

UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ARTES, CIÊNCIAS e HUMANIDADES
GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Prof. Dr. João Bernardes

Aaron Ferraz do Amaral Martins da Silva - 14554474 - T04

Paulo Ubiratan Muniz Rodrigues - 14748678 - T94

Marcos Medeiros da Silva Filho - 14594271 - T94

Relatório do EP da matéria Redes de Computadores

São Paulo

2024

SUMÁRIO

1 PROPOSTA	3
2 ESPECIFICAÇÃO DA SOLUÇÃO	4
2.1 COMO COMPILAR E EXECUTAR.....	4
2.1.1 Executáveis.....	4
2.1.2 Servidor.....	5
2.1.3 Cliente	5
2.2 ARQUITETURA	9
2.3 PROTOCOLO PARA AS MENSAGENS	10
3 ESPECIFICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO	10
4 CÓDIGO	22
5 TESTES E RESULTADOS	26
5.1 OBSERVAÇÕES	26
5.2.1 Administradores e Usuários	26
5.2.2 Login	27
5.2.3 Zones	28
5.2.4 Messages.....	29
5.2.5 Notifications	29
5.2.6 Update User.....	29
5.2.7 Association	30
5.2.8 User	31
6 FONTES	32

1 PROPOSTA

O projeto do nosso grupo consiste em um sistema de monitoramento em tempo real, desenvolvido para gerenciar dispositivos conectados em diferentes áreas de segurança, chamadas "zonas de perigo". O sistema monitora e controla esses dispositivos, enviando alertas sobre invasões de áreas restritas e mantendo uma comunicação constante com o servidor central por meio de mensagens. Entre as funcionalidades oferecidas, estão a autenticação, o monitoramento do status dos dispositivos e a notificação de eventos críticos.

Além disso, o sistema permite que o servidor gerencie e atualize dispositivos (clientes) e zonas, fornecendo alertas em tempo real sempre que um dispositivo entra em uma zona de risco. Dessa forma, o sistema assegura que os dispositivos permaneçam conectados e operacionais.

Mensagens trocadas entre os dispositivos (clientes) e o servidor:

1. Gerenciamento de Dispositivos e Zonas:

- CREATE_CLIENT: Cria novos dispositivos que serão monitorados e associados ao sistema.
- CREATE_ZONE: Define zonas de perigo, áreas específicas que, ao serem invadidas, podem gerar alertas.
- UPDATE_CLIENT: Atualiza o status e as informações dos dispositivos monitorados.

2. Notificações e Alertas:

- NOTIFICATION: Gera alertas quando um dispositivo entra em uma zona de perigo, possibilitando ações corretivas imediatas.
- SUCESS_STORE: Confirma o sucesso na criação de novos dispositivos ou zonas, facilitando o acompanhamento das operações.

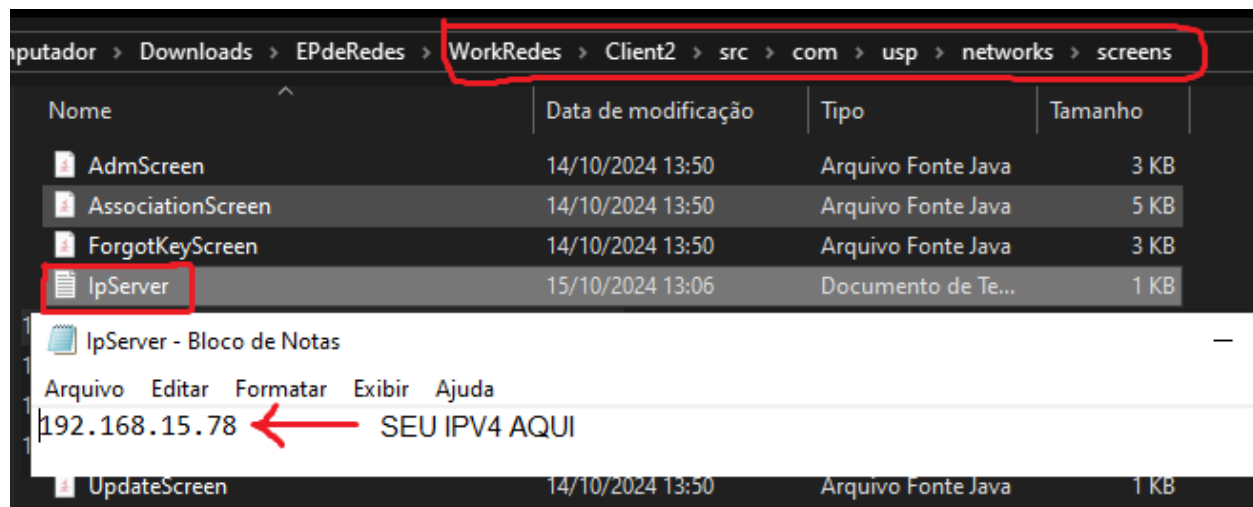
3. Autenticação e Controle de Sessões:

- LOGIN: Permite o login de dispositivos monitorados e dos administradores que gerenciam o sistema.
- ERROR_PASSWORD: Informa sobre tentativas de login fracassadas devido a senha incorreta.

2 ESPECIFICAÇÃO DA SOLUÇÃO

2.1 COMO COMPILAR E EXECUTAR

O projeto do nosso grupo foi desenvolvido na IDE Eclipse, o que permitiu que o código fosse compilado automaticamente à medida que era escrito. Além disso, é fundamental modificar o arquivo 'IpServer' no diretório "\\WorkRedes\\Client2\\src\\com\\usp\\networks\\screens". Neste arquivo, você deve inserir o endereço IPV4 da sua rede, conforme a figura abaixo ilustra:



Para executar o programa, a fim de evitar problemas, é interessante começar pelo servidor. E, para executar o programa, siga estas instruções:

2.1.1 Executáveis

Para facilitar a execução do projeto, criamos dois executáveis: um para o cliente e outro para o servidor, ambos localizados no diretório "\\WorkRedes\\exe". Primeiro, inicie o servidor clicando duas vezes no arquivo "MyServer" para garantir que a comunicação esteja disponível. Em seguida, execute o cliente clicando duas vezes no arquivo "MyClient".

2.1.2 Servidor

Se você preferir iniciar o servidor sem utilizar o executável, abra o terminal no Windows ou Linux e navegue até o diretório "WorkRedes/Server". No meu caso, o caminho é o seguinte:

```
C:\Users\paulo\Downloads\EPdeRedes\WorkRedes\Server>
```

Em seguida, execute os comandos abaixo no terminal:

```
mvn clean install
```

E, por último, digite:

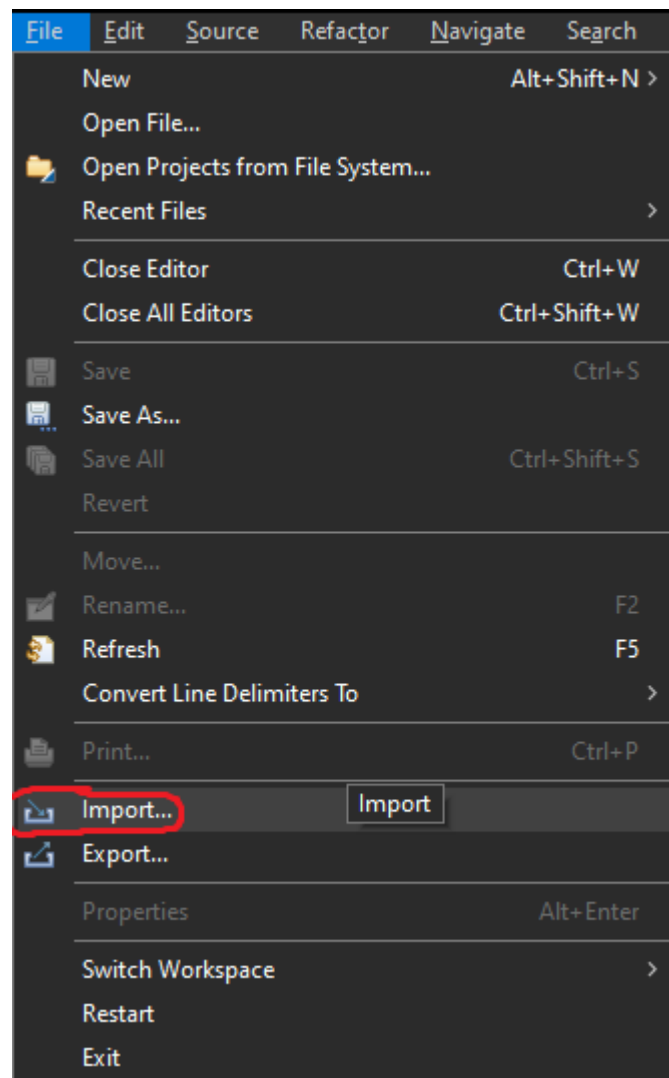
```
mvn exec:java -Dexec.mainClass="com.usp.networks.app.AppServer"
```

