

Relatório de Desenvolvimento

Sprint 3

Aluno 01: Alcivan Silva da Cruz

Aluno 02: Mizael Arthur da Silva Coelho

Sumário

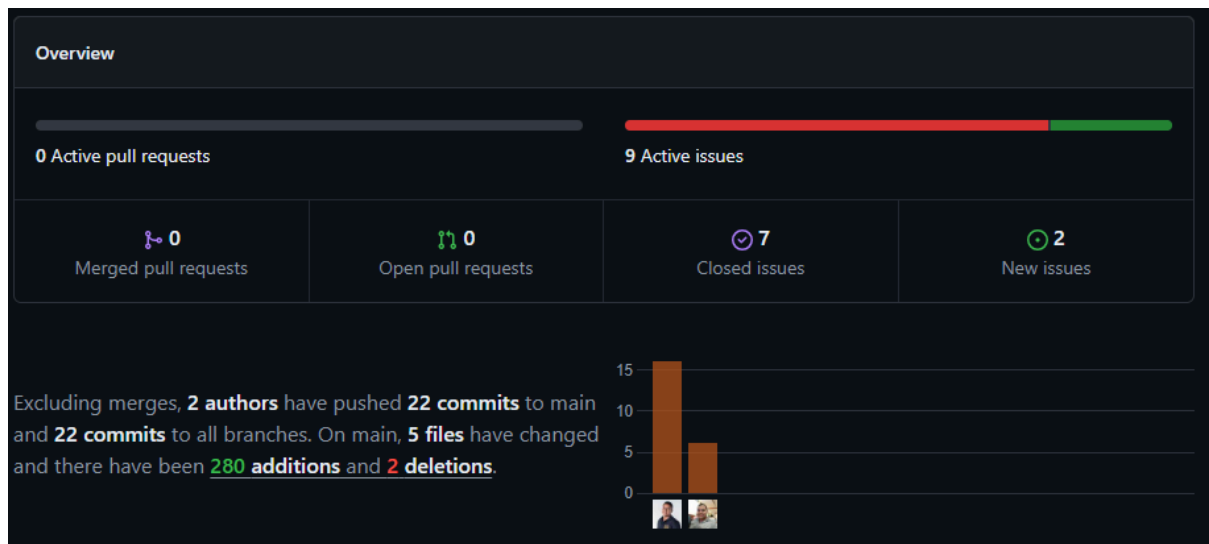
Introdução:	3
Planejamento do Sprint e Metas:	3
Desenvolvimento das Atividades:	4
Criar função Listar IP.....	4
Criar função Listar Interface.....	6
Criar Função Cadastro de usuário no Hotspot.....	10
Criar função Apagar usuário no Hotspot.....	11
Desafios e Problemas Enfrentados:	12
Reuniões da equipe:	12
Resultados Alcançados:	12

Introdução:

Este projeto propõe uma solução para a administração de serviços de redes em ambientes de pequeno porte, utilizando a plataforma de mensagens Telegram como interface de controle. A implementação de comandos em bot do Telegram permitirá aos administradores configurar e monitorar serviços de rede, de forma remota e conveniente.

Planejamento do Sprint e Metas:

O sprint 3 tinha como objetivo realizar a configuração das funcionalidades do bot. As atividades foram divididas em 4 atividades, sendo duas para cada aluno. As funções para esta versão incluem Listar os IPs em rede, Listar as interfaces de rede. Cadastrar e apagar usuários em um hotspot.



