



UNIP - UNIVERSIDADE PAULISTA
CIENCIA DA COMPUTAÇÃO

ALYSSON MOREIRA COSTA - N4352J-9
AUGUSTO BENJAMYN LEAL MARTINS - N406BG-0
CARLOS ALBERTO LEMOS MARTINS OLIVEIRA – F03677-4
FABRÍCIO OLIVEIRA DIAS - D96323-6
YURY RODRIGUES SHELKOVSKY - F06216-3

PROJETO C# DE CHAT TCP/IP
CHAT SUSTENTÁVEL

SOROCABA
2021

UNIP - UNIVERSIDADE PAULISTA
CIENCIA DA COMPUTAÇÃO

ALYSSON MOREIRA COSTA - N4352J-9
AUGUSTO BENJAMYN LEAL MARTINS - N406BG-0
CARLOS ALBERTO LEMOS MARTINS OLIVEIRA – F03677-4
FABRÍCIO OLIVEIRA DIAS - D96323-6
YURY RODRIGUES SHELKOVSKY - F06216-3

PROJETO C# DE CHAT TCP/IP
CHAT SUSTENTÁVEL

Apresentação do Tema de APS, apresentado a UNIP - UNIVERSIDADE PAULISTA da cidade de Sorocaba, como exigência parcial à obtenção da conclusão da disciplina de APS do quinto semestre em Ciência da Computação.

Orientador(a): Esp. Reverdan Springer

SOROCABA
2021

RESUMO

Este trabalho tem como objetivo desenvolver um chat com funcionamento em tempo real TCP/IP utilizando sockets de Berkeley com a linguagem de programação C#, pois além de se mostrar prática, contém um extenso catálogo de bibliotecas o que atende perfeitamente aos requisitos do projeto além de dar praticidade ao código. Possuindo como tema principal a sustentabilidade, sua proposta é promover a comunicação entre polos de uma reserva florestal, criando assim um projeto com dois programas, um deles responsável por criar e gerenciar um servidor, e outro para o cliente se conectar a esse servidor para que ocorra a troca de mensagens. Foi então desenvolvido um chat pensado de forma a ser útil, tendo uma conexão simples (necessitando apenas que os computadores estejam numa mesma rede, VPN ou domínio), com visual que remete a proposta apresentada, com botões, ícones temáticos e emojis voltados ao tema sustentabilidade, com símbolos de reciclagem, árvores, etc. Com o término de seu desenvolvimento a proposta do tema foi atendida e bem executada, o chat além de funcional, é simples e agradável aos olhos, seu desenvolvimento foi árduo, mas sua dificuldade retornou-nos frutos compensadores, pois através dele foi possível perceber complexidade de se criar uma aplicação com comunicação por rede, assim como seus tratamentos.

Palavras-Chave: Servidor, Cliente, Sustentabilidade.

ABSTRACT

This academic-work aims to develop a chat with TCP/IP real time operation using Berkeley sockets with the C# programming language, because in addition to being practical, it contains an extensive library catalog which perfectly meets the requirements of the project in addition to make the code practical. Having sustainability as its main theme, its proposal is to promote communication between the poles of a forest reserve, thus creating a project with two programs, one of them responsible for creating and managing a server, and the another one for the client to connect to that server so that exchange of messages occurs. It was developed a chat designed to be useful, having a simple connection (requiring only that the computers are on the same network, VPN or domain), with a look that refers to the proposal presented, with buttons, thematic icons and emojis focused on the sustainability theme, with recycling symbols, trees, and so on. At the finish of the development the theme proposal was attended to and well executed, the chat is not only functional, it is simple and pleasing to our eyes, its development was arduous, but its difficulty returned us with rewarding fruits, because it was possible to notice the complexity of creating application with network communication, as well as its treatments.

Key-Words: Server, Client, Sustainability.

LISTA DE FIGURAS

Figura 1 - Classe StatusChangedEventArgs.cs linhas 5 à 15.....	11
Figura 2 - Classe Client.cs completa	12
Figura 3 - Classe Client.cs linhas 29 à 56	13
Figura 4 - Classe Client.cs linhas 57 a 73	14
Figura 5 - Classe Client.cs linhas 74 a 78	14
Figura 6 - Classe Validacao.cs linhas 17 a 44.....	15
Figura 7 - Layout de formulário da classe FormLogin.cs	16
Figura 8 - Formulário FormLogin.cs linhas 14 a 36	17
Figura 9 - Layout de formulário da classe FormChat.cs	17
Figura 10 - Representação visual do menu de emojis.....	18
Figura 11 - Formulário FormChat.cs linhas 16 a 23	19
Figura 12 - Formulário FormChat.cs linhas 51 a 62	19
Figura 13 - Classe Server.cs completa.....	20
Figura 14 - Classe Server.cs linhas 13 a 17	20
Figura 15 - Classe Server.cs linhas 29 a 39.....	21
Figura 16 - Classe Server.cs linhas 40 a 51	21
Figura 17 - Classe Server.cs linhas 52 a 59.....	21
Figura 18 - Classe Server.cs linhas 60 a 72.....	22
Figura 19 - Classe Server.cs linhas 73 a 85.....	22
Figura 20 - Classe Server.cs linhas 93 a 105.....	23
Figura 21 - Classe Server.cs linhas 106 a 122	23
Figura 22 - Classe ConexaoUsuario.cs completa.....	24
Figura 23 - Classe ConexaoUsuario.cs linhas 10 a 15	24
Figura 24 - Classe ConexaoUsuario.cs linhas 23 a 65	25
Figura 25 - Classe ConexaoUsuario.cs linhas 66 a 71	25
Figura 26 - Classe ConexaoUsuario.cs linhas 72 a 85	26
Figura 27 - Classe ConexaoUsuario.cs linhas 86 a 94	26
Figura 28 - Classe ConexaoUsuario.cs linhas 95 a 100	26
Figura 29 - Layout do formulário da classe FormServer.cs	27
Figura 30 - Formulário FormServer.cs linhas 16 a 18.....	28
Figura 31 - Formulário FormServer.cs linhas 20 a 61.....	28
Figura 32 - Funcionamento do Chat.....	29

Figura 33 - Erros tratados no formulário FormLogin.cs	30
Figura 34 - Erros tratados no FormLogin.cs recebidos do servidor	31
Figura 35 - Pop-up do Cliente ao perder a conexão com o server.....	31
Figura 36 - Pop-up de erro do programa Servidor	31
Figura 37 - Demonstração do relatório de conexão e arquivo de Backup criado	32
Figura 38 - Emojis disponíveis para uso do usuário conectado	32

SUMÁRIO

1	INTRODUÇÃO	8
2	REVISÃO DE LITERATURA	9
2.1	Redes.....	9
2.1.1	<i>Transmission control protocol (TCP)</i>	9
2.1.2	<i>Sockets.....</i>	10
3	METODOLOGIA.....	11
3.1	Utilização de eventos.....	11
3.1.1	<i>StatusChangedEventArgs.cs</i>	11
3.2	Programa Cliente	11
3.2.1	<i>Client.cs.....</i>	12
3.2.1.1	Método IniciarConexao()	13
3.2.1.2	Método EscutarMensagens()	14
3.2.1.3	Método EnviarMensagem()	14
3.2.2	<i>Validacao.cs</i>	14
3.2.2.1	Método ValidarDadosConexao()	15
3.2.3	<i>FormLogin.cs</i>	15
3.2.3.1	Método Conectar()	16
3.2.4	<i>FormChat.cs</i>	17
3.2.4.1	Uso De Emoticons	18
3.2.4.2	Método Enviar()	19
3.2.4.3	Método AtualizaLogMensagens().....	19
3.3	Programa Servidor.....	19
3.3.1	<i>Server.cs.....</i>	20
3.3.1.1	Construtor Server()	20
3.3.1.2	Método IniciarServidor().....	21
3.3.1.3	Método ManterServidor()	21
3.3.1.4	Método ValidarMensagem()	21
3.3.1.5	Método EnviarMensagem()	22
3.3.1.6	Método EnviarMensagemAdmin()	22
3.3.1.7	Método FecharServidor()	23
3.3.1.8	Método CriarBackup()	23
3.3.2	<i>ConexãoUsuario.cs.....</i>	24

3.3.2.1	Construtor ConexaoUsuario()	24
3.3.2.2	Método ValidarUsuario()	25
3.3.2.3	Método AceitarUsuario()	25
3.3.2.4	Método AguardarMensagem()	26
3.3.2.5	Método RemoverUsuario()	26
3.3.2.6	Método FechaConexao()	26
3.3.3	<i>FormServidor.cs</i>	27
3.3.3.1	Atributos da Classe	28
3.3.3.2	Evento Click btnConectar	28
4	RESULTADOS	29
4.1	Funcionamento do chat	29
4.2	Tratamento de erros	29
4.2.1	<i>Cliente</i>	29
4.2.2	<i>Servidor</i>	31
4.3	Funcionalidades Extras	32
5	CONCLUSÃO	33
	REFERÊNCIAS	34
	ANEXOS	36

1 INTRODUÇÃO

O conceito de sustentabilidade aborda a maneira como deve-se agir em relação a natureza, sendo seu conceito aplicável a uma pessoa, região e por afim a todo planeta. Sustentabilidade é a capacidade de sustentação ou conservação de um processo ou sistema. Sustentável deriva do latim *sustentare* e significa sustentar, apoiar, conservar e cuidar. Quando falamos de sustentabilidade encontramos o que é chamado de tripé da sustentabilidade, que seriam eles: Social, ambiental e Econômico.

Atualmente contamos com uma vasta tecnologia no ramo da sustentabilidade, onde temos benefícios destes dois caminhando juntos, os quais seriam: Inovação na produção; logística menos poluente; avaliação sobre poluição e outros indices; uso de dados para análise e tomada de decisão; auxílio aos consumidores, para que poluam menos, entre outras inúmeras possibilidades.

Em janeiro de 2020 a Corporate Knights empresa canadense especializada em desenvolvimento sustentável, publicou um artigo com as cem maiores empresas sustentáveis do mundo. Sendo algumas destas empresas: Orsted A/S (Dinamarca) – Setor de energia; Chr. Hansen Holding A/S (Dinamarca) – Setor de alimentos e agentes químicos; Cisco System Inc (Estados Unidos) – Setor de equipamentos de comunicação.

Deste modo o presente trabalho apresentará o conceito de chat sustentável ao leitor, visando criar um meio de conversa sobre questões sustentáveis, usando a linguagem *c#*, via TCP-IP, onde duas ou mais pessoas consigam se conectar simultaneamente e conversar.

2 REVISÃO DE LITERATURA

2.1 Redes

Denomina-se rede de informática o conjunto de computadores e equipamentos interligados que dividem informações, recursos e serviços. Atualmente, todo ser humano vive interligado a uma rede, seja ela de amigos, família, trabalho, ou uma rede tecnológica, através de uma conexão com a rede mundial de computadores, através de uma simples VPN, ou até mesmo pela ancoragem ao celular de um colega, no final todos estão conectados a uma rede.

Os benefícios de se ter uma grande rede de computadores onde se pode interligar pessoas do mundo todo, é justamente a grande troca de informações que ocorre hoje em dia, diferentemente de anos atrás, onde para se falar com alguém era necessário esperar dias, meses. A evolução foi tanta que hoje temos inúmeros tipos de redes, sendo elas: LAN, CAN, MAN, WAN, VLAN, WLAN, entre outros.

Cada uma destas tem sua finalidade veja:

- LAN (local area network) tem como objetivo ser uma rede local, ou seja, a curta distância, tendo capacidade para conectar dispositivos mais próximos em um mesmo ambiente.
- WAN (wide area network) tendo maior potência em relação a LAN e MAN, sendo possível conectar aparelhos em cômodos, cidades ou até países diferentes.
- PAN (personal area network) é a rede com maior limitação de alcance, conseguindo conectar aparelhos a curta distância, exemplo: Bluetooth

2.1.1 *Transmission control protocol (TCP)*

O TCP tem a funcionalidade de transmitir informações de uma para um IP específico em uma rede, transportando sua mensagem ao destino correto sem ter alguma falha ao seu envio.