Assim, o servidor está em execução e pronto para receber conexões e interagir com o cliente. Para confirmar, verifique se o terminal do servidor exibe uma mensagem semelhante a:

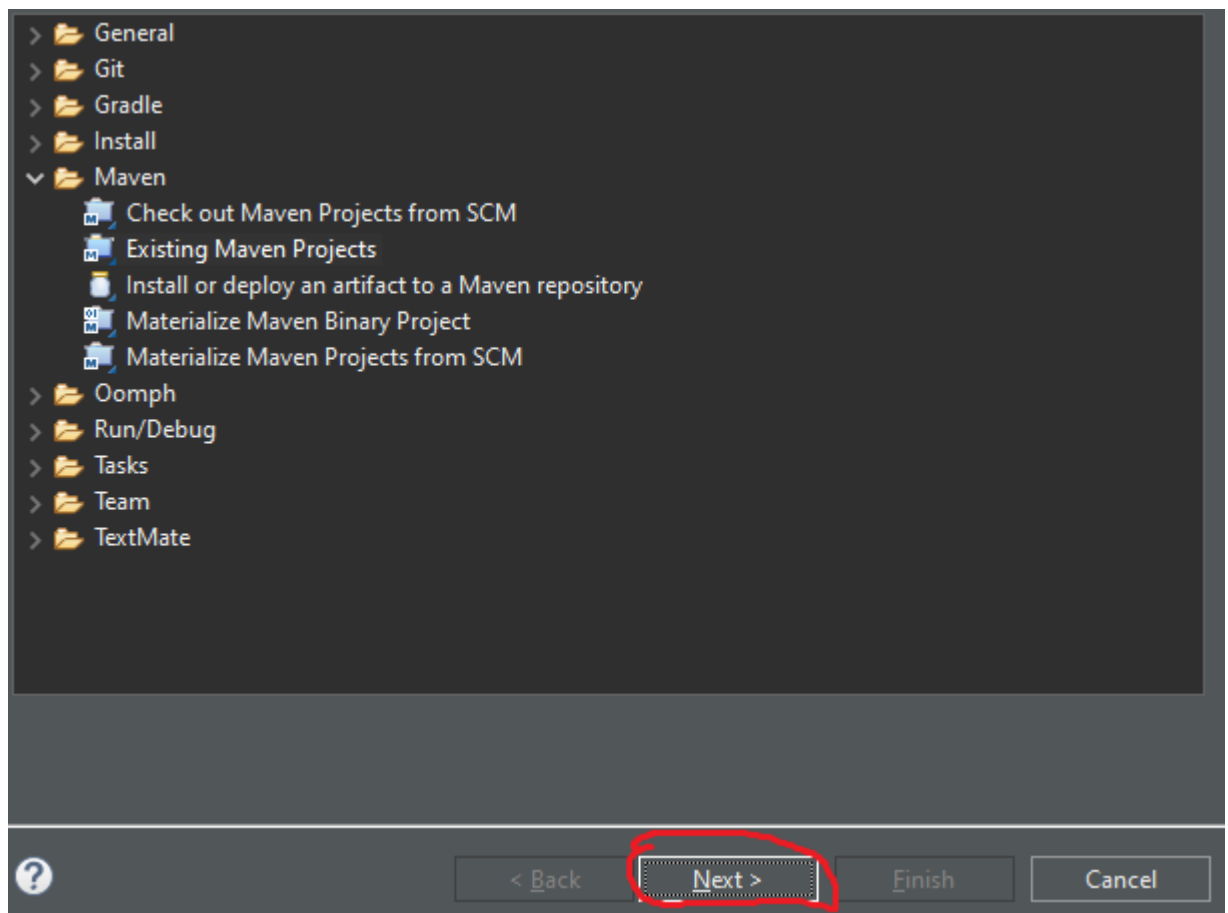
```
C:\Users\paulo\Downloads\EPdeRedes\WorkRedes\exe\MyServer.exe  
Server started and waiting for connections...
```

2.1.3 Cliente

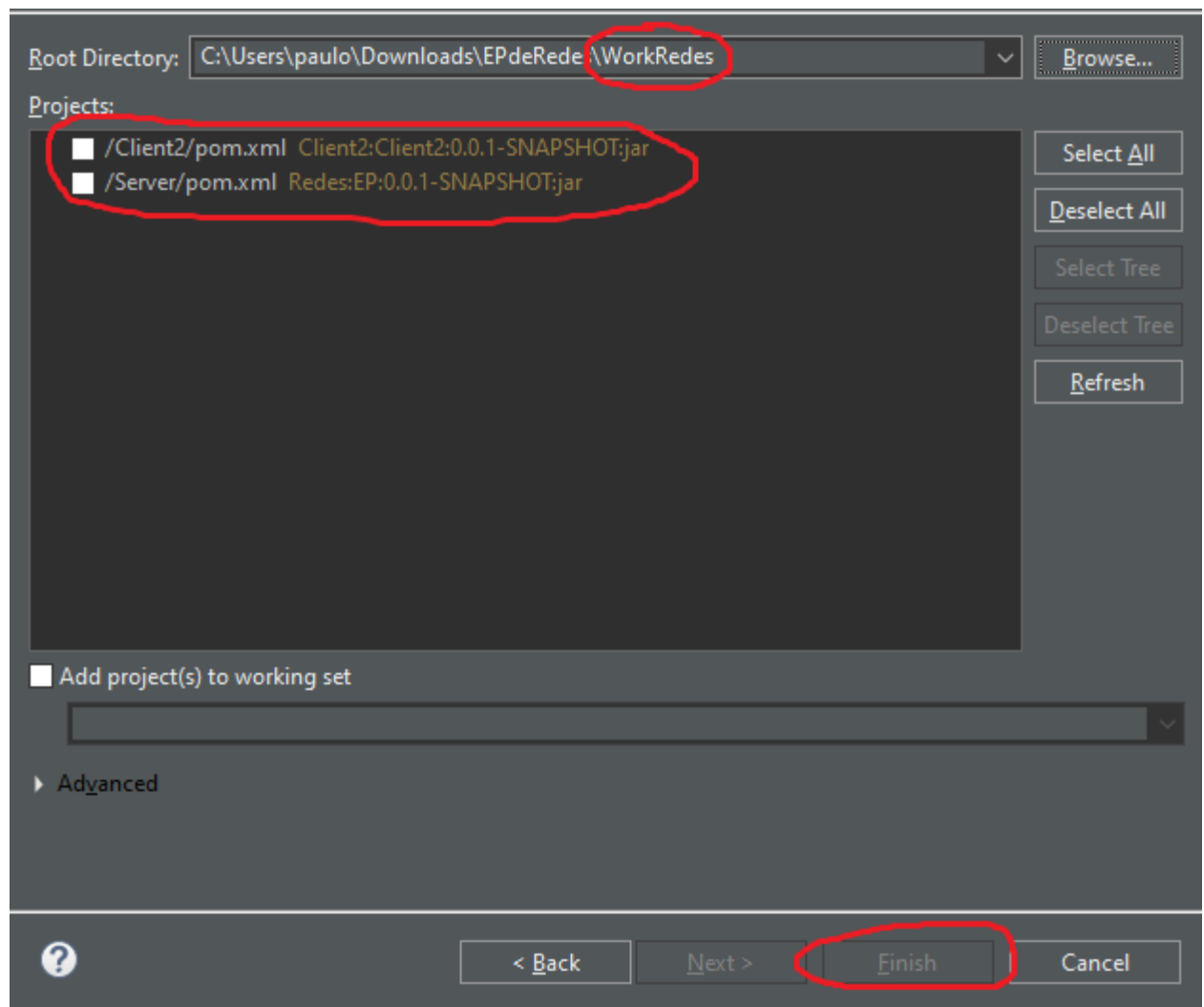
Para executar o cliente sem utilizar o executável, importe o projeto no Eclipse. Para isso, clique em “File” no canto superior esquerdo da tela, conforme ilustrado na imagem abaixo.



