СОБА ЗА ЋАСКАЊЕ
CHAT ROOM

Вук Вукашиновић

VIII, ОШ 'Илија Вирчанин', Регионални центар за таленте Београд 1 – Земун

РЕЗИМЕ

Овај рад представља апликацију 'Чат соба' - комуникациону апликацију развијену у чистом C# која омогућава реалновременску размену порука између више корисника. Апликација је дизајнирана тако да омогући једноставно повезивање клијент-сервер мрежног окружења, и са својим модуларним приступом лако се надограђује. Рад објашњава функције апликације, начин рада, кориснички интерфејс, као и структуру и значајне линије кода.

КЉУЧНЕ РЕЧИ:
чат соба, комуникација, C#, клијент-сервер архитектура, апликација

ABSTRACT:
This paper introduces the “Chat Room” application – a communication tool developed in pure C# that enables real-time message exchange among multiple users. The application is designed to facilitate simple connection between client and server modules and can easily be upgraded with its modular approach. The paper explains the functionalities, user interface, core code structure, and key components of the application.

KEYWORDS:
chat room, communication, C#, client-server architecture, application

Увод

Апликација која је предмет овог рада омогућава комуникацију више корисника у реалном времену преко интернета. Апликација је развијена унутар C# окружења уз помоћ .NET библиотека. Ово омогућава брзу и стабилну конекцију. Апликација је приближена корисницима израдом корисничког интерфејса унутар *Windows Forms.* Овај рад је примена наученог знања са часова програмирања и самосталног истраживања.

Настанак идеје

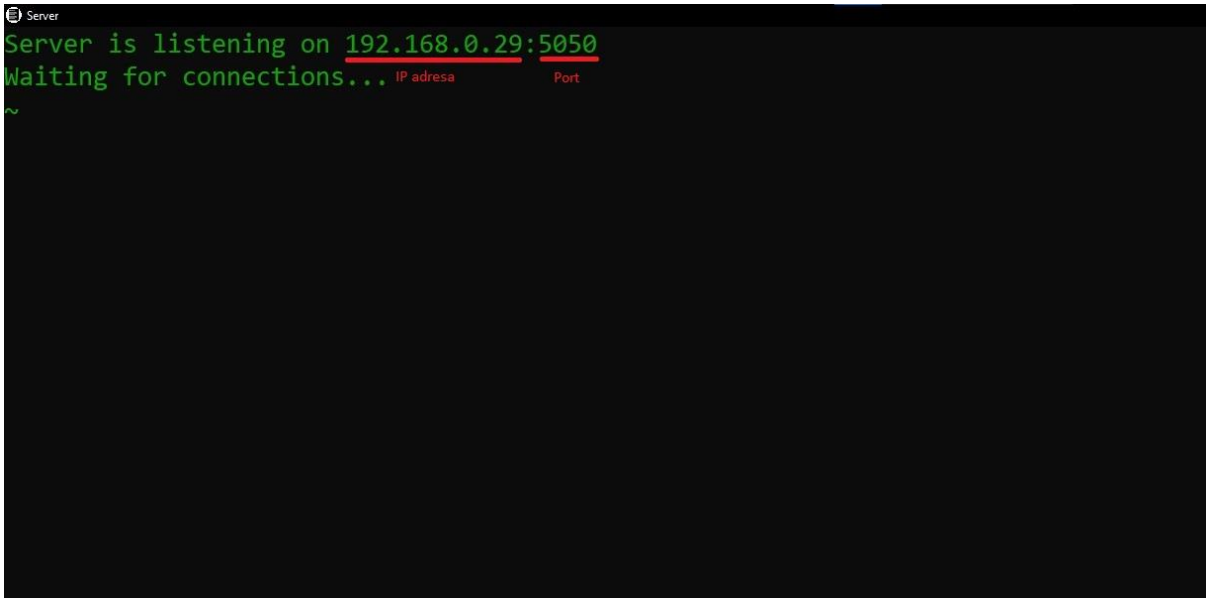
Један дан дописивао сам се са другом, запитао сам се како све то функционише. Како се врши размена порука, фајлова, видео и аудио разговора. Кренуо сам да се распитујем по интернету и да откривам како све то функционише. У мени се развила жеља да покушам да то све рекреирам.