O TCP/IP pode possuir 4 camadas para seu funcionamento adequado, entre elas são:

- Aplicação: Sua função é executar determinadas tarefas para poder fazer a comunicação.
- Transporte: Verificar se os dados ou informações chegou ao seu destino sem ter algum erro, podendo chegar em segurança ao outro usuário.

- Rede: Envio de mensagens do host garantindo que cheguem ao seu destino final.
- Interface de Rede: Seu objetivo é identificar se houve uma conexão do usuário na rede para poder ser capaz de enviar informações para outro usuário.

2.1.2 Sockets

Segundo MACORATTI (2003), “Um socket pode ser entendido como uma porta de um canal de comunicação que permite a um processo executando em um computador enviar/receber mensagens para/de outro processo que pode estar sendo executado no mesmo computador ou num computador remoto.”

Existem dois tipos de socket, sendo eles UDP e TCP. Os de tipo UDP são muito menos burocráticos e não garantem que todas as informações enviadas serão entregues de forma perfeita. Esse tipo de perda não se mostra de grande problema quando ocorrem em situações como chamadas áudio ou vídeos em geral, uma vez que apenas alguns frames ou informações perdidas não impactará a experiência final, sendo mais interessante o foco em fluidez.

Os sockets TCP, como já explicado, possuem a vantagem garantir que as informações enviadas sejam entregues sem nenhum tipo de falha. Essa segurança é necessária em diversos cenários, como transações, operações bancárias e trocas de mensagens.

Se localizando acima da camada de transporte e abaixo da aplicação, permite que essa comunicação seja entendida pelo software como uma comunicação direta, mesmo que esteja passando por uma rede.

Quando utilizada numa relação cliente/servidor, o servidor é responsável por criar um socket de comunicação e associa-lo a um IP e porta para que possa ser acessado por outro socket. Quando um cliente tenta se conectar, o socket irá aceitar a conexão, além de ouvir e escrever as informações passadas a ele.

Já no lado do cliente, é necessário apenas criar o socket e se conectar ao IP e porta existentes na rede, além de, assim como o servidor, sempre estará enviando e recebendo informações.

3 METODOLOGIA

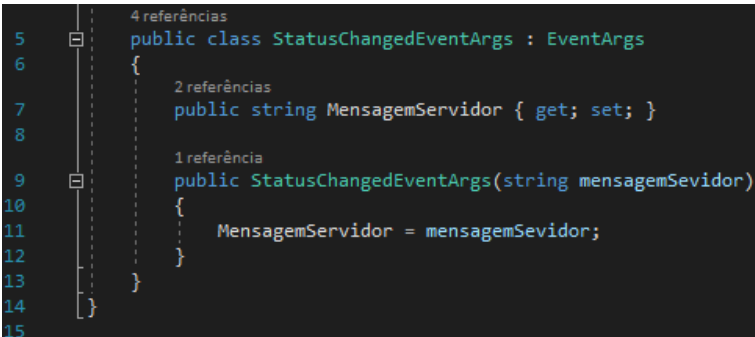
Realizado na Linguagem C#, este projeto de Chat consiste em dois programas distintos, que quando em conjunto, são responsáveis por efetivamente realizar a comunicações entre usuários em uma rede TCP/IP. Estes são: o programa Cliente e o Servidor.

3.1 Utilização de eventos

Em ambos os programas, será utilizado de eventos para que possam ser executados métodos distintos dinamicamente entre classes, podendo também ser transmitido informações entre elas. Isso ocorre através do atributo do tipo EventHandler, que estará presente na principal classe de cada programa, nele serão assinados métodos de outra classe para que, quando chamado, sejam executados.

3.1.1 StatusChangedEventArgs.cs

Herda em sua estrutura a classe EventArgs, será responsável por definir os argumentos a serem enviados aos métodos inscritos no tipo EventHandler.



```
5 public class StatusChangedEventArgs : EventArgs
6 {
7     2 referências
8     public string MensagemServidor { get; set; }
9
10    1 referência
11    public StatusChangedEventArgs(string mensagemSevidor)
12    {
13        MensagemServidor = mensagemSevidor;
14    }
15 }
```

Figura 1 - Classe StatusChangedEventArgs.cs linhas 5 à 15

3.2 Programa Cliente

Programa utilizado pelo próprio usuário para se comunicar com outros clientes conectados em uma mesma rede. Com funcionamento através de formulários (Windows Forms), ele terá uma interface visual agradável e intuitiva, voltada ao tema para que o usuário se sinta confortável com a aplicação.

3.2.1 Client.cs

Classe responsável pela maior parte do funcionamento do programa Cliente. Nela será tratada desde a tentativa inicial de conexão com o servidor, até o envio e recebimento de mensagens.

```

6 namespace Cliente
7 {
8     6 referências
9     public class Client
10    {
11        1 referência
12        public Client(string enderecoIP, int porta, string nomeUsuario)...
13
14        public static EventHandler<StatusChangedEventArgs> $statusChanged;
15
16        private string _enderecoIP;
17        private int _porta;
18        public static string NomeUsuario;
19        public string RespostaLogin;
20
21        private static TcpClient s_tcpConexao;
22        private static StreamWriter s_escritorConexao;
23        private static StreamReader s_leitorConexao;
24        private Thread _threadCliente;
25
26        1 referência
27        public void IniciarConexao()...
28
29        1 referência
30        private void EscutarMensagens()...
31
32        1 referência
33        public static void EnviarMensagem(string Mensagem)...
34
35        2 referências
36        private void OnStatusChanged(string mensagem)...
```

Figura 2 - Classe Client.cs completa

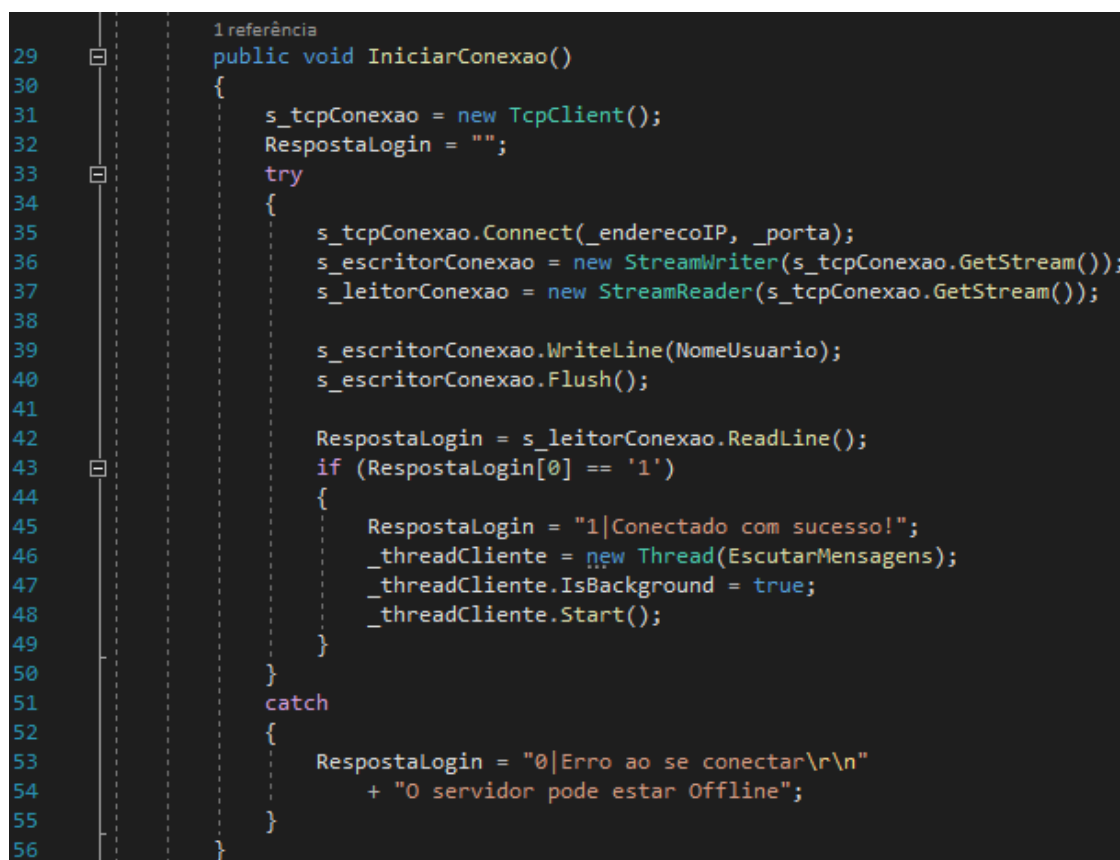
Em sua estrutura, haverá os seguintes atributos:

- StatusChanged – atributo do tipo EventHandler, responsável por armazenar métodos de outras classes para que, quando solicitado, execute todos em simultâneo de maneira dinâmica.
- _enderecoIP – string que armazenará o endereço do servidor em que será efetuada a tentativa de conexão.
- _porta – porta de conexão a qual o cliente irá se conectar.
- NomeUsuario – que contém o nome do usuário recebido pelo construtor.
- RespostaLogin – armazenará a resposta obtida pelo servidor após a tentativa de conexão.
- s_tcpConexao – conexão propriamente dita.
- s_escritorConexao – atributo responsável por escrever dados na conexão.
- s_leitorConexao – lerá qualquer novo dado que seja enviado na conexão.
- _threadCliente – thread que ficará ouvindo as mensagens do servidor em segundo plano.

3.2.1.1 Método IniciarConexao()

Nesse método será iniciada a tentativa de comunicação com o servidor. Inicialmente, são zerados os atributos `s_tcpConexao` e `RespostaLogin`, e é realizada uma tentativa de conexão com o IP e Porta solicitados, resultando em um erro de conexão em caso de quaisquer falhas.

Logo após o sucesso na conexão, tanto o escritor quanto o leitor de conexão são vinculados a conexão ativa, em seguida é enviado o nome do usuário ao servidor, onde este irá validá-lo e enviar uma resposta de aceitação ou recusa da conexão. Se o cliente for aceito, será iniciada uma Thread que escutará novas mensagens do servidor.



```
1 referência
29 public void IniciarConexao()
30 {
31     s_tcpConexao = new TcpClient();
32     RespostaLogin = "";
33     try
34     {
35         s_tcpConexao.Connect(_enderecoIP, _porta);
36         s_escritorConexao = new StreamWriter(s_tcpConexao.GetStream());
37         s_leitorConexao = new StreamReader(s_tcpConexao.GetStream());
38
39         s_escritorConexao.WriteLine(NomeUsuario);
40         s_escritorConexao.Flush();
41
42         RespostaLogin = s_leitorConexao.ReadLine();
43         if (RespostaLogin[0] == '1')
44         {
45             RespostaLogin = "1|Conectado com sucesso!";
46             _threadCliente = new Thread(EscutarMensagens);
47             _threadCliente.IsBackground = true;
48             _threadCliente.Start();
49         }
50     }
51     catch
52     {
53         RespostaLogin = "0|Erro ao se conectar\r\n"
54             + "0 servidor pode estar Offline";
55     }
56 }
```

Figura 3 - Classe Client.cs linhas 29 à 56

3.2.1.2 Método EscutarMensagens()

Este método, que será executado em Thread, consiste em um loop que irá disparar o evento OnStatusChanged a todos os métodos vinculados sempre que uma nova linha for escrita pelo servidor na conexão.

```
57 private void EscutarMensagens()
58 {
59     try
60     {
61         while (true)
62         {
63             string mensagem = "";
64             OnStatusChanged((mensagem = s_leitorConexao.ReadLine()) != null
65                 ? mensagem
66                 : throw new Exception());
67         }
68     }
69     catch
70     {
71         OnStatusChanged("0|Erro, conexão com o servidor perdida");
72     }
73 }
```

Figura 4 - Classe Client.cs linhas 57 a 73

3.2.1.3 Método EnviarMensagem()

Este método irá utilizar do s_escritorConexao para escrever a mensagem recebida como parâmetro no fluxo de dados da conexão com o servidor.

```
74 public static void EnviarMensagem(string Mensagem)
75 {
76     s_escritorConexao.WriteLine(Mensagem.Trim());
77     s_escritorConexao.Flush();
78 }
```

Figura 5 - Classe Client.cs linhas 74 a 78

3.2.2 Validacao.cs

Classe que irá validar os dados recebidos para iniciar uma conexão com o servidor escolhido.