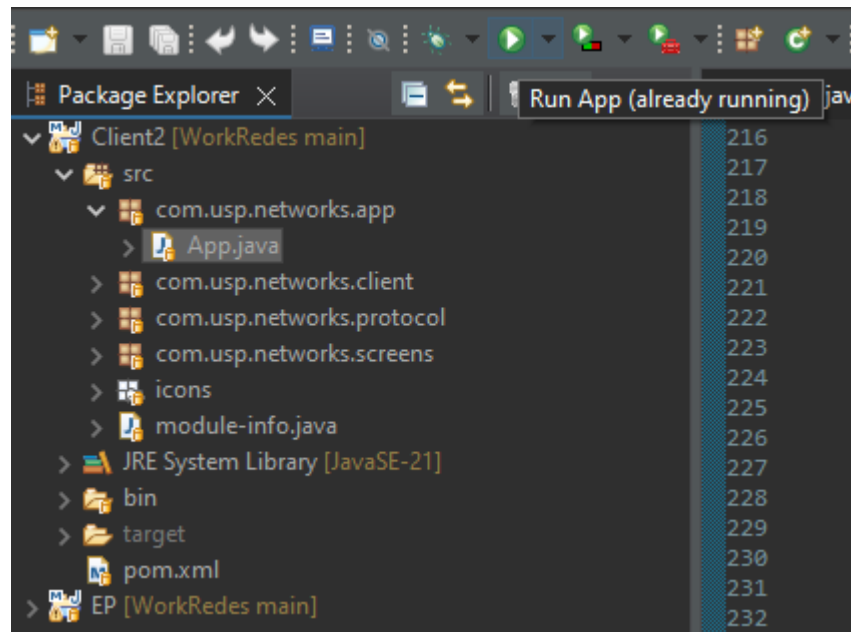
Em seguida, clique em “Import...” e depois em “Existing Maven Projects” e clique em “Next” no canto inferior da página , como na imagem abaixo:



Em seguida, selecione o caminho até a pasta “WorkRedes”, que, neste exemplo, está localizada em “C:\Users\paulo\Downloads\EPdeRedes\WorkRedes”. Marque as caixas correspondentes a “/Client2/” e “/Server/” e, em seguida, clique em “Finish” no canto inferior da tela, conforme mostrado na imagem abaixo.



Pronto, agora o projeto poderá ser colocado para rodar. Para isso, basta encontrar a classe “App.java” na aba “Package Explorer”, expanda “Client2”, em seguida expanda a pasta “src” e, por último, expanda “com.usp.networks.app”. Clique em “App.java”, e depois clique no botão “Run App”, como é exibida na imagem abaixo:



Assim, estaremos iniciando a execução da aplicação correspondente ao cliente.

2.2 ARQUITETURA

A arquitetura cliente-servidor é a mais adequada para o nosso projeto de monitoramento em tempo real devido à sua centralização e controle sobre a comunicação entre os dispositivos e o servidor. No projeto do grupo, o servidor central é responsável por gerenciar dispositivos em diferentes áreas de segurança, autenticar os clientes, monitorar o status de zonas de perigo e enviar notificações e alertas críticos quando necessário. Essa centralização é essencial para garantir que todas as informações importantes passem por um único ponto de controle, facilitando a gestão e a atualização dos dados e permitindo uma resposta rápida a qualquer evento crítico.

Por outro lado, a arquitetura Peer-to-peer (P2P) distribui responsabilidades igualmente entre os dispositivos, o que significaria que cada dispositivo precisaria coordenar diretamente com outros para monitoramento e alerta, complicando a sincronização dos dados e aumentando a possibilidade de inconsistências. Em um sistema de monitoramento de segurança, onde a confiabilidade e a integridade dos dados são cruciais, essa abordagem não seria ideal, pois dispositivos P2P podem sofrer com a falta de controle centralizado, dificultando a administração e escalabilidade de uma maneira eficaz.

Sendo assim, a arquitetura cliente-servidor foi escolhida para o projeto porque oferece centralização e controle, permitindo que o servidor gerencie dispositivos, autentique clientes e envie notificações de forma eficiente e consistente. Diferente da arquitetura Peer-to-peer (P2P), que distribui responsabilidades entre os dispositivos, a cliente-servidor evita problemas de sincronização e inconsistência, garantindo a integridade e confiabilidade necessárias para um sistema de monitoramento de segurança.

2.3 PROTOCOLO PARA AS MENSAGENS

O protocolo TCP é mais adequado para este projeto porque ele oferece uma comunicação confiável, garantindo que todas as mensagens entre os dispositivos e o servidor sejam entregues corretamente e na ordem correta. No contexto do monitoramento em tempo real, mensagens como notificações de invasão e atualizações de status são críticas e devem ser entregues sem falhas. O TCP oferece garantias de integridade e retransmissão de pacotes caso alguma mensagem se perca, o que é fundamental para evitar falhas de comunicação que poderiam resultar em problemas de segurança.

Por outro lado, o UDP, apesar de ser mais rápido por não ter controle de erros e confirmação de entrega, não é ideal para um sistema onde a perda de pacotes e a ordem das mensagens são preocupações significativas. A confiabilidade do TCP assegura que cada alerta, atualização e mensagem de autenticação seja entregue sem perda, mantendo a integridade do sistema de monitoramento e permitindo que o servidor tome decisões informadas e em tempo hábil.

Portanto, o protocolo UDP não é adequado para este projeto, pois não oferece controle de erros, confirmação de entrega, ou garantia de ordem nas mensagens, o que poderia resultar em perda de dados críticos e desordem nas notificações. Em contraste, o TCP proporciona comunicação confiável, assegurando que cada mensagem seja entregue corretamente e na ordem certa, essencial para manter a integridade e a segurança do sistema de monitoramento em tempo real.

3 ESPECIFICAÇÃO DO PROTOCOLO DE COMUNICAÇÃO

3.1 ESTRUTURA E FORMATAÇÃO

A comunicação entre o servidor e o cliente é estruturada de forma que cada mensagem contém um campo de comando, que é a primeira parte da mensagem e é sempre escrito em letras maiúsculas, seguido de seus respectivos parâmetros. Cada parâmetro é delimitado por aspas e separado por ponto e vírgula, e a mensagem é finalizada com um caractere de dois pontos (:), indicando o seu término. É importante destacar que cada tipo de comando possui parâmetros específicos que devem ser respeitados para garantir a correta interpretação da mensagem.