Resumo Semanal de Commits e Issues

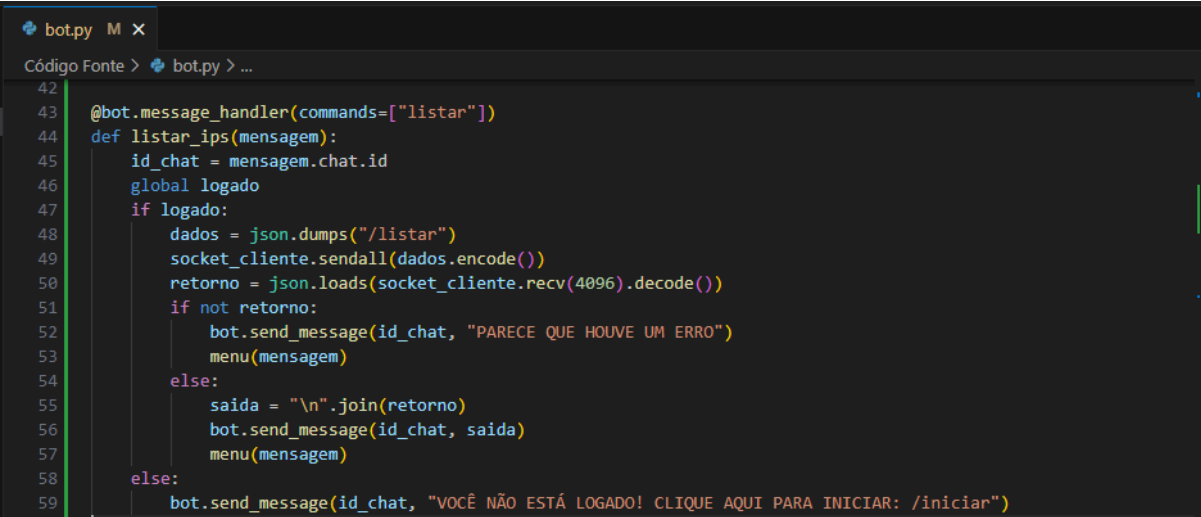
Desenvolvimento das Atividades:

Nesse sprint, Foram feitas as seguintes atividades, pelos seguintes membros:

1. Criar função Listar IP - Alcivan
2. Criar função Listar Interface - Alcivan
3. Criar Função Cadastro de usuário no Hotspot - Mizael
4. Criar função Apagar usuário no Hotspot - Mizael

Criar função Listar IP

Para atingir tais objetivos, foi necessário criar a função no arquivo “bot.py”.



```
42
43 @bot.message_handler(commands=["listar"])
44 def listar_ips(mensagem):
45     id_chat = mensagem.chat.id
46     global logado
47     if logado:
48         dados = json.dumps("/listar")
49         socket_cliente.sendall(dados.encode())
50         retorno = json.loads(socket_cliente.recv(4096).decode())
51         if not retorno:
52             bot.send_message(id_chat, "PARECE QUE HOUVE UM ERRO")
53             menu(mensagem)
54         else:
55             saida = "\n".join(retorno)
56             bot.send_message(id_chat, saida)
57             menu(mensagem)
58     else:
59         bot.send_message(id_chat, "VOCÊ NÃO ESTÁ LOGADO! CLIQUE AQUI PARA INICIAR: /iniciar")
60
```

Código da função Listar

O seu funcionamento se dá pela seguinte forma: a função quando é chamada envia via sockets TCP o comando para o servidor listar os IPs e retornar para o BOT que consequentemente irá retornar para o usuário os IPs. Quando o retorno chega, ele verifica se está tudo certo. Caso tudo esteja como deveria, ele envia a lista de IPs ao solicitante. Caso dê erro, ele envia o usuário para o menu novamente, informando que houve um erro.

Já do lado do servidor.py, precisamos chamar a função quando informado o “/listar”.

```
bot.py  servidor.py 2, U  conf_acesso.py U
Código Fonte > servidor.py > ...
1  import socket
2  import json
3  from conf_acesso import *
4  #from conexao import *
5  from threading import Thread
6
7  socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8  socket_servidor.bind(("127.0.0.1", 5000))
9  socket_servidor.listen()
10 dados = []
11
12 dicionario = {}
13
14 def exec(socket_cliente, address):
15     while True:
16         try:
17             #print('\n\n entrou no while')
18             dados = json.loads(socket_cliente.recv(1024).decode())
19             if isinstance(dados, dict):
20                 pass
21             elif isinstance(dados, list):
22                 pass
23             else:
24                 if dados == "/listar":
25                     #print(listar_lease())
26                     envio = json.dumps(listar_lease())
27                     socket_cliente.sendall(envio.encode())
28                 elif dados == "/listar_interface":
29                     envio = json.dumps(listar_enderecos())
30                     socket_cliente.sendall(envio.encode())
31         except:
32             pass
```

Código do Servidor.py que chama a função /listar

arquivo do servidor para realizar a solicitação de listar os IPs, ele verifica qual o comando solicitado pelo BOT e encaminha para a função que de fato irá executar o comando solicitado, o retorno é enviado diretamente ao BOT.