Почео сам тиме што сам направио 'Сервер' и 'Клијент' код који су могли у минималном смислу да комуницирају један између другог. То је све било лимитирано на конзолу што као што можете да замислите није најприлагођеније корисницима. Тако да је на све то морала да се угради UI *(коринички интерфејс)*. То је знатно олакшало интеракцију са апликацијом

Рад са апликацијом

Ово поглавље представља упутство за кориснике

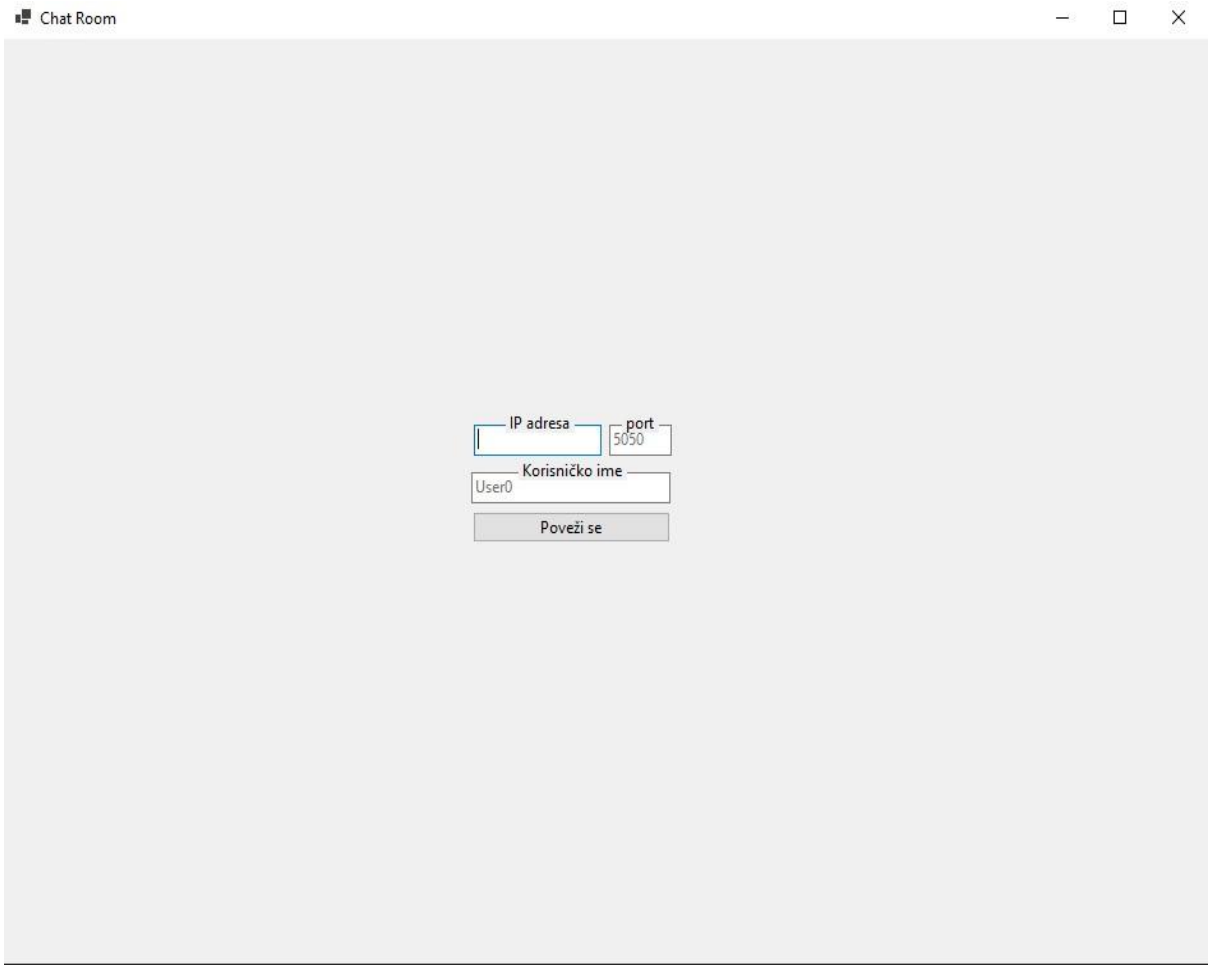
1. За почетак морамо да покренемо апликацију сервера преко које ће сви разговори да пролазе. Када се покрене имаћемо прозор (слика 1.1), исписаће локалну IP адресу и PORT на којој сервер ради, ово ће нам бити потребно да бисмо се повезали. Напомена уколико се у подешавањима рутера не подеси *port forwarding* конекције ће бити могуће унутар једне интернет везе, уколико је *port forwarding* подешен уместо исписане IP адресе мораће да се користи јавна адреса. Један од начина да се она сазна је да посетите What is my ip address.