3.2 MENSAGENS

Neste trecho, discutem-se as mensagens enviadas, incluindo sua estrutura, campos e delimitadores. Cada campo pode assumir determinados valores, os quais têm significados específicos. Além disso, descrevemos o que ocorre quando o servidor recebe essas mensagens e como realiza seu processamento.

```
String message = "\"LIST-USER\"";
```

A mensagem `String message = "\"LIST-USER\"";` contém um único comando para listar usuários. Ela é escrita em letras maiúsculas e possui uma estrutura simples, com apenas um campo de comando seguido por (:) como separador.

Ao receber essa solicitação, o servidor processa o comando "LIST-USER", consulta o banco de dados para listar todos os usuários (exceto o administrador), e formata cada registro em uma string delimitada por vírgulas e cada registro separado por ponto e vírgula (;). Essa lista é então enviada de volta ao cliente. Se ocorrer um erro durante o processo, o servidor retorna uma mensagem de erro apropriada.

Request:

“LIST-USER”:

Response:

“PACK”; (Content-list):

“MSG”; “Database error”:

“MSG”; “Server not available”:

“PACK” -> significa que o servidor vai voltar uma lista de determinada requisição

ex:

“PACK”;”joão,ferreira,joao@usp.br,123,1;pedro,paulo,pedro@usp.br,1234,0”:

```
String message = "\\LIST-ZONE\":";
```

A mensagem `String message = "\\LIST-ZONE\":";` contém um único comando, "LIST-ZONE", que solicita a lista de zonas registradas no banco de dados. O termo "ZONE" é escrito em letras maiúsculas, indicando o campo de ação, seguido por um (:), que age como um delimitador final.

Ao receber essa mensagem, o servidor aciona o `ListZoneHandler`, que consulta a tabela 'Zone' no banco de dados. A resposta gerada é uma lista de zonas, onde cada zona é representada por atributos específicos (id, x, y, radius), separados por vírgulas e delimitados por aspas. As zonas listadas são separadas entre si por ponto e vírgula, retornando ao cliente a lista completa das zonas solicitadas.

Request:

“LIST-ZONE”:

Response:

“PACK”;(Content-list):

“MSG”;”Database error”:

“MSG”;”Server not available”:

“PACK” -> significa que o servidor vai voltar uma lista de determinada requisição

ex:

“PACK”;”1,2.5,14”;”2,4,9”:

```
String msg = "\\CREATE-ASS\":" + valorIdUser + "\\;" + zoneSelect + "\\:";
```

A mensagem `String msg = "\"CREATE-ASS\";\";\" + valorIdUser + "\";\";\" + zoneSelect + "\":\""` possui três campos e é usada para solicitar a criação de uma associação entre um usuário e uma zona. Ela começa com o comando "CREATE-ASS" em letras maiúsculas, seguido pelo valores inteiros `valorIdUser`, que representa o ID do usuário, e `zoneSelect`, que representa o ID da zona. Os campos são separados por ponto e vírgula (;), e a mensagem é finalizada com dois pontos (:) como terminador.

Ao receber essa solicitação, o servidor identifica o comando "CREATE-ASS" e aciona o `CreateAssHandler`, que abre uma conexão com o banco de dados e executa uma inserção na tabela `UserZoneMapping` para associar o `user_id` ao `zone_id`. O servidor então confirma a criação da associação ao cliente com uma mensagem de sucesso ou, em caso de falha, retorna uma mensagem de erro apropriada.

Request:

`"CREATE-ASS";id_user;id_zone":`

Response:

`"MSG";Association Zone-User created with success":`

`"MSG";Unable to register association zone-user":`

`"MSG";Server not available":`

```
String msg = "\"DELETE-ASS\";\";\" + valorIdUser + "\";\";\" + zoneSelect + "\":\";
```

A mensagem `String msg = "\"DELETE-ASS\";\";\" + valorIdUser + "\";\";\" + zoneSelect + "\":\""` contém um comando para excluir uma associação entre usuário e zona. Ela é escrita em letras maiúsculas e possui uma estrutura com três campos: o comando "DELETE-ASS", seguido pelos valores inteiros `valorIdUser`, que refere-se ao Id do usuário, e `zoneSelect`, que refere-se ao Id da zona. Os valores são separados por ponto e vírgula (;), com dois-pontos (:) indicando o fim da mensagem.

Ao receber essa solicitação, o servidor processa o comando "DELETE-ASS", conecta-se ao banco de dados e remove o registro na tabela `UserZoneMapping` onde o `user_id` e o `zone_id` correspondem aos valores fornecidos. Em seguida, o servidor envia uma resposta ao cliente confirmando o sucesso da exclusão ou informando um erro, caso a operação falhe.

Request:

“DELETE-ASS”;”id_user”;”id_zone”:

Response:

“MSG”;”Association Zone-User deleted with success”:

“MSG”;”Unable to delete association zone-user”:

“MSG”;”Server not available”:

```
String message = "\"UPDATE-PASSWORD\";\";\"" + getUserField() + "\";\";\"" + new  
String(getPasswordField()) + "\";\";\"" + new String(this.getConfirmField()) + "\";\";\";
```

A mensagem acima indica uma solicitação para atualizar a senha de um usuário e segue uma estrutura padronizada: quatro campos separados por ponto e vírgula (;), finalizados com dois pontos (:). O primeiro campo, "UPDATE-PASSWORD", é o comando de atualização; o segundo campo, getUserField() pode ser uma string e representa o login do usuário; o terceiro, new String(getPasswordField()), contém a nova senha que também é uma string; e o quarto, new String(getConfirmField()), é a confirmação da senha e pode ser tanto string quanto valores inteiros.

Esses valores permitem ao servidor identificar o tipo de operação e o usuário-alvo, verificar se a nova senha e a confirmação coincidem e, em caso positivo, atualizar o banco de dados com a nova senha. Se as senhas não coincidem ou se houver um problema (como um usuário inexistente), o servidor responde com uma mensagem de erro apropriada.

Request:

“UPDATE-PASSWORD”;”login_user”;”password”;”confirm_password”:

Response:

“MSG”;”User updated with success”:

“MSG”;”Unable to updated user”:

“MSG”;”Server not available”:

```
String message = "LOGIN;" + getUserField() + ";" + new String(getPasswordField()) + ";;";
```

A mensagem acima contém um comando para autenticar um usuário. Escrita em letras maiúsculas, ela segue uma estrutura de três campos: o comando `"LOGIN"`, o nome de usuário (que é uma string) e a senha (podendo ser uma string ou números). Cada campo é separado por ponto e vírgula (`;`), e a mensagem é finalizada com um delimitador de dois-pontos (`:`) para indicar o fim dos dados.

Ao receber essa solicitação, o servidor processa o comando "LOGIN", consulta o banco de dados para localizar o nome de usuário e, se encontrado, verifica se a senha fornecida corresponde à armazenada. Caso a senha esteja correta, o servidor retorna uma mensagem de sucesso com informações adicionais, como o status de administrador e o ID do usuário. Se o usuário não for encontrado ou a senha estiver incorreta, o servidor responde com uma mensagem de erro apropriada.

Request:

`"LOGIN";"admin@usp.br";"1234":`

Response:

`"MSG";"User not found":`

`"MSG";"Incorrect password":`

`"MSG";"Connect with success"; "isAdmin"; "id_user":`

`"MSG";"Database error":`

`"MSG";"Server not available":`

```
String msg = "\"SEND\";" + userSelect + ";" + Login.getLogin() + ";" + message + ":";
```

A mensagem acima é uma solicitação para enviar uma notificação. Ela é escrita com campos delimitados por aspas duplas (`"`) e separados por ponto e vírgula (`;`), terminando com o caractere `:`. A estrutura da mensagem contém o comando `"SEND"` em letras maiúsculas, seguido por três campos.