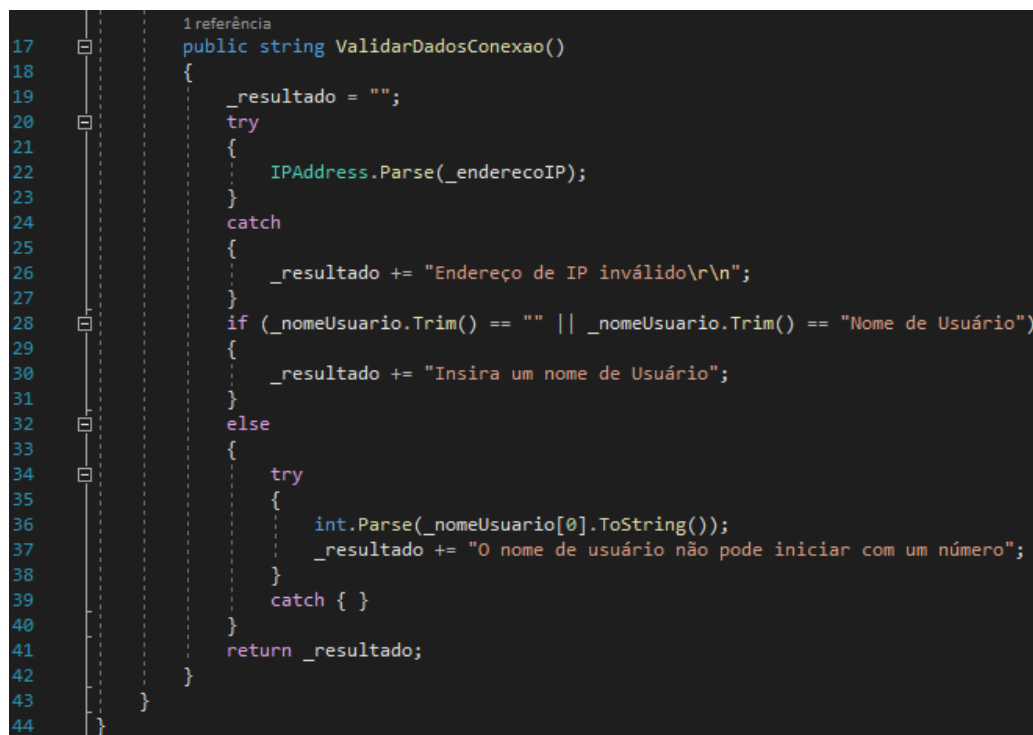
Ao ser instanciada, a classe irá atribuir o IP e o Nome recebidos pelo seu construtor aos seus atributos, em seguida ela irá esperar pelo comando para que inicie a validação dos dados obtidos, isso ocorre através do método ValidarDadosConexao().

3.2.2.1 Método ValidarDadosConexao()

Método que valida os dados obtidos para conexão, atribuindo ao atributo `_resultado` uma string com todos os erros de validação que possam ocorrer.

Ao início de sua validação, sempre será reiniciado o atributo `_resultado`, após isso o método tentará converter o IP recebido como string para o tipo `IPAddress`, de forma a verificar se o IP digitado é válido.

Em seguida será verificado em uma condicional se foi digitado algum nome. Caso sim, tentará ser convertido apenas o primeiro caractere dele para evitar que se inicie com um número, dessa forma evitando futuros erros pelo lado do servidor.



```

17 public string ValidarDadosConexao()
18 {
19     _resultado = "";
20     try
21     {
22         IPAddress.Parse(_enderecoIP);
23     }
24     catch
25     {
26         _resultado += "Endereço de IP inválido\r\n";
27     }
28     if (_nomeUsuario.Trim() == "" || _nomeUsuario.Trim() == "Nome de Usuário")
29     {
30         _resultado += "Insira um nome de Usuário";
31     }
32     else
33     {
34         try
35         {
36             int.Parse(_nomeUsuario[0].ToString());
37             _resultado += "O nome de usuário não pode iniciar com um número";
38         }
39         catch { }
40     }
41     return _resultado;
42 }
43
44

```

Figura 6 - Classe Validacao.cs linhas 17 a 44

3.2.3 FormLogin.cs

Primeira interface visual do programa, apresentada ao início de sua execução. Esta possui uma aparência simples e minimalista com apenas alguns campos para interação do usuário:

- `txbIP` – caixa de texto onde será inserido o endereço do servidor de gerenciamento de mensagens ao qual se conectará.
- `upDownPorta` – controlador do tipo `NumericUpDown`, responsável por definir a porta de conexão do IP escolhido.

- txbNomeUsuario – campo onde será digitado o nome do cliente que está a se conectar.
- btnConectar – botão responsável por iniciar o processo de conexão com o servidor.

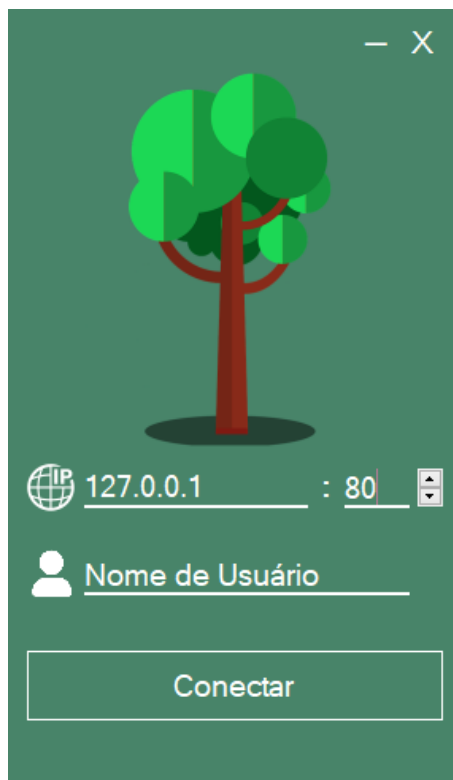
The image shows a login form window with a dark green background. At the top center is a stylized tree icon with a brown trunk and green foliage. Below the tree, there is a text input field for an IP address, containing '127.0.0.1', followed by a colon and a port number input field containing '80'. Below these fields is a text input field for the username, with a user icon on the left and the placeholder text 'Nome de Usuário'. At the bottom of the form is a large rectangular button labeled 'Conectar'. The window has a standard title bar with a minus sign and a close button (X) in the top right corner.

Figura 7 - Layout de formulário da classe FormLogin.cs

3.2.3.1 Método Conectar()

Método que será chamado como o clique do btnConectar ou com o pressionar da tecla “Enter” no campo de nome.

Ao ser chamado, irá instanciar a classe Validacao, enviando o IP e o usuário digitado, logo em seguida, será pedido a instancia de validação para validar os dados enviados. Caso a resposta não seja vazia, algum erro ocorreu, então será exibido uma caixa de diálogo exibindo os erros de validação.

Se não houverem quaisquer erros, será iniciado o processo de conexão. De início será instanciada a classe Client, enviando todos os dados do formulário como parâmetros do construtor. Em seguida será chamado o método IniciarConexao. Independente da resposta, seja positiva ou negativa, será mostrado em tela o resultado da tentativa de conexão e, se essa resposta recebida for positiva, será

instanciado e exibido o novo formulário que será responsável pela troca de mensagens com os outros usuários, o FormChat.

```

2 referências
14 private void Conectar()
15 {
16     Validacao validacao = new Validacao(txbIP.Text, txbNomeUsuario.Text);
17     string resposta = validacao.ValidarDadosConexao();
18     if (resposta != "")
19     {
20         MessageBox.Show(resposta);
21     }
22     else
23     {
24         Client cliente = new Client
25             (txbIP.Text, int.Parse(upDownPorta.Text), txbNomeUsuario.Text);
26         cliente.IniciarConexao();
27         MessageBox.Show(cliente.RespostaLogin.Substring(2));
28         if (cliente.RespostaLogin[0] == '1')
29         {
30             Hide();
31             FormChat chat = new FormChat();
32             chat.ShowDialog();
33             Close();
34         }
35     }
36 }

```

Figura 8 - Formulário FormLogin.cs linhas 14 a 36

3.2.4 FormChat.cs

Interface visual do Chat, onde ocorrerá o envio e recebimento das mensagens de forma clara ao usuário.

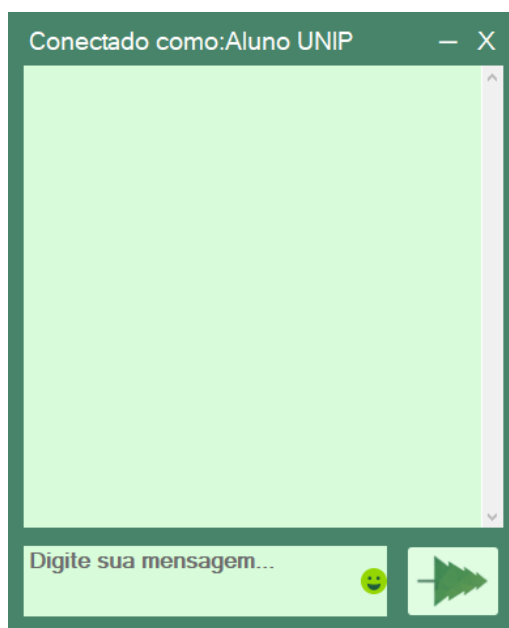


Figura 9 - Layout de formulário da classe FormChat.cs

Os seguintes elementos podem ser visualizados neste formulário:

- txbLog – caixa de texto que apresentará todo o Log de mensagens da conversa.

- txbmensagem – caixa de texto onde será digitada a mensagem do usuário para ser enviada.
- cmbEmoticons – comboBox que conterà os emoticons disponíveis para envio na conversa.
- btnEnviar – botão responsável por fazer o envio da mensagem do usuário.

3.2.4.1 Uso De Emoticons

Ao clicar no símbolo de emoticon verde do formulário, será disposto ao usuário diversas opções de emoticons para uso nas conversas, ao selecionar uma das opções, um evento será disparado para adicioná-lo à mensagem que está sendo escrita na caixa de texto.