Слика 1.1 Конзола сервера

2. Након тога покрећемо Chat Room апликацију и у њој попуњавамо поља са слике (слика 2.1), IP adresa I PORT су адреса и порт нашег сервера, а у претходном кораку смо видели како да их сазнамо. Корисничко име само говори, уносите име под којим ће вас други људи у ћаскању видети, и кликнемо дугме “Повежи se”.

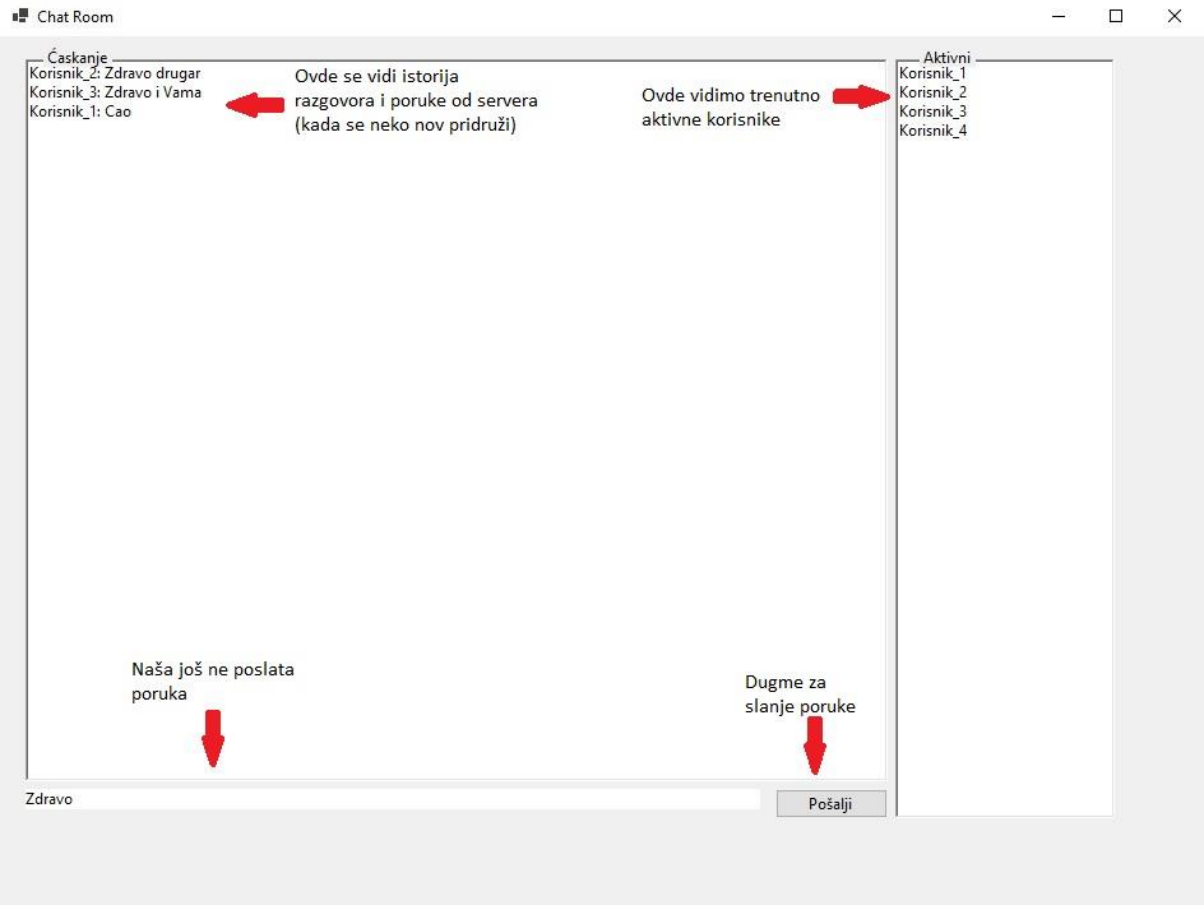
Слика 2.1 Изглед



почетног менија

3. Након повезивања имамо следећи мени (слика 2.2) и уједно и главни мени у коме се врши дописивање. У њему можемо видети разговор, тренутно активне кориснике, и шаљемо поруке