O segundo campo, ``userSelect``, é uma string que representa o destinatário, enquanto o terceiro campo, ``Login.getLogin()``, contém o login (podendo ser apenas números inteiros ou string) do remetente. O último campo, ``message`` é uma string e inclui o conteúdo da notificação, ou seja, o texto. Esses valores variam conforme o destinatário, remetente e conteúdo específico da mensagem a ser enviada.

Ao receber essa mensagem, o servidor processa o comando ``"SEND"``, extrai os campos ``user_id``, ``sender`` e ``message``, e então insere esses dados na tabela ``Notifications`` do banco de dados. Caso a operação seja bem-sucedida, o servidor retorna uma confirmação de sucesso; se ocorrer algum erro, ele responde com uma mensagem de erro apropriada.

Request:

`"SEND";"login_user";"login_admin";"message":`

Response:

`"MSG";"Sent with success":`

`"MSG";"Recipient doesn't exist":`

`"MSG";"Database error":`

`"MSG";"Server not available":`

```
String msg = "\"LIST-NOTIFY\":" + Login.getLogin() + "\";" + Login.getAdmin() + "\"";
```

A mensagem acima é uma solicitação para listar notificações, composta por três campos separados por ; e finalizada por dois pontos (":"). O primeiro campo, "LIST-NOTIFY", indica o comando; o segundo, Login.getLogin(), especifica o usuário; e o terceiro, Login.getAdmin(), define se o usuário é administrador (true ou false).

O servidor interpreta esses campos, consulta o banco de dados com o ListNotifyHandler e retorna notificações apropriadas: notificações administrativas, se true, ou específicas para o usuário, se false. Em caso de erro, uma mensagem de erro é enviada de volta ao cliente.

Request:

`"LIST-NOTIFY";"login";"isAdmin":`

Response:

`"PACK";(Content-list):`

`"MSG";"Database error":`

`"MSG";"Server not available":`

```
String msgXY = "XY;" + Login.getLogin() + ";" + getLatitude() + ";" + getLongitude() + ";";
```


A mensagem acima possui quatro campos e é utilizada para solicitar a verificação de localização de um usuário. Ela começa com o comando `"XY"` em letras maiúsculas, seguido pelo valor `Login.getLogin()`, que representa o login do usuário, e os valores `double` da `getLatitude()` e `getLongitude()`, que representam, respectivamente, a latitude e a longitude do usuário. Os campos são separados por ponto e vírgula (;), e a mensagem é finalizada com dois pontos (:) como terminador.

Ao receber essa solicitação, o servidor identifica o comando `"XY"` e aciona o `XYHandler`, que abre uma conexão com o banco de dados e consulta a localização do usuário com base em suas associações com zonas específicas. Em seguida, o servidor calcula a distância entre as coordenadas fornecidas e as zonas restritas. Se o usuário estiver dentro de uma zona proibida, o servidor gera uma notificação de invasão. Por fim, o servidor confirma a verificação ao cliente com uma mensagem de sucesso ou, em caso de erro, retorna uma mensagem de falha apropriada.

Request:

`"XY";"login";"x";"y":`

Response:

`"NOTIFY";"12";"Invaded a non-permitted zone(12)":`

`"MSG";"Successfully verified":`

`"MSG";"Unable to verify":`

`"MSG";"Server not available":`

```
String msg = "\"DELETE-ZONE\";\"" + rs[0] + "\"";
```

A mensagem `String msg = "\"DELETE-ZONE\";\"" + rs[0] + "\""` contém um comando para excluir uma zona específica no sistema. Ela é escrita em letras maiúsculas e possui uma estrutura com dois componentes: o comando `"DELETE-ZONE"`, seguido pelo valor `rs[0]`, que se refere ao `ID` da zona a ser excluída e é declarado como uma string. Os valores são separados por ponto e vírgula (;), e a mensagem é finalizada com dois pontos (:) para indicar o término.

Ao receber essa solicitação, o servidor processa o comando `"DELETE-ZONE"`, conecta-se ao banco de dados e executa uma operação para remover o registro correspondente ao `ID` da zona fornecido. Em seguida, o servidor envia uma resposta ao cliente confirmando o sucesso da exclusão ou indicando um erro, caso a operação falhe.

Request:

“DELETE-ZONE”;”id_zone”:

Response:

“MSG”;”Zone deleted with success”:

“MSG”;”Unable to delete zone”:

“MSG”;”Server not available”:

```
String msgCreate = "\"CREATE-ZONE\";" + latitude + ";" + longitude + ";" + radius +  
"\"";
```

A mensagem `msgCreate` é uma solicitação de criação de zona, escrita no formato `"CREATE-ZONE";latitude";longitude";radius":`, onde os campos são separados por ponto e vírgula (;) e delimitados por aspas duplas. O primeiro campo, `"CREATE-ZONE"`, especifica o tipo de operação a ser realizada, que neste caso é a criação de uma nova zona. Em seguida, os campos `latitude` e `longitude` definem as coordenadas da localização da zona, representando respectivamente a posição vertical e horizontal no mapa. O campo `radius` determina o tamanho da zona a partir do ponto central especificado. A mensagem é finalizada com um caractere de dois pontos (:), que atua como terminador.

Vale ressaltar que o valor das variáveis latitude, longitude e raio são declarados como string, conforme a assinatura do método `private void CreateZone(String latitude, String longitude, String radius)` indica.

Ao receber essa mensagem, o `CreateZoneHandler` verifica se o valor de `radius` é negativo, indicando uma condição inválida para a criação de uma zona. Se o valor for positivo, o handler insere os valores de latitude, longitude e raio no banco de dados para definir a nova zona. A resposta gerada pelo servidor será uma mensagem de sucesso, se a operação for bem-sucedida, ou uma mensagem de erro caso a criação falhe. Essa estrutura garante a criação de zonas apenas com valores de raio válidos e permite uma resposta adequada ao cliente.

Request:

“CREATE-ZONE”;”x”;”y”;”radius”:

Response:

“MSG”;”Zone created with success”:

“MSG”;”Unable to register zone”:

“MSG”;”Server not available”:

```
String msgCreate = "\"CREATE-USER\";" + fname + "\";" + lname + "\";" + login + "\";" + password + "\";" + admin + "\":";
```

A mensagem `msgCreate` é formatada para enviar uma solicitação ao servidor, instruindo-o a criar um novo usuário no banco de dados. Ela utiliza o comando `CREATE-USER`, seguido por cinco campos que representam informações do usuário, todos separados por ";", com o ":" no final indicando o término da mensagem. O primeiro campo (string), `fname`, contém o primeiro nome do usuário, seguido de `lname` (string) para o sobrenome, `login` (string) para o identificador de acesso, `password` (string) para a senha e `admin` (booleano), que indica se o usuário possui ou não os privilégios de administrador.

Quando o servidor recebe a mensagem "CREATE-USER";"Joao";"Bernardes";"jb@usp.br";"12345";"true":, ele passa a solicitação ao CreateUserHandler, que abre uma conexão com o banco de dados e executa um comando INSERT para criar o usuário com os dados fornecidos. Se o processo for bem-sucedido, o servidor retorna "User created with success"; caso contrário, responde com "Unable to register user", registrando a mensagem correspondente no console para log.

Request:

“CREATE-USER”;”fname”;”lname”;”login”;”password”;”isAdmin”:

Response:

“MSG”;“User created with success”:

“MSG”;“Unable to register user”:

“MSG”;”Server not available”:

```
String msg = "\"DELETE-USER\";" + idUsers.get(rs[3]) + "\":";
```

A mensagem `String msg = "\"DELETE-USER\";" + idUsers.get(rs[3]) + "\":` é um comando que indica ao servidor a exclusão de um usuário específico. A mensagem é estruturada com dois campos principais separados por ponto e vírgula (;) e termina com dois pontos (:).