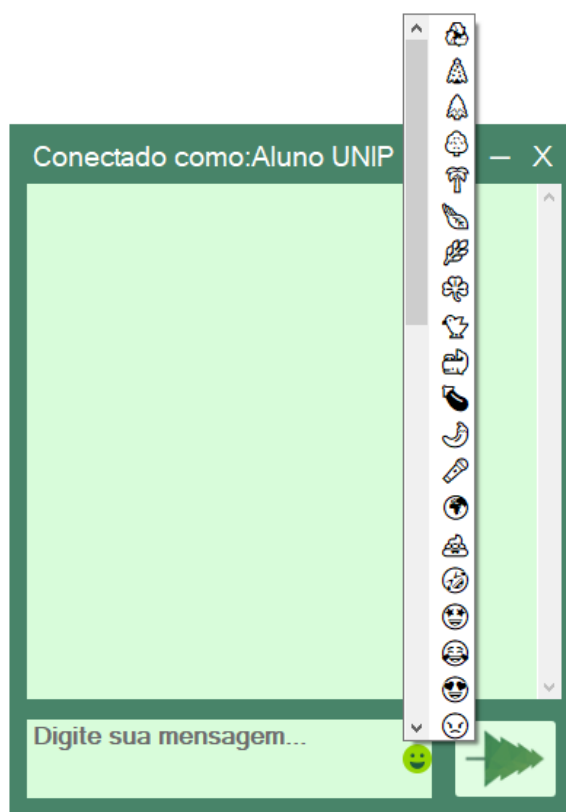


Figura 10 - Representação visual do menu de emojis

3.2.4.2 Método Enviar()

Método que irá solicitar o envio da mensagem contida na caixa de texto para a classe Client. Este método é chamado com o clique no botão conectar, e ao pressionar a tecla “Enter” enquanto digita uma mensagem.

```

2 referências
16 private void Enviar()
17 {
18     if (txbMensagem.Text.Trim() != "")
19     {
20         Client.EnviarMensagem(txbMensagem.Text);
21         txbMensagem.Text = "";
22     }
23 }

```

Figura 11 - Formulário FormChat.cs linhas 16 a 23

3.2.4.3 Método AtualizaLogMensagens()

Método que será executado sempre que for disparado um evento pela classe Cliente. É responsável por adicionar a mensagem recebida pelo servidor na classe Client a caixa de texto do formulário, caso esta não seja uma mensagem de erro. Se este for o caso, um pop-up aparecerá informando o erro.

```

51 private void AtualizaLogMensagens(string mensagem)
52 {
53     if (mensagem[0].ToString() == "0")
54     {
55         MessageBox.Show(mensagem.Substring(2));
56         Application.Exit();
57     }
58     else
59     {
60         txbLog.AppendText($"{mensagem}\r\n");
61     }
62 }

```

Figura 12 - Formulário FormChat.cs linhas 51 a 62

3.3 Programa Servidor

Programa responsável por criar uma instância de servidor em um determinado IP e porta, assim como receber as mensagens enviadas pelos usuários conectados e apresenta-las dinamicamente aos mesmos. Por se tratar de uma camada interna a qual os clientes não terão acesso, possui uma interface menos requintada e informações mais específicas e completas.

3.3.1 Server.cs

Classe responsável por fazer iniciar a instância do servidor e mantê-la rodando, desligar o servidor, validar e enviar mensagens encaminhadas ao servidor, além de criar o backup das conversas.

```

9  namespace Servidor
10 {
11     16 referências
12     class Server
13     {
14         1 referência
15         public Server(IPAddress enderecoIp, int porta) {...}
16
17         public static Hashtable Usuarios = new Hashtable(10);
18         public static EventHandler<StatusChangedEventArgs> StatusChanged;
19
20         private IPAddress _enderecoIp;
21         private int _porta;
22         private TcpClient _tcpServer = new TcpClient();
23         private Thread _threadListener;
24         private TcpListener _listenerServidor;
25         private bool _servidorRodando = false;
26
27         1 referência
28         public void IniciarServidor() {...}
29
30         1 referência
31         private void ManterServidor() {...}
32
33         1 referência
34         public static void ValidarMensagem(string usuario, string mensagem) {...}
35
36         1 referência
37         private static void EnviarMensagem(string usuario, string mensagem) {...}
38
39         3 referências
40         public static void EnviarMensagemAdmin(string mensagem) {...}
41
42         2 referências
43         private static void OnStatusChanged(string eventMessage) {...}
44
45         1 referência
46         public void FecharServidor() {...}
47
48         1 referência
49         public static void CriarBackup(string logMensagens) {...}
50     }
51 }

```

Figura 13 - Classe Server.cs completa

3.3.1.1 Construtor Server()

O construtor da classe é responsável apenas por receber o IP e a porta na qual o servidor será instanciado.

```

13     1 referência
14     public Server(IPAddress enderecoIp, int porta)
15     {
16         _enderecoIp = enderecoIp;
17         _porta = porta;
18     }

```

Figura 14 - Classe Server.cs linhas 13 a 17

3.3.1.2 Método IniciarServidor()

Método responsável por iniciar a instância do servidor baseada nas informações obtidas pelo construtor da classe. Além disso, inicia também uma Thread que garantirá que o método ManterServidor, que será explicado a seguir, seja executado até que seja solicitado seu encerramento.

```
1 referência
29 public void IniciarServidor()
30 {
31     _listenerServidor = new TcpListener(_enderecoIp, _porta);
32     _listenerServidor.Start();
33
34     _servidorRodando = true;
35
36     _threadListener = new Thread(ManterServidor);
37     _threadListener.IsBackground = true;
38     _threadListener.Start();
39 }
```

Figura 15 - Classe Server.cs linhas 29 a 39

3.3.1.3 Método ManterServidor()

Este método possui um loop que garantirá que qualquer requisição de conexão ao servidor seja aceita enquanto o mesmo estiver rodando.

```
1 referência
40 private void ManterServidor()
41 {
42     while (_servidorRodando)
43     {
44         try
45         {
46             _tcpServer = _listenerServidor.AcceptTcpClient();
47             ConexaoUsuario novaConexao = new ConexaoUsuario(_tcpServer);
48         }
49         catch { }
50     }
51 }
```

Figura 16 - Classe Server.cs linhas 40 a 51

3.3.1.4 Método ValidarMensagem()

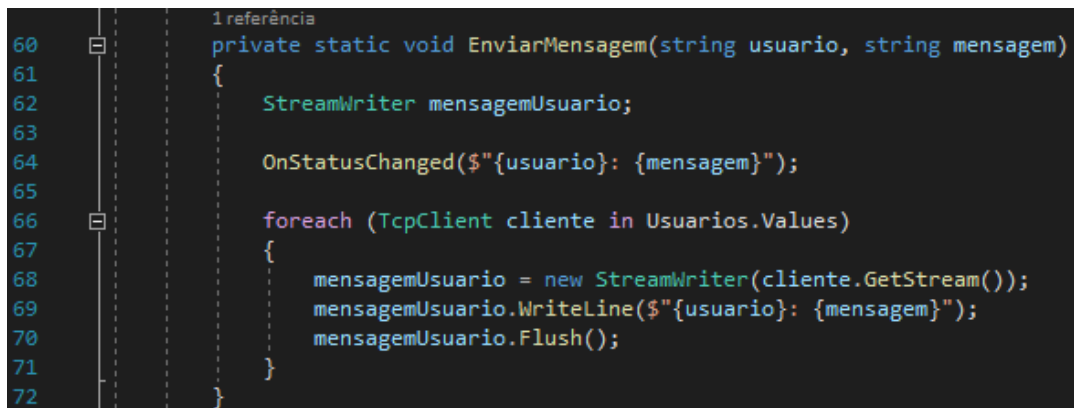
Recebe como parâmetros o nome do usuário e a mensagem a ser enviada. Tem como objetivo apenas garantir que a mensagem enviada não seja vazia.

```
1 referência
52 public static void ValidarMensagem(string usuario, string mensagem)
53 {
54     string tempMensagem = mensagem.Trim();
55     if (tempMensagem != "")
56     {
57         EnviarMensagem(usuario, tempMensagem);
58     }
59 }
```

Figura 17 - Classe Server.cs linhas 52 a 59

3.3.1.5 Método EnviarMensagem()

Recebe como parâmetros o nome do usuário e a mensagem já tratada. Transmite a mensagem formatada para o método OnStatusChanged para que seja transmitida a todos os outros eventos inscritos a ele. Em seguida, transmite a mensagem para cada usuário conectado ao servidor.

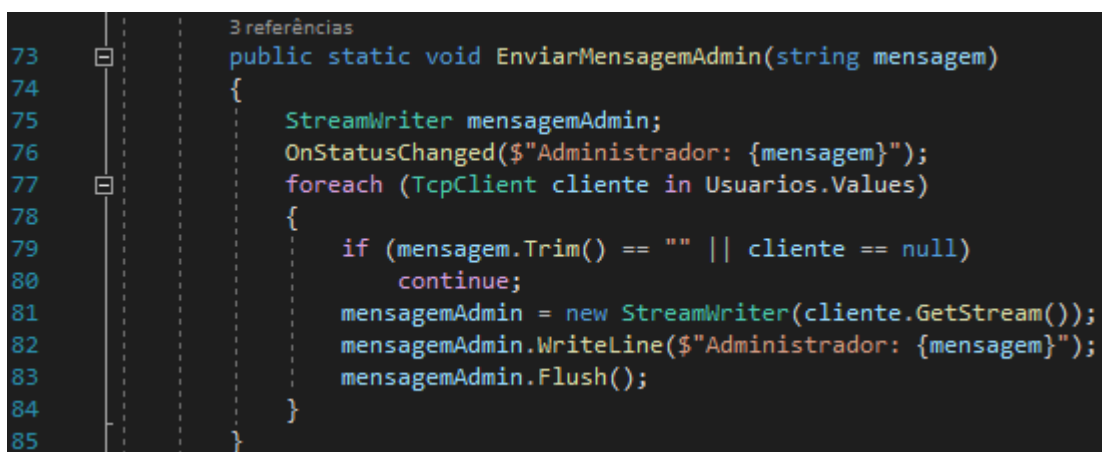


```
1 referência
60 private static void EnviarMensagem(string usuario, string mensagem)
61 {
62     StreamWriter mensagemUsuario;
63
64     OnStatusChanged($"{usuario}: {mensagem}");
65
66     foreach (TcpClient cliente in Usuarios.Values)
67     {
68         mensagemUsuario = new StreamWriter(cliente.GetStream());
69         mensagemUsuario.WriteLine($"{usuario}: {mensagem}");
70         mensagemUsuario.Flush();
71     }
72 }
```

Figura 18 - Classe Server.cs linhas 60 a 72

3.3.1.6 Método EnviarMensagemAdmin()

Recebe como parâmetro a mensagem que será enviada pelo servidor. Da mesma forma que o método anterior, transmite a mensagem ao método OnStatusChanged e, em seguida, aos usuários conectados no servidor.

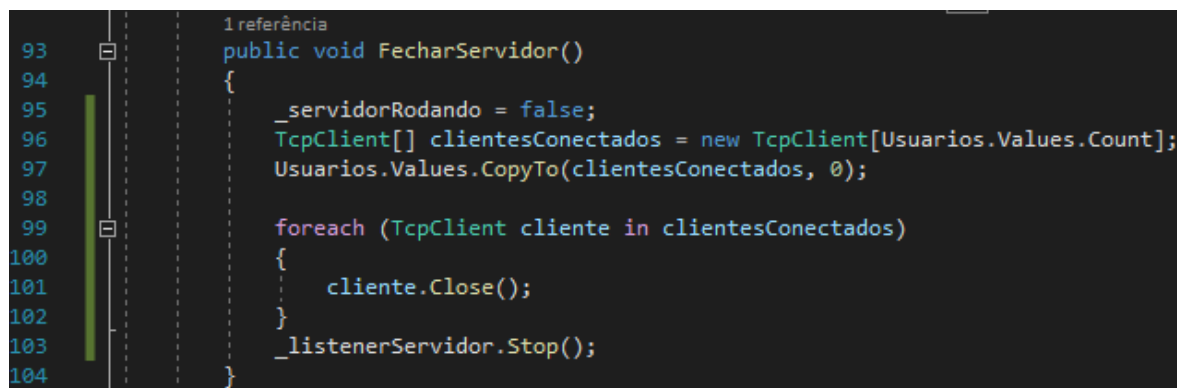


```
3 referências
73 public static void EnviarMensagemAdmin(string mensagem)
74 {
75     StreamWriter mensagemAdmin;
76     OnStatusChanged($"Administrador: {mensagem}");
77     foreach (TcpClient cliente in Usuarios.Values)
78     {
79         if (mensagem.Trim() == "" || cliente == null)
80             continue;
81         mensagemAdmin = new StreamWriter(cliente.GetStream());
82         mensagemAdmin.WriteLine($"Administrador: {mensagem}");
83         mensagemAdmin.Flush();
84     }
85 }
```

Figura 19 - Classe Server.cs linhas 73 a 85

3.3.1.7 Método FecharServidor()

Definirá o estado do servidor como parado e, em seguida, encerrará a conexão de todos os usuários conectados. Após concluída a desconexão dos usuários, o funcionamento do servidor será interrompido.