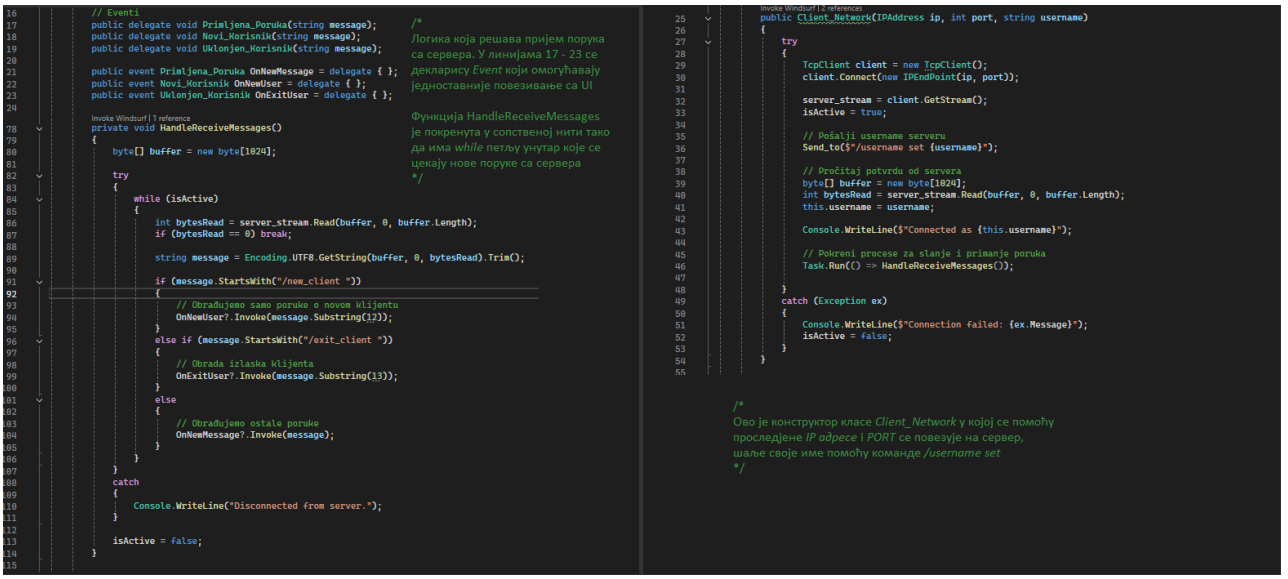
Слика 2.2 Изглед главног менија



Опис Апликације

Апликација је развијена у чистом C# уз коришћењ .NET оквира. Примарни фокус је био на дизајнирању сервер-клијент конекције који је накнадно надограђен да користи UI. Апликацију би поделио у два главна дела, сервер и клијент класе. Конекција је базирана на *socket* и TCP протоколом. Постоје разне библиотеке које су ово имплементирале тако да је лакше да се користи и на бољи начин али ја сам изабрао да радим овако јер се на овај начин много више може научити рад са *socket-има.*

- Клијент: Ова класа има за задатак да успостави и одржава конекцију са сервером



```
// Eventi
public delegate void Primljena_Poruka(string message);
public delegate void Novi_Korisnik(string message);
public delegate void Uklonjen_Korisnik(string message);

public event Primljena_Poruka OnNewMessage = delegate { };
public event Novi_Korisnik OnNewUser = delegate { };
public event Uklonjen_Korisnik OnExitUser = delegate { };

Invoke Windsurf | 1 reference
private void HandleReceiveMessages()
{
    byte[] buffer = new byte[1024];

    try
    {
        while (isActive)
        {
            int bytesRead = server_stream.Read(buffer, 0, buffer.Length);
            if (bytesRead == 0) break;

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

            if (message.StartsWith("/new_client "))
            {
                // Obrađujemo samo poruke o novom klijentu
                OnNewUser?.Invoke(message.Substring(12));
            }
            else if (message.StartsWith("/exit_client "))
            {
                // Obrada izlaska klijenta
                OnExitUser?.Invoke(message.Substring(13));