O primeiro campo, "DELETE-USER", identifica a ação a ser executada, que neste caso é deletar um usuário. A variável idUsers é um HashMap que armazena pares de valores onde a chave é uma String representando o email do usuário e o valor é um Integer representando o ID do usuário. O campo idUsers.get(rs[3]), dessa forma, contém o ID do usuário a ser excluído, obtido do objeto idUsers. Por exemplo, se o ID for 101, a mensagem completa fica "DELETE-USER";"101":.

Ao receber esta mensagem, o servidor aciona o DeleteUserHandler, que verifica se o usuário não é o administrador padrão (admin@usp.br). Se for permitido, ele executa um comando SQL para remover o usuário correspondente ao ID, retornando uma mensagem de sucesso ou erro, conforme o resultado da operação.

Request:

"DELETE-USER";"id_user":

Response:

"MSG";"User deleted with success":

"MSG";"Unable to delete user":

"MSG";"Server not available":

```
String message = "\"UPDATE-USER\";" + idUsers.get(getUser()) + ";" + getFname() +
"\";" + getLname() + ";" + getUser() + ";" + getPassword() + ";" + isAdmin + ";"
```

A mensagem acima solicita ao servidor a atualização de um usuário no banco de dados. Ela é composta por campos delimitados por ponto e vírgula (;), encerrada com dois pontos (:), e contém os seguintes valores: "UPDATE-USER", que indica o tipo de ação; o ID do usuário idUsers.get(getUser()); a string primeiro nome getFname(); a string do sobrenome getLname(); a string do login getUser(); a string da senha getPassword(); e o status booleano administrativo isAdmin.

O UpdateUserHandler processa essa mensagem, verificando se o usuário é o administrador padrão, caso em que não permite a atualização. Se o ID não corresponde ao administrador padrão, ele atualiza os campos fname, lname, password, e admin no banco de dados com os novos valores fornecidos na mensagem.

Request:

"UPDATE-USER";"id_user";"fname";"lname";"login";"password";"isAdmin":

Response:

MSG;"User updated with success":

MSG;"Unable to updated user":

MSG;"Server not available":

4 CÓDIGO

A função ``main`` do cliente está localizada na classe ``App.java``, no diretório ``\\WorkRedes\\Client2\\src\\com\\usp\\networks\\app``. Já a função ``main`` do servidor encontra-se na classe ``AppServer.java``, localizada no diretório ``\\WorkRedes\\Server\\src\\com\\usp\\networks\\app``.

Primeiramente, foram desenvolvidos dois projetos Maven para representar o cliente e o servidor. No lado do cliente, a estrutura foi dividida em quatro pacotes: `app`, `client`, `protocol` e `screens`. O pacote `app` contém o arquivo `App.java`, que representa o programa principal e chama as funcionalidades encapsuladas nas demais classes. Além disso, o pacote `client` contém o código responsável pelas funcionalidades dos usuários comuns e dos administradores, os quais utilizam instâncias das telas para realizar suas tarefas.

Adicionalmente, foi criada uma classe `Protocol` no lado do cliente com o objetivo de facilitar a decodificação e execução das mensagens específicas do cliente. Nesse contexto, o pacote `protocol` contém uma classe chamada `Protocol`, que utiliza o padrão de design Singleton para garantir a existência de uma única instância durante toda a execução. A classe `Protocol` é responsável por identificar o tipo de mensagem recebida, decodificá-la e executá-la por meio de uma estratégia específica para o tipo de mensagem. Para cada tipo de mensagem, foi desenvolvida uma estratégia de tratamento distinta, onde a execução apropriada é obtida através de uma tabela hash, com o tipo de mensagem como chave.

Além disso, a biblioteca Swing foi utilizada para desenvolver as interfaces gráficas da aplicação distribuída. O pacote `screens` é responsável por armazenar todas as telas da aplicação, incluindo telas específicas para usuários comuns e administradores. No caso dos usuários comuns, estão disponíveis as telas `XYScreen`, `NotificationsScreen`, `LoginScreen` e `ForgotKeyScreen`. Para os administradores, estão disponíveis as telas `LoginScreen`, `ForgotKeyScreen`, `NotificationsScreen`, `AdminScreen`, `UsersScreen`, `AssociationScreen`, `ZonesScreen` e `MessageScreen`. A descrição detalhada de cada tela encontra-se na tabela abaixo.

LoginScreen	Tela responsável por efetuar a operação de login, a qual tem os campos usuário e senha, um botão “esqueci senha” e um botão de entrar.
ForgotKeyScreen	Tela que modifica a senha do usuário caso tenha esquecido. Possui os campos “usuário”, “senha” e “confirmar senha”, um botão de enviar e um ícone de voltar a tela de login.
XYScreen	Tela destinada a usuário comum, onde se envia as coordenadas de longitude e latitude. Possui os campos “longitude” e “latitude”, um botão de enviar, um ícone de voltar a tela de login e um ícone que leva à página de notificações.
NotificationsScreen	Tela que mostra todas as notificações recebidas pelo usuário, tanto infração quanto mensagem de administrador. No caso de administradores, apenas recebem avisos sobre infrações cometidas pelos usuários. Contém uma lista de notificações, um botão “excluir” para apagar notificações selecionadas e um ícone para voltar à página anterior.
UsersScreen	Tela destinada a administradores, onde se visualiza os usuários já criados e possui um formulário para criar ou atualizar os usuários. Há como excluir um usuário, através do botão “apagar”.

ZonesScreen	Tela destinada a administradores, onde se visualiza as zonas criadas por meio de uma lista e possui um formulário para criar ou atualizar zonas. Há como excluir zonas, através do botão “apagar”.
AssociationScreen	Tela destinada a administradores, onde se visualiza as associações entre zonas e usuários já criadas e possui os campos “zona” e “usuário” para criar novas associações. Há como excluir associações, via “apagar”.
MessageScreen	Tela destinada a administradores, onde possui um campo para selecionar o destinatário, um campo para digitar a mensagem e um botão de enviar a mensagem.
AdminScreen	Tela inicial de administradores, onde possui botões para encaminhá-los para as telas reservadas à administradores.

Client -> Classe abstrata que implementa operações comuns entre usuários e admins.

User -> Classe que representa um usuário comum.

Admin -> Classe que representa um administrador.

Server -> Classe que representa um servidor.

App -> Classe que contém a void main que chama a tela de login e , com base no resultado, chama-se um User ou Admin.

IPServer.txt -> Arquivo texto que acompanha o app para saber o endereço do servidor para fazer requisições.

myDB.db -> banco de dados que acompanha o executável do servidor

O projeto do servidor é responsável por processar as requisições dos clientes. No lado do servidor, todas as operações são exibidas no terminal, sem interface gráfica. O banco de dados SQLite3 foi utilizado para armazenar informações sobre usuários, zonas, notificações e associações entre usuários e zonas. O servidor está organizado em três pacotes principais: server, items e app.

Primeiramente, o pacote app contém a classe AppServer, que gerencia a execução do servidor, invocando métodos encapsulados nos outros pacotes. O pacote items reúne todas as estratégias de manipulação de mensagens e inclui a classe Database, responsável pela comunicação com o banco de dados. Por exemplo, o arquivo ListNotifyHandler.java contém o algoritmo para processar requisições do tipo LIST-NOTIFY:. Além disso, o pacote server possui uma classe principal que implementa o padrão de design Singleton para a execução do servidor e inclui o arquivo Protocol.java, cuja função é decodificar e processar as mensagens recebidas dos clientes. Abaixo, uma imagem ilustra a execução do servidor.



```
C:\Users\Usuario\Documents\USP\4_SEMESTRE\REDES\EP1\WorkRedes\Server>mvn exec:java -Dexec.mainClass="com.usp.networks.app.AppServer"
[INFO] Scanning for projects...
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-metadata.xml
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/maven-metadata.xml
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/plugins/maven-metadata.xml (14 kB at 43 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/maven-metadata.xml (21 kB at 62 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/maven-metadata.xml
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/mojo/exec-maven-plugin/maven-metadata.xml (958 B at 40 kB/s)
[INFO]
[INFO] -----< Redes:EP >-----
[INFO] Building MaganerZones 0.0.1-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- exec:3.4.1:java (default-cli) @ EP ---
Server started and waiting for connections...
```

Em suma, o projeto foi dividido em cliente e servidor, onde as partes se comunicam através de sockets TCP, o qual a cada requisição deve-se estabelecer a conexão novamente. Foi utilizado o Maven para a criação da aplicação distribuída, visto que tal ferramenta facilitou a extração das dependências para comunicar o java com o banco de dados, além de facilitar a execução via terminal.

