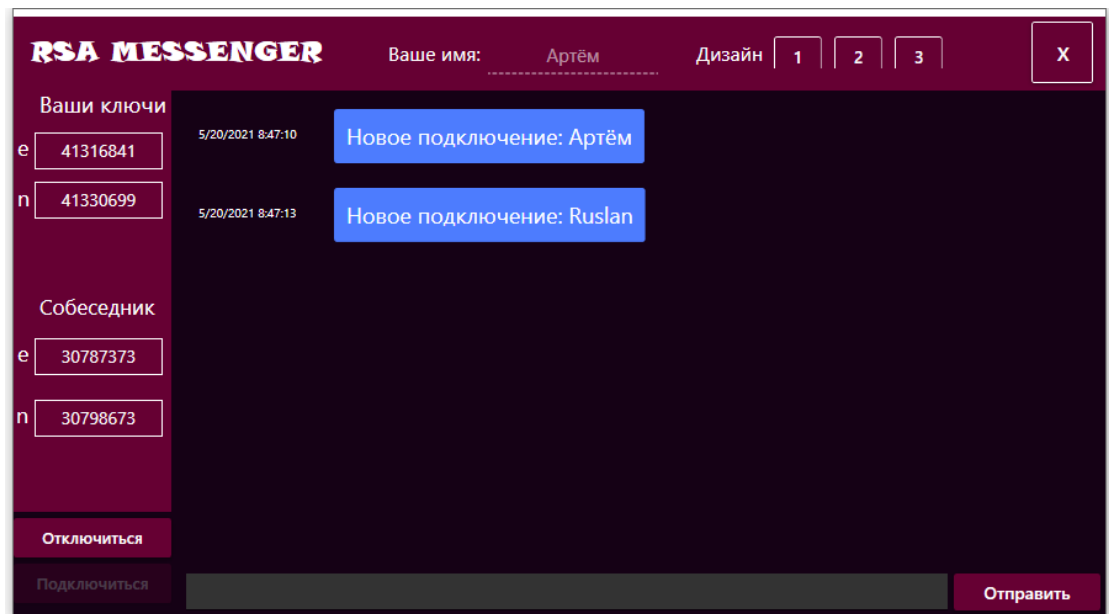


Рисунок 26. Интерфейс пользователей.

Теперь каждое сообщение пользователей на сервере будет зашифровано с помощью RSA.