            }
            else
            {
                // Obrađujemo ostale poruke
                OnNewMessage?.Invoke(message);
            }
        }
    }
    catch
    {
        Console.WriteLine("Disconnected from server.");
    }

    isActive = false;
}
```

/*
Логика која решава пријем порука са сервера. У линијама 17 - 23 се декларису *Event* који омогућавају једноставније повезивање са UI

Функција HandleReceiveMessages је покренута у сопственој нити тако да има *while* петљу унутар које се цекају нове поруке са сервера
*/

```
Invoke Windsurf | 2 references
public Client_Network(IPAddress ip, int port, string username)
{
    try
    {
        TcpClient client = new TcpClient();
        client.Connect(new IPEndPoint(ip, port));

        server_stream = client.GetStream();
        isActive = true;

        // Pošalji username serveru
        Send_to($"/username set {username}");

        // Pročitaj potvrdu od servera
        byte[] buffer = new byte[1024];
        int bytesRead = server_stream.Read(buffer, 0, buffer.Length);
        this.username = username;

        Console.WriteLine($"Connected as {this.username}");

        // Pokreni procese za slanje i primanje poruka
        Task.Run(() => HandleReceiveMessages());

    }
    catch (Exception ex)
    {
        Console.WriteLine($"Connection failed: {ex.Message}");
        isActive = false;
    }
}
```

/*
Ово је конструктор класе *Client_Network* у којој се помоћу прослеђене *IP адресе* и *PORT* се повезује на сервер, шаље своје име помоћу команде */username set*
*/

Слика 3.1 Структура Клијента

- Сервер: Задужен за обраду клијената прослеђивање порука.