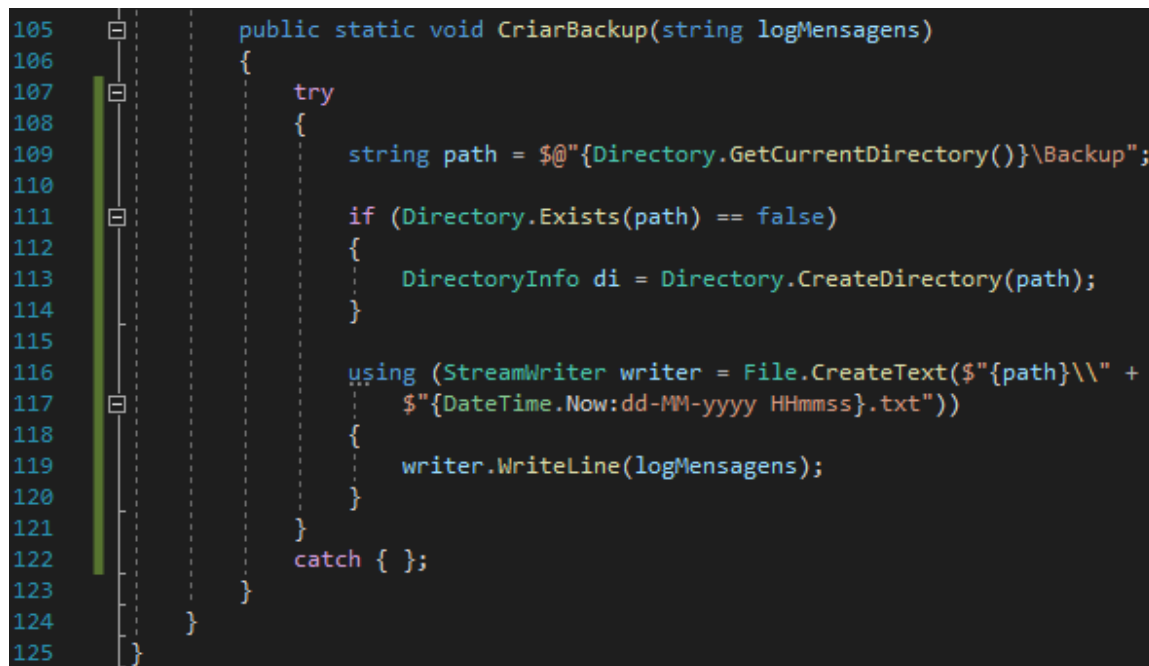
A screenshot of a code editor showing the implementation of the FecharServidor() method in the Server class. The code is in C# and spans lines 93 to 105. It sets a private field _servidorRodando to false, creates an array of TcpClient objects from the connected users, and then iterates through this array to close each client connection. Finally, it stops the server listener.

```
1referência
93 public void FecharServidor()
94 {
95     _servidorRodando = false;
96     TcpClient[] clientesConectados = new TcpClient[Usuarios.Values.Count];
97     Usuarios.Values.CopyTo(clientesConectados, 0);
98
99     foreach (TcpClient cliente in clientesConectados)
100     {
101         cliente.Close();
102     }
103     _listenerServidor.Stop();
104 }
```

Figura 20 - Classe Server.cs linhas 93 a 105

3.3.1.8 Método CriarBackup()

Inicialmente, verifica se na pasta do programa existe o diretório “Backup”, criando-o caso não exista. Em seguida, baseando-se num formato “Dia-Mês-Ano – HoraMinutoSegundo” é criado um novo arquivo no formato de texto dentro da pasta Backup.

A screenshot of a code editor showing the implementation of the CriarBackup() method in the Server class. The code is in C# and spans lines 106 to 125. It uses a try-catch block to attempt creating a 'Backup' directory. If it fails, it creates the directory. Then, it creates a text file with a timestamp in the format 'dd-MM-yyyy HHmmss.txt' and writes the log messages to it.

```
105 public static void CriarBackup(string logMensagens)
106 {
107     try
108     {
109         string path = $"{Directory.GetCurrentDirectory()}\\Backup";
110
111         if (Directory.Exists(path) == false)
112         {
113             DirectoryInfo di = Directory.CreateDirectory(path);
114         }
115
116         using (StreamWriter writer = File.CreateText($"{path}\\\" +
117             $"{DateTime.Now:dd-MM-yyyy HHmmss}.txt"))
118         {
119             writer.WriteLine(logMensagens);
120         }
121     }
122     catch { };
123 }
124
125 }
```

Figura 21 - Classe Server.cs linhas 106 a 122

3.3.2 ConexãoUsuario.cs

Classe responsável por validar a conexão do usuário com o servidor e, a partir disso, incluí-lo ou não na instância, assim como removerá o usuário caso o mesmo se desconecte por conta própria. Além disso, fica responsável por receber a apresentar sempre que uma mensagem é enviada ao servidor e por fechar a conexão quando solicitado.

```

6  namespace Servidor
7  {
8      3 referências
9      class ConexaoUsuario
10     {
11         1 referência
12         public ConexaoUsuario(TcpClient cliente) {...}
13
14         private TcpClient _tcpClient;
15         private Thread _threadValidacao;
16         private StreamReader _leitorConexao;
17         private StreamWriter _escritorConexao;
18         private string _usuarioAtual;
19
20         1 referência
21         private void ValidarUsuario() {...}
22
23         1 referência
24         private void AceitarUsuario(TcpClient cliente, string usuarioAtual) {...}
25
26         1 referência
27         private void AguardarMensagem() {...}
28
29         1 referência
30         private void RemoverUsuario(string usuarioAtual) {...}
31
32         4 referências
33         private void FechaConexao() {...}
34
35     }
36 }

```

Figura 22 - Classe ConexaoUsuario.cs completa

3.3.2.1 Construtor ConexaoUsuario()

Recebe como parâmetro o usuário que tentar se conectar ao servidor e inicia uma Thread para validá-lo. Essa Thread é necessária pois, caso múltiplos usuários tentassem se conectar ao mesmo tempo no servidor, os mesmos seriam validados um a um, o que criaria um gargalo na execução do programa.

```

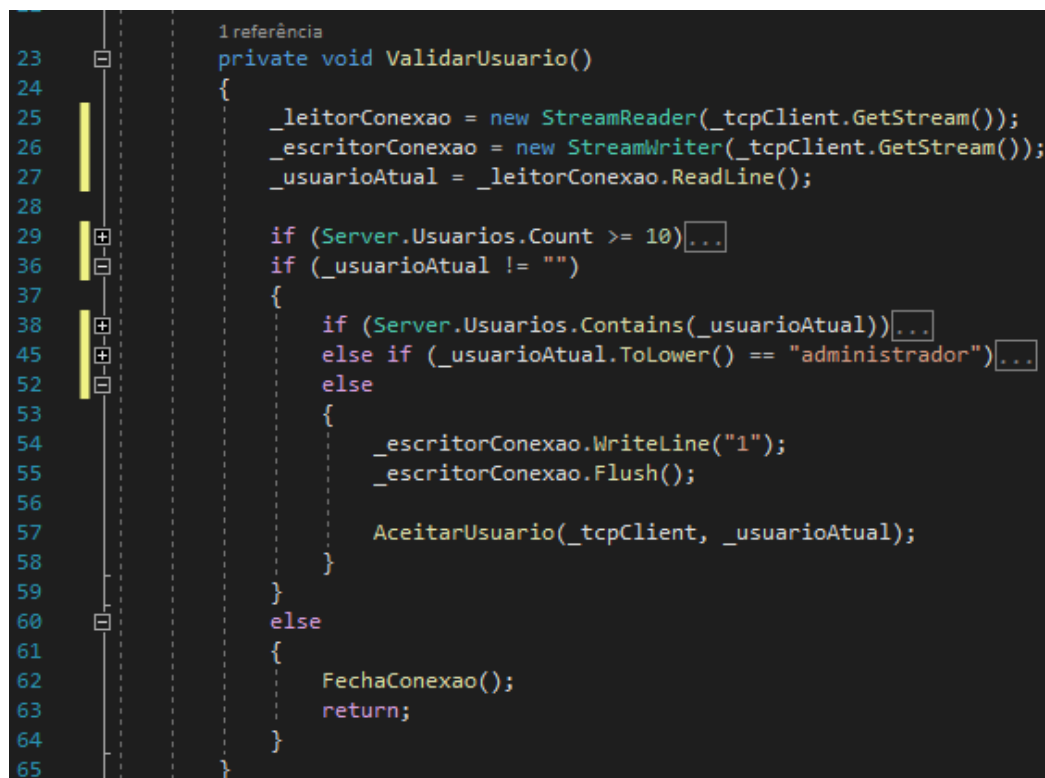
10     1 referência
11     public ConexaoUsuario(TcpClient cliente)
12     {
13         _tcpClient = cliente;
14         _threadValidacao = new Thread(ValidarUsuario);
15         _threadValidacao.Start();
16     }

```

Figura 23 - Classe ConexaoUsuario.cs linhas 10 a 15

3.3.2.2 Método ValidarUsuario()

Este método é responsável por validar se o usuário pode ou não se conectar ao servidor. Os testes de validação incluídos são: verificação do número máximo de conexões permitidos por instância do servidor, nomes de usuário vazios ou contendo apenas espaços e nome igual a “administrador”. Caso as validações corram bem, as informações são então enviadas ao método AceitarUsuario descrito a seguir.

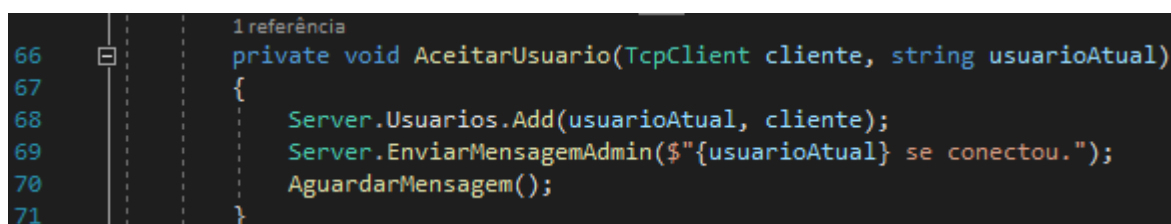


```
1 referência
23 private void ValidarUsuario()
24 {
25     _leitorConexao = new StreamReader(_tcpClient.GetStream());
26     _escritorConexao = new StreamWriter(_tcpClient.GetStream());
27     _usuarioAtual = _leitorConexao.ReadLine();
28
29     if (Server.Usuarios.Count >= 10) ...
30     if (_usuarioAtual != "")
31     {
32         if (Server.Usuarios.Contains(_usuarioAtual)) ...
33         else if (_usuarioAtual.ToLower() == "administrador") ...
34         else
35         {
36             _escritorConexao.WriteLine("1");
37             _escritorConexao.Flush();
38
39             AceitarUsuario(_tcpClient, _usuarioAtual);
40         }
41     }
42     else
43     {
44         FechaConexao();
45         return;
46     }
47 }
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
```

Figura 24 - Classe ConexaoUsuario.cs linhas 23 a 65

3.3.2.3 Método AceitarUsuario()

Recebe como parâmetros o nome do usuário já validado anteriormente e sua conexão. A partir disso, é adicionado à lista de clientes conectados ao servidor e é disparado uma mensagem informando a nova conexão. Ao fim, o método AguardarMensagem é chamado.



```
1 referência
66 private void AceitarUsuario(TcpClient cliente, string usuarioAtual)
67 {
68     Server.Usuarios.Add(usuarioAtual, cliente);
69     Server.EnviaMensagemAdmin($"{usuarioAtual} se conectou.");
70     AguardarMensagem();
71 }
```