Por fim, criamos o arquivo que irá fazer o SSH com o mikrotik, equipamento utilizado para nossos testes.


```
bot.py M X  servidor.py 1, U  conf_acesso.py U
Código Fonte > bot.py > listar_interface
61 @bot.message_handler(commands=["listar_interface"])
62 def listar_interface(mensagem):
63     id_chat = mensagem.chat.id
64     global logado
65     if logado:
66         dados = json.dumps("/listar_interface")
67         socket_cliente.sendall(dados.encode())
68         retorno = json.loads(socket_cliente.recv(4096).decode())
69         if not retorno:
70             bot.send_message(id_chat, "PARECE QUE HOVE UM ERRO")
71             menu(mensagem)
72         else:
73             saida = "\n".join(retorno)
74             bot.send_message(id_chat, saida)
75             menu(mensagem)
76     else:
77         bot.send_message(id_chat, "VOCÊ NÃO ESTÁ LOGADO! CLIQUE AQUI PARA INICIAR: /iniciar")
78
79 def menu(mensagem):
80     id_chat = mensagem.chat.id
81     bot.send_message(id_chat, """ESCOLHA A OPÇÃO DESEJADA:
82 /listar LISTA TODOS OS IPs QUE ESTÃO NO LEASE DO DHCP
83 /listar_interface LISTA TODOS OS IPs DAS INTERFACES DE REDE
84 /cadastrar_usuario CADASTRA UM USUÁRIO NO HOTSPOT
85 /apagar_usuario APAGA UM USUÁRIO NO HOTSPOT
86 """)
87
88
89
90 # Retorna True toda vez que a função for chamada
91 def iniciar(mensagem):
92     return True
```