на слици 4.1 можемо видети класу која је представља клијенте на серверу (15-31), и функцију која обрађује везу са корисницима (107-171). На слици 4.2 можемо видети функцију која обрађује кориснички унос у конзоли, и видимо *Main* функцију која има

задатак да покрене сервер.

```
Invoke Windsurf | 11 references
private class Client     // Класа која представља корисникове податке
{
    private static uint curect_id = 0;  // Глобални тренутни ID који ће
                                        бити постављен кориснику, такође
    public TcpClient TcpClient;         у случају да нема username биће
    public NetworkStream stream;        му исти као ID. Пошто је static биће
    public string username;             исти за све кориснике
    public uint ID;

    Invoke Windsurf | 1 reference
    public Client(TcpClient tcpClient)
    {
        TcpClient = tcpClient;
        stream = TcpClient.GetStream();
        ID = curect_id++;
        username = "Guest" + ID;
    }
}

private static void HandleClient(Client client)
{
    // Obaveštavanje novog klijenta da se povezao i slanje liste postojećih korisnika.
    Send_to_All(client, $"/new_client {client.username}", false); // Novi klijent se obaveštava da se povezao.

                                                                  /*
    foreach (Client c in ActiveClient)                            Шаље списак активних коисника новом члану као би
    {                                                             мога да их забележи
        Console.WriteLine($"Sending /new_client to {c.username}...");  Тајмер од 25ms како бих корисник стигао да обради
        Send_to(client, $"/new_client {c.username}", default_encoder);
        Thread.Sleep(25);
    }

    Send_to_All(client, $"[SERVER]: {client.username} joined the chat.");   Шаље у chat информацију о новом кориснику
                                                                  */
    while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)   /*
    {                                                             Овај део функције прима
        if (bytesRead == 0) break;                                поруке од корисника и
                                                                  проследјује је даље.
        string receivedMessage = Recv_from(buffer, bytesRead, default_encoder);


        if (HandleCommand(client, receivedMessage) == "OK")       Или у случају да је команда
        {                                                         онда је обради
            continue;
        }


        Console.WriteLine($"Received: {client.username}: {receivedMessage.Trim()}");   */
        Send_to_All(client, receivedMessage);
    }
}
```

Слика 4.1 Структура класе која чува податке клијената и обрада клијената

```
184     private static void HandleAdminInput(TcpListener server)
185     {
186         while (serverRunning)
187         {
188             Console.Write("~ ");
189             string adminInput = Console.ReadLine().ToLower();
190 
191             if (EXIT_WORDS.Contains(adminInput))
192             {
193                 Console.WriteLine("Shutting down server...");
194                 serverRunning = false;
195                 lock (clientListLock)
196                 {
197                     foreach (Client client in ActiveClient)
198                     {
199                         Send_to(client, "[SERVER] Server is shutting down. Disconnected.", default_encoder);
200                         client.TcpClient.Close();
201                     }
202                     ActiveClient.Clear();
203                 }
204                 server.Stop();
205                 break;
206             }
207         }
208     }
209 
210     static void Main(string[] args)
211     {
212 
213         IPAddress ipAddress = IPAddress.Any;
214         int ipPort = 5050;
215         TcpListener server = new TcpListener(ipAddress, ipPort);
216         server.Start();
217 
218         // Dobijanje lokalne IP adrese klijenta, u slucaju da je rezultat null postavljamo Unknown IP
219         string localIP = Dns.GetHostEntry(Dns.GetHostName())
220             .AddressList
221             .FirstOrDefault(ip => ip.AddressFamily == AddressFamily.InterNetwork)
222             ?.ToString()     // ?  poziva samo ako vrednost nije null
223             ?? "Unknown IP"; // ?? postavlja vrednost samo ako je null
224 
225 
226         Console.WriteLine($"Server is listening on {localIP}:{ipPort}");
227         Console.WriteLine("Waiting for connections...");
228 
229         Task.Run(() => HandleAdminInput(server));
230 
231         while (serverRunning)
232         {
233             try
234             {
235                 Client client = new Client(server.AcceptTcpClient());
236                 lock (clientListLock)
237                 {
238                     ActiveClient.Add(client);
239                 }
240                 Task.Run(() => HandleClient(client));
241             }
242             catch (SocketException)
243             {
244                 Console.WriteLine("Server has stopped accepting new connections.");
245                 break;
246             }
247         }
248         Console.WriteLine("Server shut down.");
249 
250     }
251 }
```