Figura 25 - Classe ConexaoUsuario.cs linhas 66 a 71

3.3.2.4 Método AguardarMensagem()

Este método possui um laço infinito que irá receber todas as mensagens enviadas pelo usuário e encaminha-las a validação. Caso a mensagem a ser encaminhada seja nula, significa que o usuário se desconectou, então uma exceção será lançada e o método RemoverUsuario será chamado.

```

72 1 referência
73 private void AguardarMensagem()
74 {
75     try
76     {
77         while (true)
78         {
79             Server.ValidarMensagem(_usuarioAtual, _leitorConexao.ReadLine());
80         }
81     }
82     catch (Exception)
83     {
84         RemoverUsuario(_usuarioAtual);
85     }

```

Figura 26 - Classe ConexaoUsuario.cs linhas 72 a 85

3.3.2.5 Método RemoverUsuario()

Recebe como parâmetro o nome do usuário que será removido e o remove da lista de conexões. Em seguida, envia através do servidor uma mensagem com o nome e horário em que o usuário se desconectou.

```

86 private void RemoverUsuario(string usuarioAtual)
87 {
88     if (Server.Usuarios[usuarioAtual] != null)
89     {
90         Server.Usuarios.Remove(usuarioAtual);
91         Server.EnviarMensagemAdmin($"{usuarioAtual} se desconectou. (" +
92             $"{DateTime.Now:HH:mm}")");
93     }
94 }

```

Figura 27 - Classe ConexaoUsuario.cs linhas 86 a 94

3.3.2.6 Método FechaConexao()

Responsável por encerrar todas as conexões já estabelecidas com o servidor.

```

95 4 referências
96 private void FechaConexao()
97 {
98     _tcpClient.Close();
99     _leitorConexao.Close();
100    _escritorConexao.Close();

```

Figura 28 - Classe ConexaoUsuario.cs linhas 95 a 100

3.3.3 FormServidor.cs

Formulário do próprio servidor, responsável por criar e visualizar o chat em tempo real. É composta pelos seguintes componentes:

- txblp – TextBox que receberá o IP do servidor que será instanciado.
- upDownPort – Campo que receberá a porta do servidor.
- btnCriarServidor – Botão que iniciará a instância do servidor no IP e porta definidos.
- txbLog – Campo de texto responsável por apresentar todas as mensagens enviadas ao servidor.
- txbMensagem – Campo de texto que receberá a mensagem que o administrador deseja enviar ao servidor e, conseqüentemente, a todos os usuários conectados.
- btnEnviarMensagem – Responsável por enviar as mensagens do administrador ao servidor.

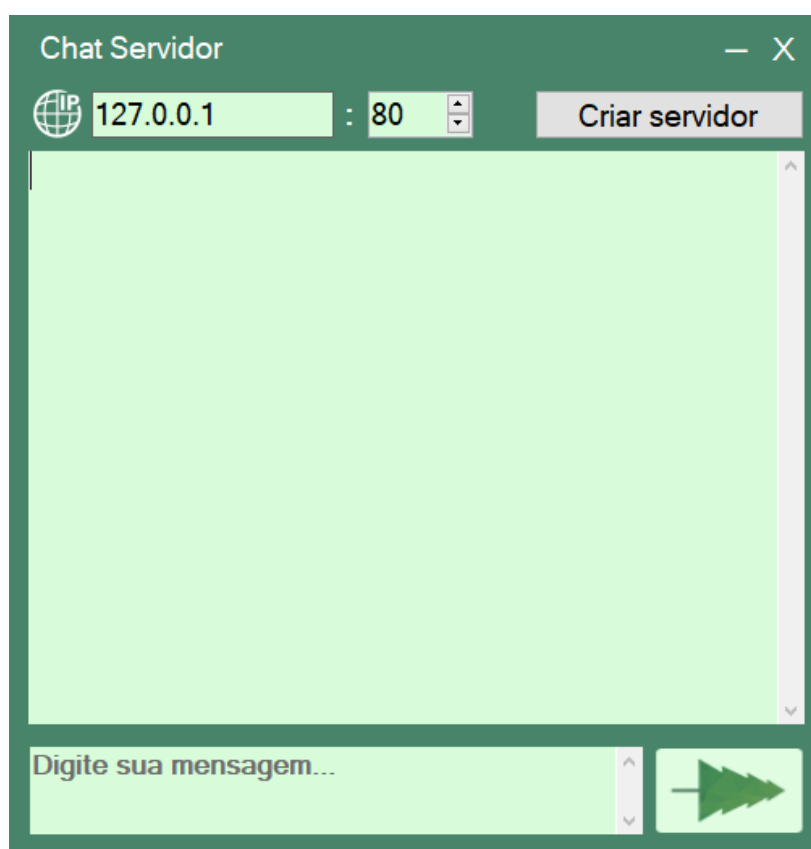


Figura 29 - Layout do formulário da classe FormServer.cs

3.3.3.1 Atributos da Classe

Em sua estrutura possuímos três atributos, inicialmente é apresentado o método delegate para que seja possível a chamada de um método assinado em tempo de execução, há também uma booleana que indica se o servidor está iniciado e uma instância servidor da classe Server.cs.

```
16 | private delegate void AtualizaLogCallBack(string mensagem);
17 | private bool _servidorRodando = false;
18 | private Server _servidor;
```

Figura 30 - Formulário FormServer.cs linhas 16 a 18

3.3.3.2 Evento Click btnConectar()

Ao ser pressionado o botão de criação do servidor será feito um teste para verificar se o servidor está rodando, se não, ele tenta iniciar uma conexão TCP através da classe Server.cs com o IP e Porta digitados. Em seguida o método AtualizaStatus escuta ao EventHandler do servidor, iniciando-o logo após isso.

```
20 | 1referência
21 | private void btnCriarServidor_Click(object sender, EventArgs e)
22 | {
23 |     if (_servidorRodando == false)
24 |     {
25 |         if (txbIp.Text == "")...
26 |         try
27 |         {
28 |             IPAddress enderecoIp = IPAddress.Parse(txbIp.Text);
29 |             int porta = (int)upDownPort.Value;
30 |
31 |             _servidor = new Server(enderecoIp, porta);
32 |             Server.StatusChanged += OnServidorStatusChanged;
33 |             _servidor.IniciarServidor();
34 |
35 |             txbLog.Text = "";
36 |             txbLog.AppendText("Esperando conexões...\r\n");
37 |
38 |             _servidorRodando = true;
39 |             btnCriarServidor.Text = "Fechar servidor";
40 |             txbMensagem.Enabled = true;
41 |
42 |         }
43 |         catch (Exception ex)
44 |         {
45 |             MessageBox.Show(ex.ToString());
46 |         }
47 |     }
48 |     else
49 |     {
50 |         Server.StatusChanged -= OnServidorStatusChanged;
51 |         LogFecharServidor();
52 |         _servidor.FecharServidor();
53 |         btnCriarServidor.Text = "Criar servidor";
54 |         txbMensagem.Enabled = false;
55 |         CriarBackup();
56 |         _servidorRodando = false;
57 |     }
58 | }
59 |
60 |
61 | }
```

Figura 31 - Formulário FormServer.cs linhas 20 a 61

4 RESULTADOS

Com o fim do desenvolvimento da aplicação, foram testados tanto seu funcionamento quanto suas funcionalidades extras, testes estes que correram com sucesso demonstrando a eficácia do programa.

4.1 Funcionamento do chat

De maneira simples, inicialmente deve ser iniciado a aplicação de servidor e iniciado os serviços de recepção e gerenciamento de mensagens através do clique no botão “Criar Servidor”.

Com o servidor iniciado, os clientes já podem se conectar com suas aplicações, necessitando apenas colocar IP e porta que corresponda ao servidor e um nome de usuário válido, ocorrendo, a partir deste momento, a possibilidade de troca de mensagens com todos conectados.

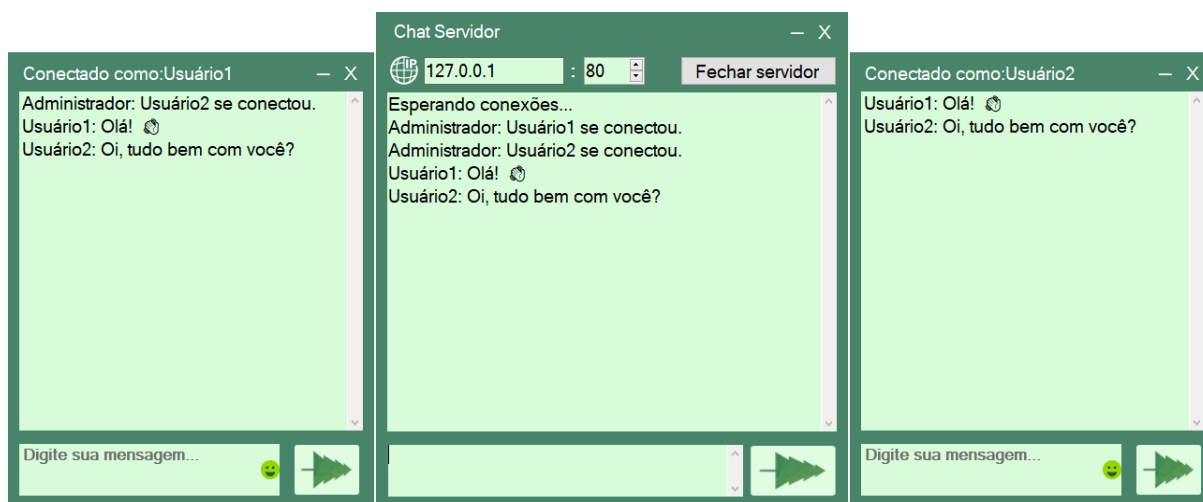


Figura 32 - Funcionamento do Chat

4.2 Tratamento de erros

Por trabalhar com dois programas em paralelo interligados através de uma rede, houve uma atenção especial ao tratamento de erros para que fosse garantido o correto funcionamento de ambos os programas.

4.2.1 Cliente

Com o início de sua execução é apresentado o formulário de Login com o servidor onde deve ser inserido IP, Porta e Usuário. Neste ponto, diversas situações

podem ocorrer, sendo assim, foram tratados diferentes cenários, os seguintes erros podem ocorrer independente da execução do servidor ao tentar se conectar:

- Erro de IP: Caso seja digitado um endereço IP que não obedece ao padrão, será exibido o erro em tela notificando o cliente.
- Erro de Nome: São tratados os casos em que não é digitado um nome, assim como se for digitado um que se inicie com número, evitando assim futuros erros no programa.
- Erro de Conexão: Apesar de ser uma falha relacionada ao servidor, ela ocorre independente da execução dele e deve ser tratada. Ao não conseguir comunicar o Soquete com o IP escolhido o cliente será avisado de que houve uma falha de comunicação e que o servidor pode estar Offline.

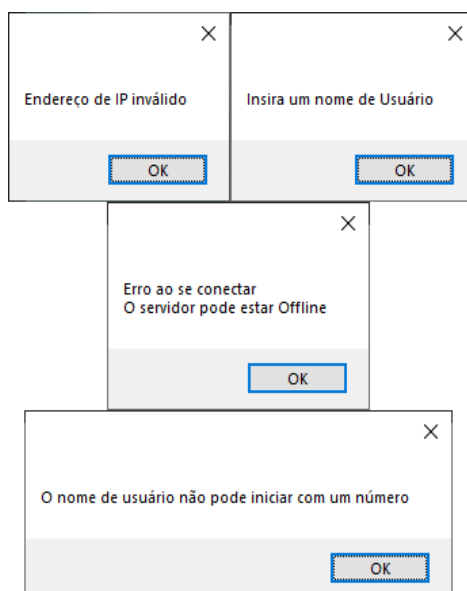


Figura 33 - Erros tratados no formulário FormLogin.cs

Apesar dos erros que ocorrem independente do servidor, o cliente está sujeito a receber alguns erros após a tentativa de conexão apesar dos campos estarem corretos, erros estes que são recebidos pelo servidor após se conectar e também são mostrados na tela através de um pop-up, eles incluem:

- Limite de usuários atingido: ocorre quando o servidor já atingiu o limite e usuários conectados.
- Nome já existente: é exibido quando já existe no servidor algum cliente conectado com o mesmo nome de usuário.