Função /listar_interfaces no código do bot.py

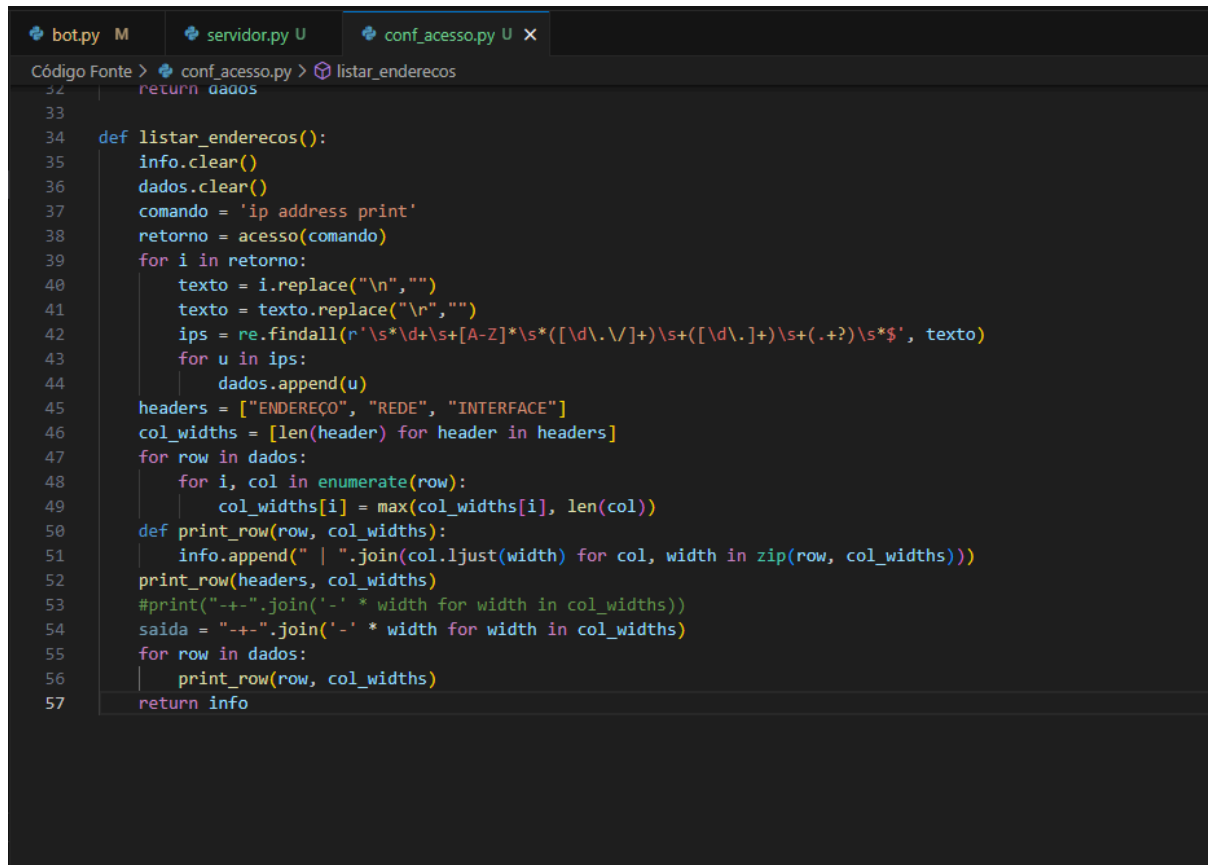
a função quando é chamada envia via sockets TCP o comando para o servidor listar os IPs das interfaces e retornar para o BOT que consequentemente irá retornar para o usuário as Interfaces de Redes com seus respectivos IPs.

```
bot.py  servidor.py 2, U  conf_acesso.py U
Código Fonte > servidor.py > ...
1  import socket
2  import json
3  from conf_acesso import *
4  #from conexao import *
5  from threading import Thread
6
7  socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
8  socket_servidor.bind(("127.0.0.1", 5000))
9  socket_servidor.listen()
10 dados = []
11
12 dicionario = {}
13
14 def exec(socket_cliente, address):
15     while True:
16         try:
17             #print('\n\n entrou no while')
18             dados = json.loads(socket_cliente.recv(1024).decode())
19             if isinstance(dados, dict):
20                 pass
21             elif isinstance(dados, list):
22                 pass
23             else:
24                 if dados == "/listar":
25                     #print(listar_lease())
26                     envio = json.dumps(listar_lease())
27                     socket_cliente.sendall(envio.encode())
28                 elif dados == "/listar_interface":
29                     envio = json.dumps(listar_enderecos())
30                     socket_cliente.sendall(envio.encode())
31         except:
32             pass
```

Código do servidor.py para chamar a função /listar_interfaces

essa é a parte do código do servidor.py com a função no servidor para realizar a solicitação de listar os IPs das interfaces de rede, ele verifica qual o comando solicitado pelo BOT e encaminha para a função que de fato irá executar o comando solicitado, o retorno é enviado diretamente ao BOT.

No mikrotik os comandos passados são diferentes, veja na imagem abaixo



```
32 return dados
33
34 def listar_enderecos():
35     info.clear()
36     dados.clear()
37     comando = 'ip address print'
38     retorno = acesso(comando)
39     for i in retorno:
40         texto = i.replace("\n", "")
41         texto = texto.replace("\r", "")
42         ips = re.findall(r'\s*\d+\s+[A-Z]*\s*([\d\.\/]+\s+([\d\.]+\s+(.+?)\s*$)', texto)
43         for u in ips:
44             dados.append(u)
45     headers = ["ENDEREÇO", "REDE", "INTERFACE"]
46     col_widths = [len(header) for header in headers]
47     for row in dados:
48         for i, col in enumerate(row):
49             col_widths[i] = max(col_widths[i], len(col))
50     def print_row(row, col_widths):
51         info.append(" | ".join(col.ljust(width) for col, width in zip(row, col_widths)))
52     print_row(headers, col_widths)
53     #print("-+-".join('-' * width for width in col_widths))
54     saida = "-+-".join('-' * width for width in col_widths)
55     for row in dados:
56         print_row(row, col_widths)
57     return info
```

Código para executar a função no mikrotik

A função `listar_enderecos` é chamada ela envia o comando para a função `acesso`, essa função é responsável por acessar o mikrotik e executar de fato o comando no equipamento, então ela retorna se o comando foi executado nesse caso ela retorna a lista de ips das interfaces para a função `listar_enderecos` que por sua vez pega o retorno do comando, formata e retorna para o servidor o resultado.