5 TESTES E RESULTADOS

Nesta seção, discutiremos os desafios que encontramos durante o desenvolvimento da aplicação, ilustrando como diferentes tipos de entrada impactam o seu funcionamento. Também descreveremos o comportamento do servidor em resposta a cada uma das situações problemáticas apresentadas a seguir.

5.1 OBSERVAÇÕES

Pode ocorrer um bug onde o administrador tenta se auto excluir. Ao fazer isso, ele não é impedido de acessar a tela de mensagens e enviar uma mensagem, pois sua sessão ainda não foi finalizada (ele ainda não foi deslogado ou foi redirecionado para a tela de login).

Para excluir uma zona ou um usuário, é necessário primeiro desassociar as zonas dos usuários. Em outras palavras, é preciso remover essas associações entre usuários e zonas antes de excluir qualquer usuário ou zona.

Além disso, as telas de “Users”, “Zones” e “Notifications” não possuem barra de rolagem. Portanto, se novos usuários, zonas ou notificações forem adicionados além do limite exibido, é necessário excluir um item anterior no topo da lista para que o novo conteúdo (usuário, zona ou notificação) seja visível.

Para concluir, é importante destacar o problema que enfrentamos relacionado à comunicação TCP. Nossa aplicação não estava encerrando corretamente as conexões TCP durante o envio das mensagens. Com o passar do tempo, isso resultou no acúmulo de mensagens, o que consumia uma quantidade significativa de recursos de memória e, eventualmente, levou ao travamento da aplicação. Ademais, a falha no fechamento adequado das conexões TCP, juntamente com a não liberação de recursos como sockets, streams e buffers, pode ter contribuído para vazamentos de memória, agravando ainda mais a instabilidade do sistema.

Para solucionar esse problema, implementamos um bloco ‘finally’, que é sempre executado após a conclusão do bloco try-catch que gerencia o envio das mensagens. Essa abordagem assegura o encerramento apropriado dos recursos utilizados, como `PrintWriter` (out), `BufferedReader` (in) e o socket (client), mesmo que ainda estejam abertos. Após testar essa solução, constatamos que, aparentemente, o problema foi resolvido e não enfrentamos mais travamentos na aplicação relacionados à falta de encerramento adequado da conexão TCP.

5.2 RESULTADOS

Para facilitar o uso da aplicação e garantir um preenchimento correto dos campos, decidimos fornecer alguns exemplos de preenchimento em cada tela.

5.2.1 Administradores e Usuários

O sistema já possui alguns usuários e administradores pré-cadastrados para facilitar o uso da aplicação e auxiliar nos testes.

- Lista de administradores:

Login	Senha
joao_ferreira@usp.br	12345
aaron@usp.br	12345
paulo@usp.br	12345
prof_admin@usp.br	12345

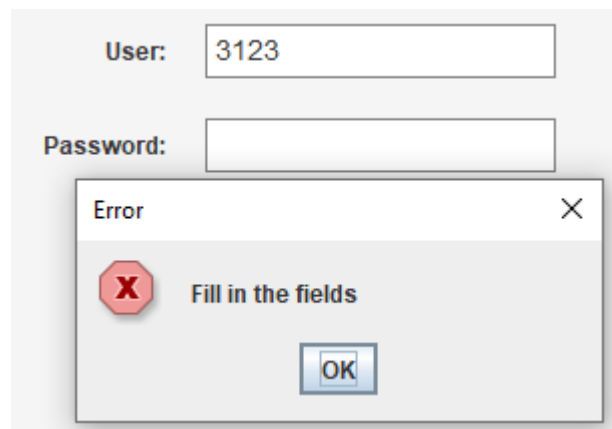
- Lista de usuários:

Login	Senha
rogerio_nobre@usp.br	323232mm
dilma_roussef@usp.br	323232mm
marcos@usp.br	12345
luiz@usp.br	12345

prof_client@usp.br	12345
--------------------	-------

5.2.2 Login

Ao tentar fazer login, se o usuário deixar qualquer um dos campos (Usuário e Senha) em branco, seja preenchendo apenas um campo ou nenhum deles, uma mensagem de erro será exibida na tela, conforme mostrado na imagem abaixo:

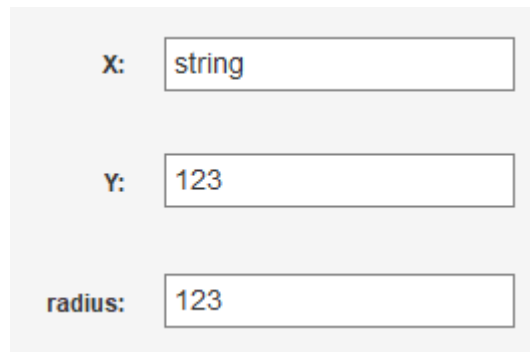


5.2.3 Zones

O cadastro de uma zona com raio negativo não é permitido. Caso essa situação ocorra, o servidor exibirá a seguinte mensagem no terminal:

```
Connected Client: 26.7.210.20
There is no zone with a negative radius
```

Além disso, uma zona só é cadastrada se todos os campos (x, y e raio) estiverem preenchidos. Caso o usuário tente cadastrar uma zona inserindo texto (string) em qualquer um dos campos — seja em x, y ou raio, ou em qualquer combinação entre texto e número inteiro nesses campos — um erro será registrado no terminal, conforme mostrado na imagem abaixo:



A form with three input fields. The first field is labeled 'X:' and contains the text 'string'. The second field is labeled 'Y:' and contains the number '123'. The third field is labeled 'radius:' and contains the number '123'.

O terminal do servidor irá exibir o seguinte erro:

```
Connected Client: 26.7.210.20
Modo WAL ativado.
Error in SQL: [SQLITE_ERROR] SQL error or missing database (no such column: string)
Unable to register zone
```

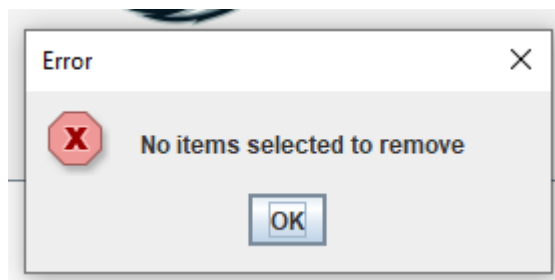
5.2.4 Messages

Inicialmente, a tela de "Messages" parece estar funcionando corretamente, sem erros aparentes. Contudo, é importante observar que apenas administradores podem enviar mensagens para usuários. Além disso, para que um usuário possa ser excluído do sistema, ele deve primeiro remover todas as suas mensagens.