- Nome reservado: ocorre quando o cliente tenta utilizar do nome de “administrador”.

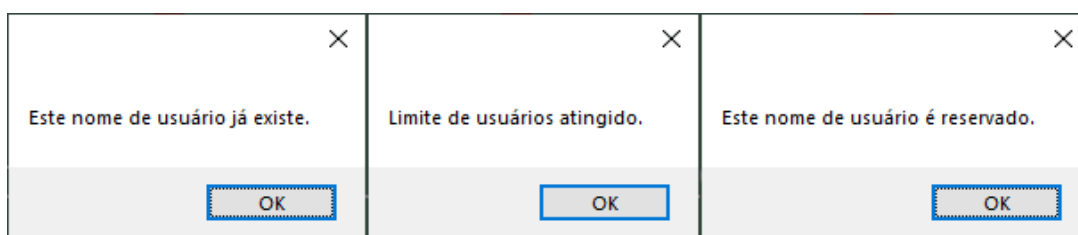


Figura 34 - Erros tratados no FormLogin.cs recebidos do servidor

Com o sucesso na conexão do cliente é exibido o formulário principal do Chat, onde alguns problemas também são tratados, sendo estes a tentativa de envio de uma mensagem vazia, ou o encerramento inesperado da conexão por parte do servidor, o que resulta em um pop-up de erro que notifica ao cliente sobre essa perda de conexão encerrando a aplicação.

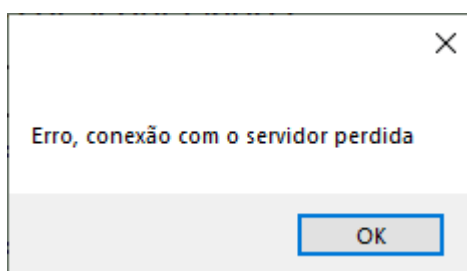


Figura 35 - Pop-up do Cliente ao perder a conexão com o server

4.2.2 Servidor

Por se tratar geralmente do lado do proprietário da rede ou do chat, os erros que podem ocorrer no servidor são exibidos de forma mais direta, com uma caixa de pop-up sendo exibida com a exceção que ocorreu. O exemplo a seguir se trata da digitação de um IP inválido para abertura do servidor.

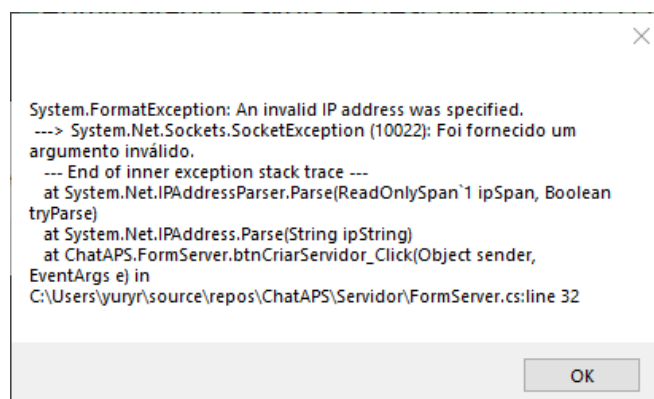


Figura 36 - Pop-up de erro do programa Servidor

4.3 Funcionalidades Extras

Além da troca de mensagens de texto comum presente no programa, foram implementados também recursos extras em ambas as aplicações.

Para o servidor, foi adicionado a possibilidade de interação com os integrantes do chat, permitindo que esse também consiga enviar mensagens a todos os clientes conectados. Além disso, o servidor também conta com um relatório que exibe horário e clientes desconectados ao encerrar o serviço de troca de mensagem. Outro recurso de extrema utilidade ao servidor é que, após gerar o relatório, o programa cria um diretório que armazena um backup com todo o log de mensagens de cada conexão, salvo em um arquivo de texto com o nome registrando data e hora do fechamento do servidor.

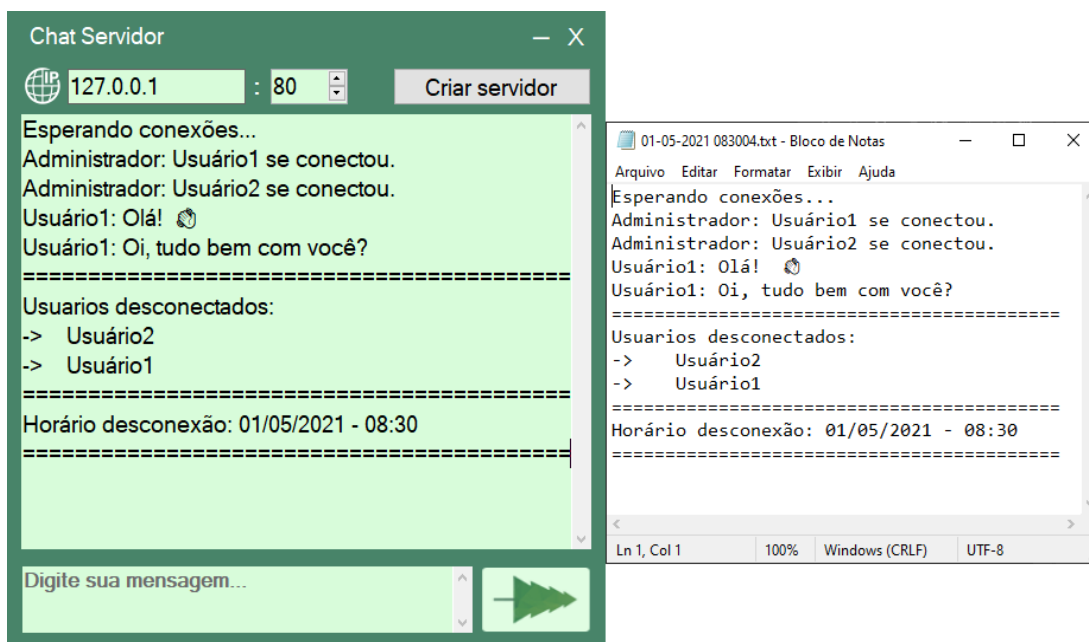


Figura 37 - Demonstração do relatório de conexão e arquivo de Backup criado

No programa Cliente, foi implementado o uso de Emojis, de forma a tornar a comunicação entre usuários conectados mais prática, descontraída e com maior variedade.



Figura 38 - Emojis disponíveis para uso do usuário conectado

5 CONCLUSÃO

O objetivo deste trabalho foi, por meio da linguagem de programação C#, criar aplicações para servidor e cliente de forma a permitir que pessoas de uma rede interna se conectem para trocar informações ou dados importantes numa situação que envolva o Meio Ambiente.

Na aplicação foram utilizados dos conceitos de soquetes para estabelecer a conexão entre Servidor e Cliente, também houve o uso de Threads e de eventos para tratar dinamicamente do formulário durante a execução dos programas. Além disso, foi colocado um grande esforço em sua interface de forma a fugir da tradicional aparência de Forms das linguagens de programação, trazendo um design minimalista com cores suaves, agradáveis ao olhar.

Em seu desenvolvimento, houveram diversas dificuldades quanto execução de métodos de classes diferentes em tempo real. Após muita pesquisa, foi possível entender o funcionamento do tratamento de eventos e de funções delegadas, permitindo a aplicação assinar métodos de outras classes e executa-los dinamicamente através de uma única chamada. Após solucionar este problema, foram implementadas as funcionalidades fundamentais ao chat e adicionados alguns recursos próprios como o envio de emojis e o backup de todo o registro de mensagens.

Os estudos realizados acerca do tema proposto permitiram que o grupo se desenvolvesse de forma a criar uma aplicação que não só satisfizesse os requisitos do projeto, mas que também motivasse a equipe a ir além, adicionando novos recursos e funcionalidades para enriquecer a solução final.

REFERÊNCIAS

CONCEITO. CONCEITO.DE. **Conceito de Rede**, 2012. Disponível em: <<https://conceito.de/rede>>. Acesso em: Março 2021.

FIGUEIREDO, I. L. Oficina da Net. **História das redes de computadores**, 2014. Disponível em: <<https://www.oficinadanet.com.br/post/10123-historia-das-redes-de-computadores>>. Acesso em: Março 2021.

GAIDARGI, J. Infonova. **O que é TCP/IP e como funciona**, 2018. Disponível em: <<https://www.infonova.com.br/artigo/o-que-e-tcp-ip-e-como-funciona/>>. Acesso em: abril 2021.

MACORATTI, J. C. Macoratti..**NET - Usando a comunicação Cliente - Servidor com sockets**, 2003. Disponível em: <http://www.macoratti.net/net_soc1.htm>. Acesso em: Maio 2021.

MACORATTI, J. C. Macoratti. **C # - Criando um Chat - Parte 1 - O Cliente**, 2011. Disponível em: <http://www.macoratti.net/11/08/c_chat1.htm>. Acesso em: Maio 2021.

MACORATTI, J. C. Macoratti. **C # - Criando um Chat - Parte 2 - O Servidor**, 2011. Disponível em: <http://www.macoratti.net/11/08/c_chat2.htm>. Acesso em: Maio 2021.

PANTUZA, G. Blog Pantuza. **O QUE SÃO E COMO FUNCIONAM OS SOCKETS**, 2017. Disponível em: <<https://blog.pantuza.com/artigos/o-que-sao-e-como-funcionam-os-sockets>>. Acesso em: Abril 2021.


REDES, E. S. D. Escola Superior de Redes. **Arquitetura TCP/IP: conceitos básicos**, 2020. Disponível em: <<https://esr.rnp.br/administracao-e-projeto-de-redes/arquitetura-tcp/ip-conceitos-basicos/>>. Acesso em: Abril 2021.

SUPPORT, N. NetSupport. **Redes de computadores: o que são e quais os principais tipos?**, 2018. Disponível em: <<https://netsupport.com.br/blog/redes-de-computadores/>>. Acesso em: Abril Redes de computadores: o que são e quais os principais tipos?

TANENBAUM, A. S. **Redes de computadores**. 5ª. ed. [S.l.]: Pearson Universidades, 2011.

TEDESCO, K. TreinaWeb. **Uma introdução a TCP, UDP e Sockets**, 2020. Disponível em: <<https://www.treinaweb.com.br/blog/uma-introducao-a-tcp-udp-e-sockets>>. Acesso em: Abril 2021.