Criar Função Cadastro de usuário no Hotspot

Primeiro foi Efetuado a criação da função responsável por receber o pedido de cadastro, verificando se o usuário se encontra logado, caso logado, ele prossegue solicitando o usuário.

```
@bot.message_handler(commands=["cadastrar_usuario"])
def cadastrar_usuario(mensagem):
    id_chat = mensagem.chat.id
    global logado
    if logado:
        selecoes.append("/cadastrar_usuario")
        bot.send_message(id_chat, "INFORME O USUARIO DO HOTSPOT:")
        bot.register_next_step_handler(mensagem, usuario_hotspot)
    else:
        bot.send_message(id_chat, "VOCÊ NÃO ESTÁ LOGADO! CLIQUE AQUI PARA INICIAR: /iniciar")
```

Criação da função no bot.py

Logo em seguida, a Função que recebe o nome informado na resposta da mensagem anterior, e prossegue para a solicitação da senha.

```
def usuario_hotspot(mensagem):
    id_chat = mensagem.chat.id
    selecoes.append(mensagem.text)
    bot.send_message(id_chat, "INFORME A SENHA DO HOTSPOT:")
    bot.register_next_step_handler(mensagem, senha_hotspot)
```

Função para o nome do usuário do hotspot

Logo em seguida, com o nome do usuário em posse, ele solicita a senha, e chama a função para a mesma.

```
96 def senha_hotspot(mensagem):
97     id_chat = mensagem.chat.id
98     selecoes.append(mensagem.text)
99     envio = json.dumps(selecoes)
100     socket_cliente.sendall(envio.encode())
101     retorno = json.loads(socket_cliente.recv(4096).decode())
102
103     if retorno == "comando realizado com sucesso":
104         bot.send_message(id_chat, "USUARIO CADASTRADO COM SUCESSO!")
105         menu(mensagem)
106     else:
107         bot.send_message(id_chat, "USUÁRIO JÁ EXISTE!")
108         menu(mensagem)
```

Função para receber a senha solicitada

ao final ele irá informar, caso ocorra tudo certo, se o usuário foi(ou não) cadastrado com sucesso. resta verificar como testar se o usuario existe, possivelmente aplicado junto ao banco de dados nas próximas sprints

Criar função Apagar usuário no Hotspot

seguindo o mesmo procedimento do código anterior, para apagar o usuário, o processo é mais simples. primeiramente fizemos a função, da qual o servidor irá chamar.

```
@bot.message_handler(commands=["apagar_usuario"])
def apagar_usuario(mensagem):
    id_chat = mensagem.chat.id
    global logado
    if logado:
        selecoes.append("/apagar_usuario")
        bot.send_message(id_chat, "INFORME O USUARIO DO HOTSPOT A SER APAGADO:")
        bot.register_next_step_handler(mensagem, usuario_apagar)
    else:
        bot.send_message(id_chat, "VOCÊ NÃO ESTÁ LOGADO! CLIQUE AQUI PARA INICIAR: /iniciar")
```

Função para apagar usuário

Primeiramente, assim como na função anterior, verificamos se o usuário está logado com o bot. caso sim, ele solicita o nome do usuário que será deletado do hotspot. logo em seguida, uma função coleta a informação.

```
def usuario_apagar(mensagem):
    id_chat = mensagem.chat.id
    selecoes.append(mensagem.text)
    envio = json.dumps(selecoes)
    socket_cliente.sendall(envio.encode())
    retorno = json.loads(socket_cliente.recv(4096).decode())
    if retorno == "comando realizado com sucesso":
        bot.send_message(id_chat, "USUARIO APAGADO COM SUCESSO")
        menu(mensagem)
    else:
        bot.send_message(id_chat, "USUÁRIO INEXISTENTE!")
        menu(mensagem)
    selecoes.clear()
```