// Чита унос из конзоле и ако се погађа са речима за гашење (exit, quit) сервер се гаси

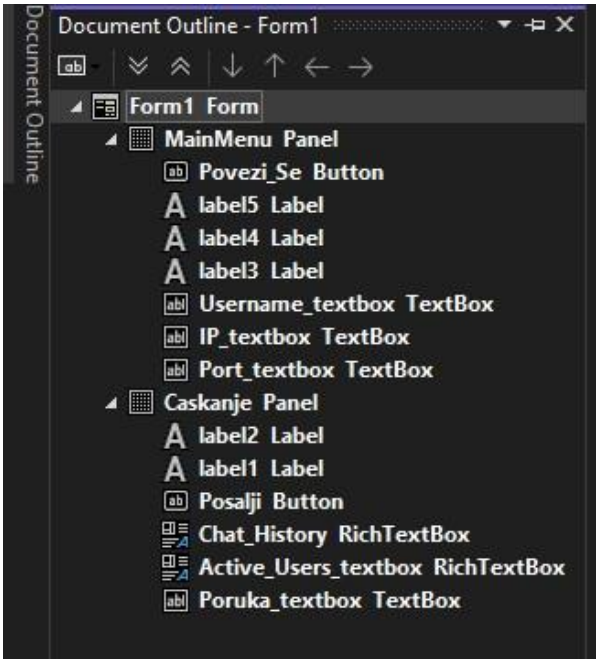
// Потом иде кроз листу активних клијената и *disconect-uje* их

// И на крају гаси *socket* и престаје да очекује нове клијенте

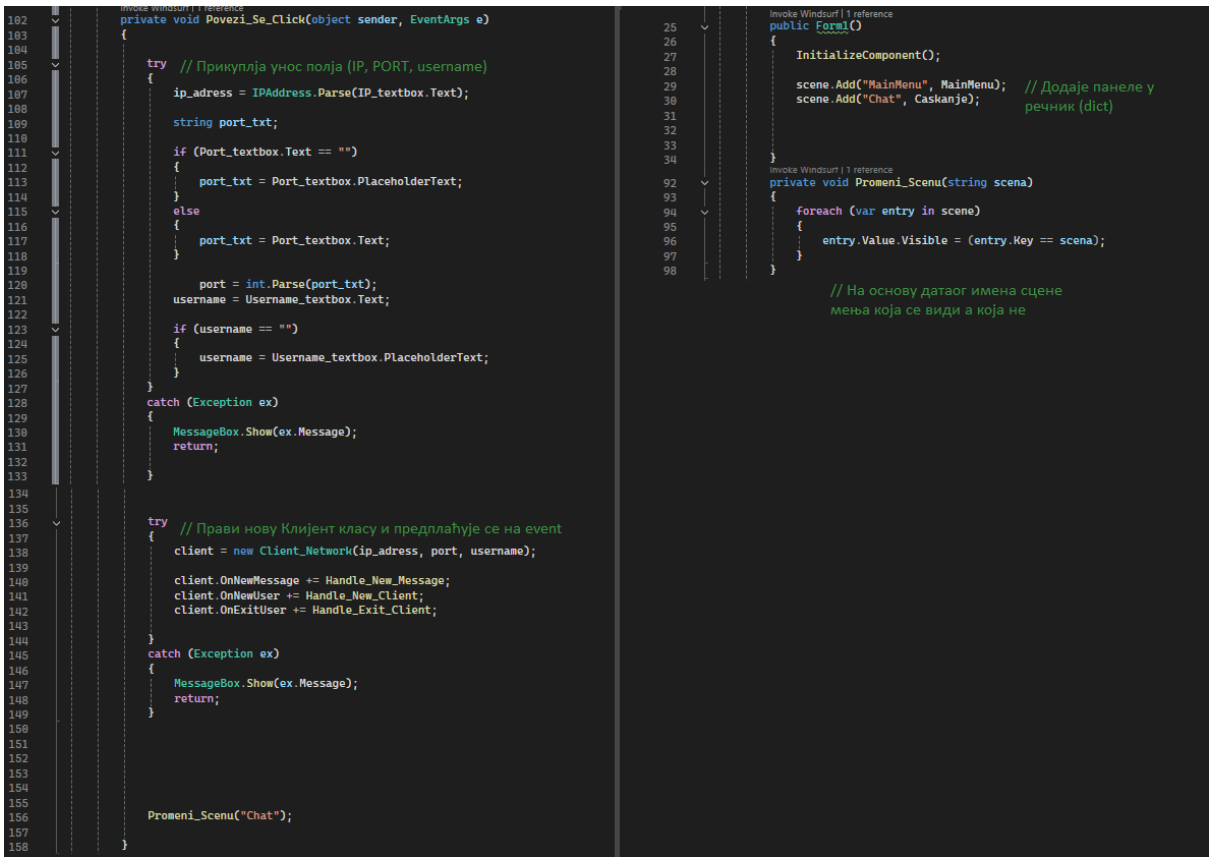
/*
Прихвата нове клијенте, додаје их у листу активних и покреће нову нит у којој се клијент обрађује
*/

Слика 4.2 Обрада корисничког уноса и функција за покретање сервера

- UI: То је само спајање Windows Forms и класе Client_Network (слике 5.2). Како бих лако могао да мењам 'сцене' додао сам два панела, један је, видљив други није када се мења 'сцена' само се ротирају који је панел видљив, а који није (слика 5.1).



Слика 5.1 Приказ WF дизајна



```
private void Povezi_Se_Click(object sender, EventArgs e)
{

    try  // Прикупља унос поља (IP, PORT, username)
    {
        ip_adress = IPAddress.Parse(IP_textbox.Text);

        string port_txt;

        if (Port_textbox.Text == "")
        {
            port_txt = Port_textbox.PlaceholderText;
        }
        else
        {
            port_txt = Port_textbox.Text;
        }

            port = int.Parse(port_txt);
        username = Username_textbox.Text;

        if (username == "")
        {
            username = Username_textbox.PlaceholderText;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;

    }


    try  // Прави нову Клијент класу и предплаћује се на event
    {
        client = new Client_Network(ip_adress, port, username);

        client.OnNewMessage += Handle_New_Message;
        client.OnNewUser += Handle_New_Client;
        client.OnExitUser += Handle_Exit_Client;

    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }




    Promeni_Scenu("Chat");

}
```

```
Invoke Windsurf | 1 reference
public Form1()
{
    InitializeComponent();

    scene.Add("MainMenu", MainMenu);
    scene.Add("Chat", Caskanje);        // Додаје панеле у речник (dict)


}
Invoke Windsurf | 1 reference
private void Promeni_Scenu(string scena)
{
    foreach (var entry in scene)
    {
        entry.Value.Visible = (entry.Key == scena);
    }
}

    // На основу датаог имена сцене мења која се види а која не
```

Слика 5.2 Функција за повезивање на сервер и Промена 'сцене'

ЗАКЉУЧАК

Апликација 'Чат соба' је развијена са циљем истраживања методологије за комуникацију у реалном времену преко мреже. Пројекат је пружио практично искуство у раду са мрежним протоколима, сокет програмирању, и изради графичко корисничког интерфејса.

Апликација је врло забавна и функционална, нуди разна поља за усавршавање, попут оптимизација и безбедности.

Уколико сте заинтересовани за цео код, и да пробате апликацију можете то урадити на GitHub страници: Chat Room

Захвалница

Захваљујем се мојим наставницима Рељи Ђурчин и Дијани Стефановић који су ми објаснили било које недоумице.

Литература

1. Званична документација за .NET: docs.microsoft.com