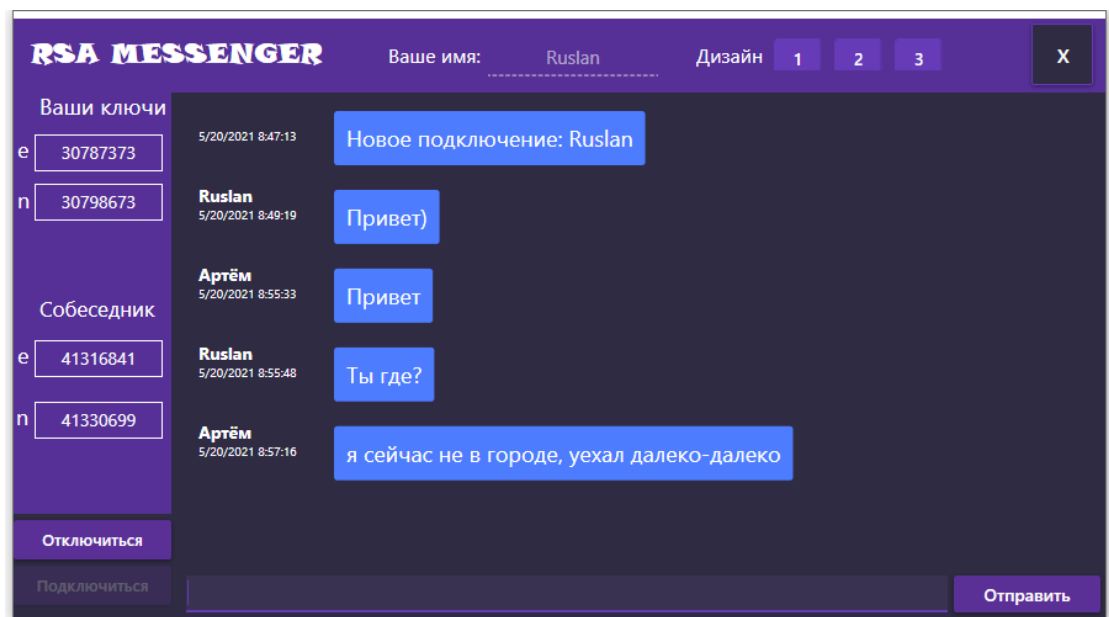


Рисунок 27. Сообщение 1 клиента

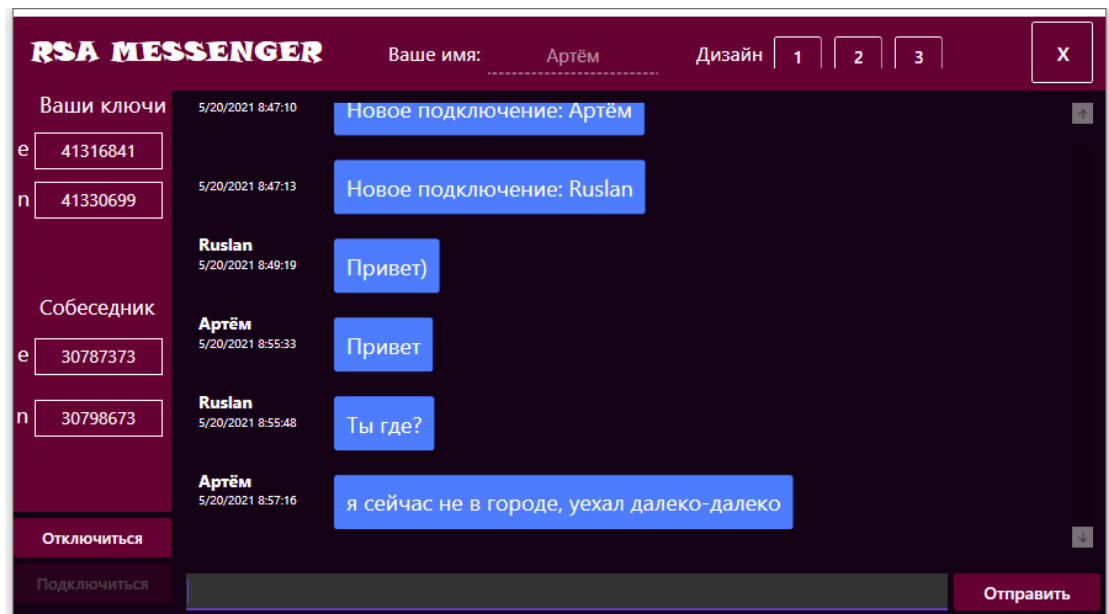


Рисунок 28. Сообщение 2 клиента(собеседник)

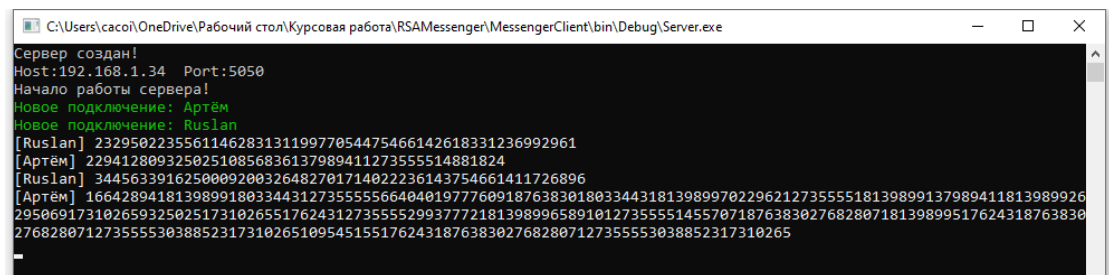


Рисунок 29. Сообщения на сервере.

Отключиться от сервера можно 2 способами, нажав на кнопку отключиться или нажав кнопку закрыть.

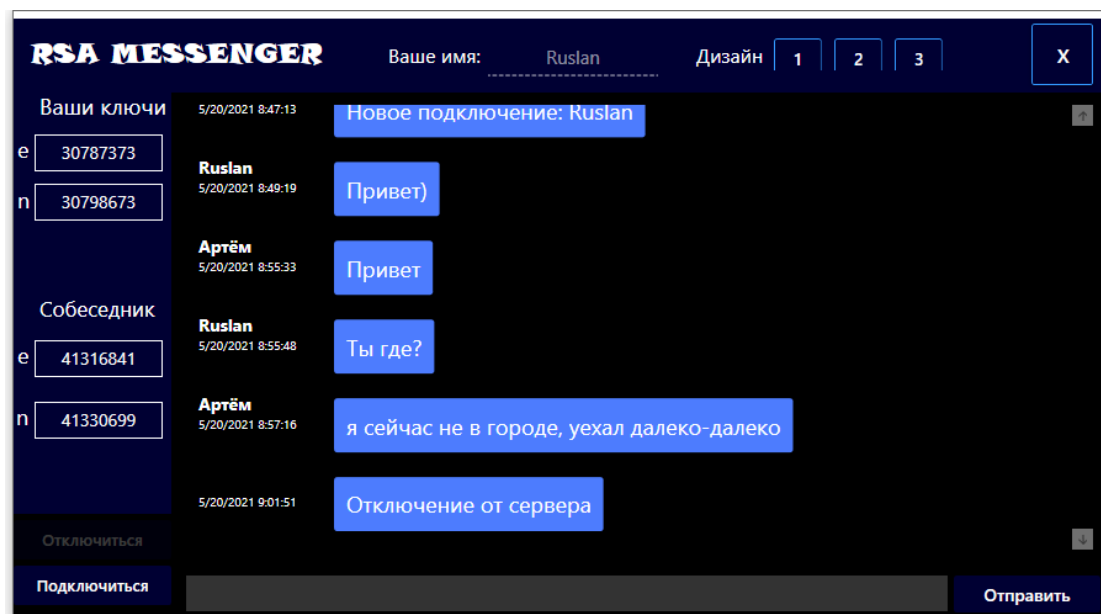


Рисунок 30. Отключение от сервера

После отключения и закрытия окна мессенджера, будет выполнен выход из аккаунта, пользователи вновь перекинет на окно авторизации, откуда он и сможет закрыть приложение.

Вывод

Для обмена зашифрованными сообщениями через сервер, который работает по протоколу TCP, была написана библиотека с алгоритмами RSA шифрования, в котором с помощью различных методов генерируются публичный и приватный ключи. Так же был написан сам сервер. Была добавлена база данных SQLite для регистрации пользователей в приложении. В самом клиенте были реализованы методы подключения и отключения от сервера. Методы отправки сообщения, расшифровка, шифрование данных. Было реализовано 3 вида дизайна на выбор пользователя.