Função para tratar a resposta do usuário

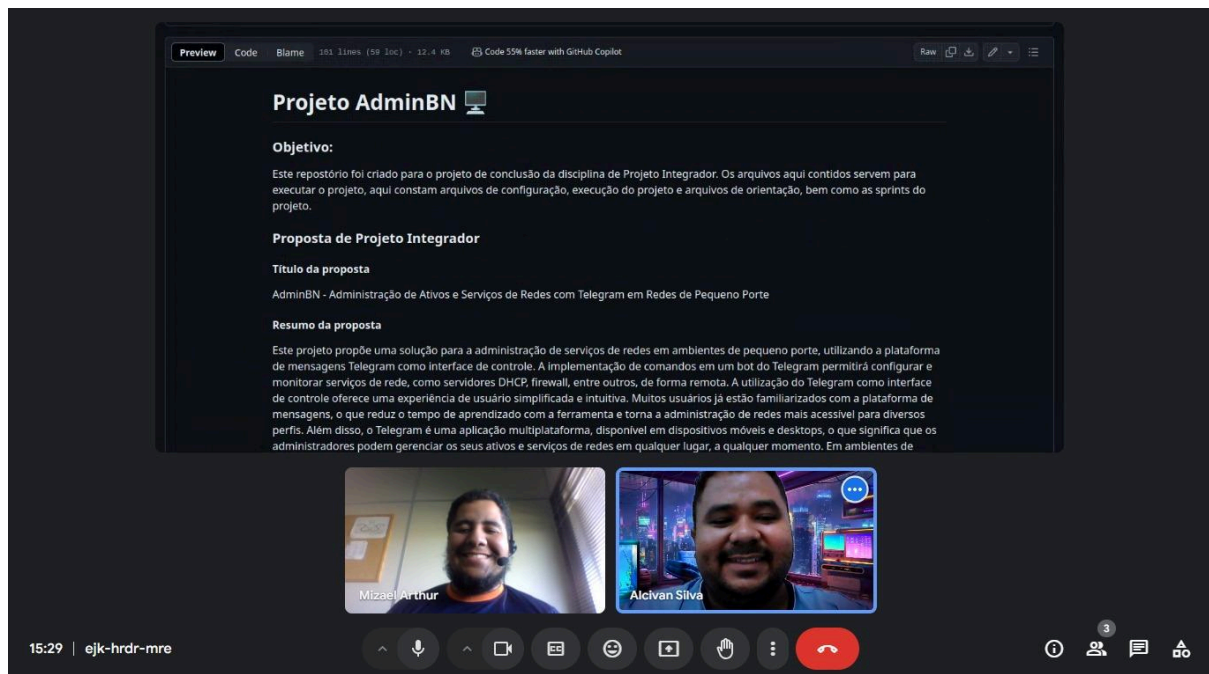
assim, o usuário é deletado.

Desafios e Problemas Enfrentados:

Dentre os dias do Sprint 3, o problema encontrado foi o tratamento de alguns erros, tendo em vista o tempo disponível. Analisando os resultados desse Sprint, será tirado um tempo no sprint seguinte para testes, afim de sanar ou ao menos minimizar bugs e tratamentos

Reuniões da equipe

No dia 12/08 foi realizado uma reunião entre os alunos, nessa reunião, foram decididos alguns tópicos sobre como seria o funcionamento das funcionalidades em relação ao equipamento utilizado, que no caso é Mikrotik. As funções teriam que passar pelo SSH e serem executadas no equipamento de destino.



No dia 13/08 foi feita uma reunião com o orientador, onde o mesmo passou para a equipe orientações a respeito do código dos quais os alunos estavam com dúvidas. por ser uma reunião curta e presencial, não houve ata, apenas orientação.

Resultados Alcançados:

Por fim, acreditamos que alcançamos o proposto para o primeiro sprint, tendo em vista que precisamos ainda alinhar alguns pontos de reuniões, mas foi perceptível que a equipe é sim capaz de concluir o projeto.