5.2.5 Notifications

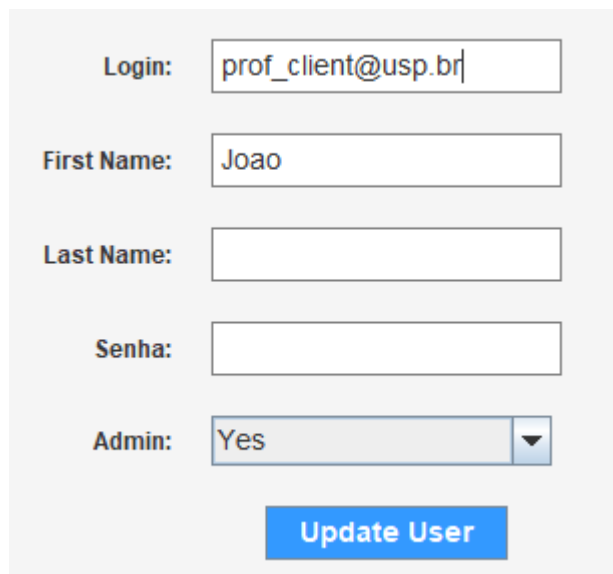
Assim como na tela de "Mensagens," a tela de "Notificações" também aparenta estar livre de erros. Nessa tela, são exibidas mensagens relacionadas a algum usuário que invadiu alguma zona, bem como quaisquer comunicados enviados pelo administrador ao usuário.

Se você tentar excluir uma mensagem na tela de "Notificações" sem que haja nenhuma notificação disponível, será exibida a seguinte mensagem de erro:



5.2.6 Update User

Nesta tela, você pode atualizar as informações do usuário, como o nome, sobrenome, senha e até o status da conta (administrador ou não), com exceção do campo de login, que permanece inalterado. É importante destacar que, se algum campo não for preenchido, a atualização não será realizada, conforme mostrado na imagem abaixo:



Form fields and values:

- Login: prof_client@usp.br
- First Name: Joao
- Last Name:
- Senha:
- Admin: Yes

Update User

Ao clicar em “Update User”, nenhuma alteração será realizada no usuário com o login “[prof_client@usp.br](#)” e os campos serão limpos e exibidos em branco.

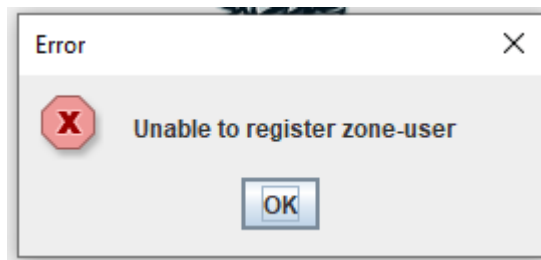
5.2.7 Association

Nesta tela, o administrador pode associar usuários a zonas. Caso o administrador deseje remover uma associação de uma zona que já foi excluída, o sistema permite essa ação e exibe a seguinte mensagem no terminal:

```
Connected Client: 26.7.210.20
Modo WAL ativado.
Association Zone-User deleted with success
Connected Client: 26.7.210.20
Modo WAL ativado.
Association Zone-User deleted with success
```

Observe que o sistema permitiu duas exclusões consecutivas sem apresentar nenhum problema.

Agora, ao tentar associar uma zona que já foi previamente associada (ou seja, ao tentar associar o mesmo usuário a uma zona pela segunda vez), ocorre o seguinte erro:



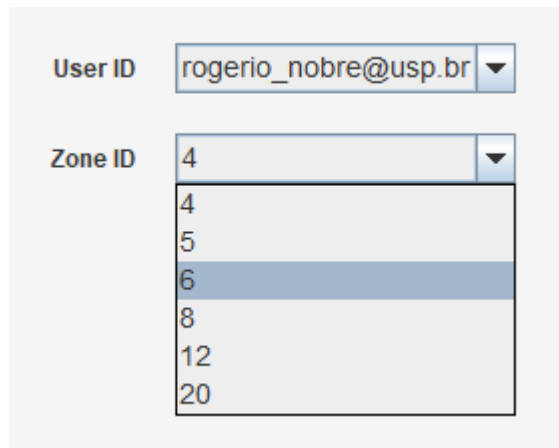
E no terminal do servidor aparece a seguinte mensagem:

```
Connected Client: 26.7.210.20
Modo WAL ativado.
Error in SQL: [SQLITE_CONSTRAINT_PRIMARYKEY] A PRIMARY KEY constraint failed (UNIQUE constraint failed: UserZoneMapping.
user_id, UserZoneMapping.zone_id)
Unable to register association zone-user
```

Na tela “Association”, ao abri-la, note que os campos ‘User ID’ e ‘Zone ID’ já estão preenchidos automaticamente, conforme ilustrado na imagem abaixo:

Entretanto, para que o administrador possa associar ou desassociar uma zona a um usuário, é imprescindível que os campos 'User ID' e 'Zone ID' sejam selecionados novamente, utilizando o botão indicado pela seta, conforme ilustrado na figura abaixo:

Selecione um usuário, como a imagem acima mostra.



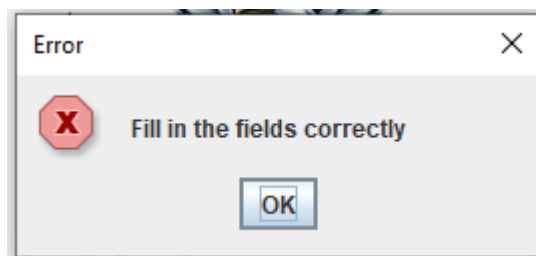
Selecione também uma zona, conforme a imagem acima. Dessa forma, é feita a associação entre usuário e zona.

A razão para esse comportamento é que o JFrame (Java JFrame) exibe os campos de entrada como se já estivessem preenchidos, criando a ilusão de que contêm informações. No entanto, na realidade, esses campos estão vazios, configurando um efeito visual que pode ser interpretado como um ‘bug’.

5.2.8 User

Quando o usuário do programa digitar o login e senha de uma conta que pertence a um “User”, isto é, usuário cadastrado no sistema que não seja administrador, a tela de “User” é aberta.

Nesta tela, o usuário pode inserir coordenadas específicas (latitude e longitude), e o sistema verificará se essas coordenadas violam uma ou mais zonas associadas a esse usuário. É importante notar que, caso os campos de latitude e longitude estejam vazios ou apenas um deles seja preenchido, uma mensagem será exibida, conforme descrito abaixo:



Agora, caso o usuário digite uma string (ou até mesmo símbolos como ‘*’ ou ‘-’) nos campos latitude e longitude, conforme é mostrado na imagem abaixo:



A screenshot of a web form with a light gray background. It contains two input fields: the first is labeled 'Latitude:' and contains the text 'abcde'; the second is labeled 'Longitude:' and contains the text 'asdsadsdsaad'. Below these fields is a blue button with the text 'Send Position' in white.

A seguinte mensagem é exibida no terminal do servidor:

```
Connected Client: 26.7.210.20
Modo WAL ativado.
Modo WAL ativado.
Modo WAL ativado.
Exception in thread "Thread-51" java.lang.NumberFormatException: For input string: "abcde"
    at java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2054)
    at java.base/jdk.internal.math.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
    at java.base/java.lang.Double.parseDouble(Double.java:944)
    at com.usp.networks.items.XYHandler.execute(XYHandler.java:22)
    at com.usp.networks.server.Protocol.execute(Protocol.java:62)
    at com.usp.networks.server.Server.execute(Server.java:20)
    at com.usp.networks.app.ThreadClient.run(ThreadClient.java:23)
    at java.base/java.lang.Thread.run(Thread.java:1575)
```

6 FONTES

Ao longo do desenvolvimento deste trabalho, utilizamos o Chat GPT como uma ferramenta importante para gerar ideias, aprender novos conteúdos e solucionar problemas. A inteligência artificial nos forneceu suporte em diversas áreas, desde a compreensão de conceitos até a implementação de soluções práticas para questões que surgiram durante o processo. As sugestões, explicações e exemplos fornecidos pelo programa nos ajudaram a acelerar o aprendizado e a resolver desafios de maneira mais eficiente. Portanto, nossas fontes de consulta e referências baseiam-se nas informações presentes nas bases de dados utilizadas pelo Chat GPT.