ANEXOS



UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Alysson Moreira Costa TURMA: CC5P17 RA: N4352J-9

CURSO: Ciência da Computação CAMPUS: Sorocaba SEMESTRE: 5ºSemestre TURNO: Noite

CÓDIGO DA ATIVIDADE: 77B2 SEMESTRE: 5ºSemestre ANO GRADE: 2021/1

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
13/03/2021	Definição do grupo	1H	Alysson Moreira Costa	1H	
16/03/2021	Discussão em grupo sobre a linguagem utilizada	2H	Alysson Moreira Costa	2H	
20/03/2021	Pesquisas sobre sockets	4H	Alysson Moreira Costa	4H	
22/03/2021	Pesquisa sobre o uso de sockets em C#	5H	Alysson Moreira Costa	5H	
25/03/2021	Realizado os primeiros testes de conexão por sockets em linguagem C#	6H	Alysson Moreira Costa	6H	
03/04/2021	Obtido sucesso a implementação de sockets em C#	5H	Alysson Moreira Costa	5H	
10/04/2021	Início do desenvolvimento do projeto	6H	Alysson Moreira Costa	6H	
15/04/2021	Estudos sobre execução dinâmica de métodos entre classes	5H	Alysson Moreira Costa	5H	
20/04/2021	Iniciado estudo sobre transmissão de informações com eventos	4H	Alysson Moreira Costa	4H	
24/04/2021	Realizado estudo sobre delegates	5H	Alysson Moreira Costa	5H	
02/05/2021	Finalização da primeira versão funcional do programa	7H	Alysson Moreira Costa	7H	
20/05/2021	Início do desenvolvimento da documentação	3H	Alysson Moreira Costa	3H	
20/05/2021	Limpeza de código e correção de bugs	6H	Alysson Moreira Costa	6H	
21/05/2021	Limpeza de código	3H	Alysson Moreira Costa	3H	
20/05/2021	Desenvolvimento do FrmChat e ConexãoUsuario na documentação	4H	Alysson Moreira Costa	4H	
21/05/2021	Desenvolvimento da conclusão na documentação	3H	Alysson Moreira Costa	3H	
22/05/2021	Desenvolvimento do TCP/IP na documentação	3H	Alysson Moreira Costa	3H	
22/05/2021	Correção finais e conclusão do projeto	2H	Alysson Moreira Costa	2H	
22/05/2021	Finalização da documentação	6H	Alysson Moreira Costa	6H	

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____
 AVALIAÇÃO: _____
 Aprovado ou Reprovado
 NOTA: _____
 DATA: ____/____/_____

 CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Augusto Benjamyn Leal Martins

TURMA: CC5P17

RA: N406BG0

CURSO: Ciência da computação

CAMPUS: Sorocaba

SEMESTRE: 5º

TURNO: Noite

CÓDIGO DA ATIVIDADE: 77B2

SEMESTRE: 5º

ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
13/03/2021	Definição de grupo	1h	Augusto Benjamyn Leal Martins	1	
16/03/2021	Decisão em grupo sobre a linguagem a ser utilizada	1h	Augusto Benjamyn Leal Martins	1	
20/03/2021	Pesquisa sobre sockets	4h	Augusto Benjamyn Leal Martins	4	
22/03/2021	Pesquisa sobre sockets em c#	4h	Augusto Benjamyn Leal Martins	4	
25/03/2021	Realizados os primeiros testes de conexão por sockets em c#	6h	Augusto Benjamyn Leal Martins	6	
02/04/2021	Pesquisa sobre Conexões TCP/IP e Servidores	5h	Augusto Benjamyn Leal Martins	5	
03/04/2021	Sucesso na implementação de socket em C#	5h	Augusto Benjamyn Leal Martins	5	
10/04/2021	Início do desenvolvimento do projeto	6h	Augusto Benjamyn Leal Martins	6	
15/04/2021	Estudos sobre execução dinâmica de métodos entre classes	5h	Augusto Benjamyn Leal Martins	5	
20/04/2021	Estudo sobre transmissão de informações com eventos	4h	Augusto Benjamyn Leal Martins	4	
22/04/2021	Teste de conexão servidor	6h	Augusto Benjamyn Leal Martins	6	
27/04/2021	Implementação de eventos com delegate	4h	Augusto Benjamyn Leal Martins	4	
02/05/2021	Primeira versão funcional do programa	7h	Augusto Benjamyn Leal Martins	7	
10/05/2021	Adição de recursos e correção de bugs	8h	Augusto Benjamyn Leal Martins	8	
20/05/2021	Início do desenvolvimento da documentação	3h	Augusto Benjamyn Leal Martins	3	
21/05/2021	limpeza do código	3h	Augusto Benjamyn Leal Martins	3	
22/05/2021	correções finais e conclusão do projeto	2h	Augusto Benjamyn Leal Martins	2	
22/05/2021	Finalização da documentação	6h	Augusto Benjamyn Leal Martins	6	

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: CARLOS ALBERTO LEMOS MARTINS OLIVEIRA TURMA: CCSP17 RA: F036774

CURSO: CIÊNCIAS DA COMPUTAÇÃO CAMPUS: SOROCABA SEMESTRE: 5º TURNO: NOTURNO

CÓDIGO DA ATIVIDADE: 77B2 SEMESTRE: 5º ANO GRADE: 2021/1

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
13/03/2021	Definição do grupo	1	Carlos Alberto L M Oliveira	1	
16/03/2021	Definição do grupo sobre a linguagem a ser utilizada	1	Carlos Alberto L M Oliveira	1	
20/03/2021	Pesquisas sobre sockets	3	Carlos Alberto L M Oliveira	3	
22/03/2021	Pesquisas sobre uso de sockets em C#	4	Carlos Alberto L M Oliveira	4	
25/03/2021	Realizados os primeiros testes de sockets em C#	4	Carlos Alberto L M Oliveira	4	
03/04/2021	Obtido sucesso na implementação de sockets em C#	3	Carlos Alberto L M Oliveira	3	
10/04/2021	Início desenvolvimento aplicação	5	Carlos Alberto L M Oliveira	5	
15/04/2021	Estudos execução dinâmica de métodos entre threads	4	Carlos Alberto L M Oliveira	4	
20/04/2021	Estudos sobre transmissão de informações utilizando eventos	4	Carlos Alberto L M Oliveira	4	
24/04/2021	Estudos sobre delegates	5	Carlos Alberto L M Oliveira	5	
27/04/2021	Implementação de eventos com delegates	4	Carlos Alberto L M Oliveira	4	
02/05/2021	Finalização da primeira versão funcional do programa	7	Carlos Alberto L M Oliveira	7	
10/05/2021	Adição de recursos e correção de bugs	8	Carlos Alberto L M Oliveira	8	
16/05/2021	Adição de recursos no servidor e melhorias visuais no cliente	8	Carlos Alberto L M Oliveira	8	
17/05/2021	Melhoria usabilidade do cliente e tratamento exceções	6	Carlos Alberto L M Oliveira	6	
19/05/2021	Finalização da parte visual dos programas	4	Carlos Alberto L M Oliveira	4	
19/05/2021	Adição de múltiplos recursos no servidor e cliente	8	Carlos Alberto L M Oliveira	8	
20/05/2021	Limpeza de código e correção de bugs	6	Carlos Alberto L M Oliveira	6	
20/05/2021	Início desenvolvimento da documentação	3	Carlos Alberto L M Oliveira	3	
21/05/2021	Limpeza de código	3	Carlos Alberto L M Oliveira	3	
22/05/2021	Correções finais e conclusão do projeto	2	Carlos Alberto L M Oliveira	2	
22/05/2021	Finalização da documentação	6	Carlos Alberto L M Oliveira	6	

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Fabrício Oliveira Dias

TURMA: CC5P17

RA: D963236

CURSO: Ciência da Computação

CAMPUS: Sorocaba

SEMESTRE: 5ºSemestre

TURNO: Noite

CÓDIGO DA ATIVIDADE: 7782 SEMESTRE: 5ºSemestre ANO GRADE: 2021/1

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
13/03/2021	Definição do grupo	1H	Fabrício Oliveira Dias	1H	
16/03/2021	Discussão em grupo sobre a linguagem utilizada	2H	Fabrício Oliveira Dias	2H	
20/03/2021	Pesquisas sobre sockets	4H	Fabrício Oliveira Dias	4H	
22/03/2021	Pesquisa sobre o uso de sockets em C#	5H	Fabrício Oliveira Dias	5H	
25/03/2021	Realizado os primeiros testes de conexão por sockets em linguagem C#	6H	Fabrício Oliveira Dias	6H	
03/04/2021	Obtido sucesso a implementação de sockets em C#	5H	Fabrício Oliveira Dias	5H	
10/04/2021	Início do desenvolvimento do projeto	6H	Fabrício Oliveira Dias	6H	
15/04/2021	Estudos sobre execução dinâmica de métodos entre classes	5H	Fabrício Oliveira Dias	5H	
20/04/2021	Iniciado estudo sobre transmissão de informações com eventos	4H	Fabrício Oliveira Dias	4H	
24/04/2021	Realizado estudo sobre delegates	5H	Fabrício Oliveira Dias	5H	
02/05/2021	Finalização da primeira versão funcional do programa	7H	Fabrício Oliveira Dias	7H	
20/05/2021	Início do desenvolvimento da documentação	3H	Fabrício Oliveira Dias	3H	
20/05/2021	Limpeza de código e correção de bugs	6H	Fabrício Oliveira Dias	6H	
21/05/2021	Limpeza de código	3H	Fabrício Oliveira Dias	3H	
20/05/2021	Desenvolvimento do FrmChat e ConexãoUsuario na documentação	4H	Fabrício Oliveira Dias	4H	
21/05/2021	Desenvolvimento da conclusão na documentação	3H	Fabrício Oliveira Dias	3H	
22/05/2021	Desenvolvimento do TCP/IP na documentação	3H	Fabrício Oliveira Dias	3H	
22/05/2021	Correção finais e conclusão do projeto	2H	Fabrício Oliveira Dias	2H	
22/05/2021	Finalização da documentação	6H	Fabrício Oliveira Dias	6H	

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: _____ YURY RODRIGUES SHELKOVSKY _____ TURMA: _____ CC5P17 _____ RA: _____ F062163 _____

CURSO: _____ CIENCIA DA COMPUTAÇÃO _____ CAMPUS: _____ SOROCABA _____ SEMESTRE: _____ 5 _____ TURNO: _____ NOTURNO _____

CÓDIGO DA ATIVIDADE: _____ 77B2 _____ SEMESTRE: _____ 5º _____ ANO GRADE: _____ 2021/1 _____

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
13/03/2021	Definição do grupo	1	Yury Rodrigues Shelkovsky	1	
16/03/2021	Decisão em grupo sobre a linguagem a ser utilizada	1	Yury Rodrigues Shelkovsky	1	
20/03/2021	Pesquisas sobre soquetes	3	Yury Rodrigues Shelkovsky	3	
22/03/2021	Pesquisa sobre uso de soquetes em C#	4	Yury Rodrigues Shelkovsky	4	
25/03/2021	Realizados os primeiros testes de conexão por soquetes em C#	4	Yury Rodrigues Shelkovsky	4	
03/04/2021	Obtido sucesso na implementação de soquetes em C#	3	Yury Rodrigues Shelkovsky	3	
10/04/2021	Início do desenvolvimento do projeto	5	Yury Rodrigues Shelkovsky	5	
15/04/2021	Estudos sobre execução dinâmica de métodos entre Threads	4	Yury Rodrigues Shelkovsky	4	
20/04/2021	Iniciado estudo sobre transmissão de informações com eventos	4	Yury Rodrigues Shelkovsky	4	
24/04/2021	Realizado estudo sobre delegates	5	Yury Rodrigues Shelkovsky	5	
27/04/2021	Implementação de eventos no programa utilizando delegates	4	Yury Rodrigues Shelkovsky	4	
02/05/2021	Finalização da primeira versão funcional do programa	7	Yury Rodrigues Shelkovsky	7	
10/05/2021	Adição de recursos e correção de bugs	8	Yury Rodrigues Shelkovsky	8	
16/05/2021	Adição de recursos ao servidor e melhorias visuais do cliente	8	Yury Rodrigues Shelkovsky	8	
17/05/2021	Melhorada usabilidade do cliente e tratamento de exceções	6	Yury Rodrigues Shelkovsky	6	
19/05/2021	Finalização da parte visual das aplicações	4	Yury Rodrigues Shelkovsky	4	
19/05/2021	Adição de novas funcionalidades para servidor e para o cliente	8	Yury Rodrigues Shelkovsky	8	
20/05/2021	Início do desenvolvimento da documentação	3	Yury Rodrigues Shelkovsky	3	
20/05/2021	Limpeza de código e correção de bugs	6	Yury Rodrigues Shelkovsky	6	
21/05/2021	Limpeza de código	3	Yury Rodrigues Shelkovsky	3	
22/05/2021	Correções finais e conclusão do projeto	2	Yury Rodrigues Shelkovsky	2	
22/05/2021	Finalização da documentação	6	Yury Rodrigues Shelkovsky	6	

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AVALIAÇÃO: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO