Laboratório 4 - Parte 1

Projeto de aplicação de “bate-papo distribuído ERT”

Atividade 1: Projeto de interface do usuário

O projeto de interface de usuário foi elaborado focando-se em três telas principais: “autenticação”, “inicial” e “chat”. A tela de autenticação é a primeira que aparecerá para o usuário, onde ele pode logar-se, cadastrar-se - caso ainda não tenha cadastro - ou sair da aplicação. Além disso, é possível deletar seu cadastro da aplicação através de seu login e senha. Após o login do usuário no sistema, ele será redirecionado para a tela inicial da aplicação onde será mostrado o menu inicial. Através do menu inicial é possível escolher um chat para iniciar uma conversa, alterar status do usuário para inativo ou ativo, mostrar quais usuários estão ativos, mostrar as últimas 5 notificações recebidas ou realizar o logout da aplicação. Caso o usuário em questão receba alguma mensagem sem estar com o chat do remetente aberto, uma mensagem de notificação aparecerá acima do menu inicial.

Ao selecionar um chat para abrir, o usuário irá se deparar com a tela de chat. Essa tela terá duas seções: o menu de chat e o chat em si. O menu de chat oferece as seguintes opções de ação: fechar chat atual e limpar mensagens do chat atual. Da mesma forma que o menu da tela inicial, caso um outro usuário que não o do chat aberto envie uma mensagem, uma notificação aparecerá no final da tela avisando que um chat com ele foi aberto. No chat em si serão mostradas as mensagens enviadas pelo remetente em questão e o usuário também poderá escrever e mensagem que deseja enviar.

Segue abaixo um resumo com todas as funcionalidades e ações para cada tela mencionada:

Funcionalidades

- Cadastrar na aplicação
- Deletar cadastro na aplicação
- Logar na aplicação
- Visualizar usuários ativos da aplicação, incluindo o do próprio usuário
- Alterar o status dos usuário da aplicação para ativo ou inativo
- Enviar mensagem para usuários ativos da aplicação
- Visualizar mensagens de usuários da aplicação, apresentando as últimas 20 mensagens
- Apagar mensagens com determinado usuário
- Mostrar notificações (convites de chat, mensagens de sucesso/falha, etc)
- Fazer o logout da aplicação
- Sair da aplicação

Interface de usuário

Tela de autenticação

Menu de autenticação

- Logar na aplicação
- Criar cadastro do usuário
- Deletar cadastro de usuário
- Sair da aplicação

Tela inicial da aplicação

Menu principal do usuário

- Mostrar usuários que estão ativos
- Alterar status (ativo/inativo)
- Logout da aplicação
- Abrir chat com usuário X
- Mostrar as 5 últimas notificações

Tela de chat

- Menu do usuário dentro do chat X
    - Fechar chat com usuário X / Voltar para menu principal
    - Apagar mensagens com o usuário X
    - Mostrar notificações
- Chat com o usuário X
    - Mostrar as últimas mensagens enviadas e recebidas por X
    - Enviar mensagem para X

Atividade 2: Projeto da arquitetura de software da aplicação

A arquitetura de software escolhida para este projeto foi a em camada, onde cada uma delas é melhor descrita nos tópicos abaixo:

- **Camada de interface com o usuário:** esta camada recebe todas as entradas de teclado do usuário, seja comandos para navegar na aplicação, quanto às mensagens a serem enviadas para outros usuários. Além disso, ela é responsável por gerir a tela como um todo, apresentando todos os elementos visuais da aplicação: menus, mensagens, chats, notificações, etc. Esta camada receberá da camada de processamento somente as mensagens enviadas por outros usuários, ou o resultado de quaisquer processamentos necessários advindos dos comandos dados pelo usuário. Ou seja, toda “roupagem” visual será feita na camada de interface.
- **Camada de processamento:** esta camada é responsável por fazer a autenticação, cadastro, deleção do usuário e navegação pelos menus. Além disso, informa o status dos usuários cadastrados bem como trata e envia as mensagens trocadas a nível usuário-usuário e usuário-servidor.

- **Camada de acesso aos dados:** esta camada é aquela que fará acesso à base de dados, onde terá informações de cadastro dos usuários e seus status. Esta camada será acessada pela camada de processamento e retornará para esta última os dados pedidos.

Atividade 3: Projeto da arquitetura de sistema da aplicação

Para a arquitetura de sistema da aplicação, foi escolhido a arquitetura híbrida (mistura entre arquiteturas cliente/servidor e peer-to-peer), onde o uso que cada um dos tipos de arquiteturas que a compõem possuem objetivos diferentes, enunciados a seguir:

- **Cliente/servidor:** esta será utilizada pelo cliente para obter informações dos demais usuários cadastrados e disponíveis na plataforma, além de possibilitar que os usuários possam realizar autenticação e gerenciamento da sua conta.

- **Peer-to-peer (P2P):** esta abordagem será utilizada para conexão direta entre os usuários, de modo que os mesmos possam enviar e receber mensagens.

Analisando do ponto de vista cliente/servidor existem dois tipos de papéis, onde cada um deles implementa um conjunto de camadas, como listadas abaixo:

- **Lado cliente:** irá implementar a camada de interface com o usuário e parte da camada de processamento. Além disso, a aplicação terá três fluxos de execução concorrentes. O primeiro terá como objetivo a comunicação exclusiva com o servidor. Já o segundo irá implementar o lado passivo do cliente que permitirá receber conexões de outros clientes. Por fim, o terceiro será responsável pelo lado ativo do cliente, permitindo que o mesmo possa iniciar comunicação com outros usuários.

- **Lado servidor:** irá implementar a interface com o gerente do servidor, parte da camada de processamento e a camada de acesso aos dados. O servidor neste caso será concorrente, para que possa atender mais de um cliente paralelamente. Além disso, o gerente pode enviar comandos para o servidor, sendo atendido em paralelo aos clientes.

Contudo, deve ser considerado que para as funcionalidades executadas em modo P2P cada cliente (usuários) poderá assumir o papel de ativo e/ou passivo, dependendo de quem inicie a troca de mensagens.

Atividade 4: Projeto de protocolo da camada de aplicação

Para este projeto, será utilizado o TCP como protocolo da camada de transporte. Sendo assim, haverá a necessidade de conexão para a transmissão de mensagens, tanto entre clientes quanto cliente-servidor. As mensagens trocadas estão alocadas em um dos conjuntos abaixo, cada um com os campos listados. Todas as mensagens são enviadas no formato JSON. Todas as requisições possíveis com as suas estruturas estão melhor detalhadas no Anexo I.

Estrutura das mensagens

- **Requisições**
  - **“method” (String):** define qual é o método a ser acessado. Este campo pode ser:
    - **Métodos Cliente-Servidor**
      - “createAccount”: método para criar uma conta para o usuário no sistema. Deve ser enviado o “userName” e o “password” como campos para o nickname e senha no campo “data”.
      - “deleteAccount”: método para deletar uma conta de usuário no sistema. Deve ser enviado o “userName” e o “password” como campos para o nickname e senha no campo “data”.
      - “authAccount”: método para autenticar o usuário no sistema. Deve ser enviado o “userName”, “password” e a “porta” como campos para o nickname e senha no campo “data”.
      - “getUsers”: método para buscar todos os usuários que estão cadastrados no sistema. “None” deve ser passado no campo “data”.
      - “getMyStatus”: método para retornar qual é o status (ativo ou inativo) do usuário em questão. Deve ser enviado o “userName” e o “password” como campos para o nickname e senha no campo “data”.
      - “setMyStatus”: método para modificar o status (ativo ou inativo) do usuário no sistema. Deve ser enviado o “userName” e o “password” e o valor do campo “status” **(inteiro)** dentro do dicionário “data”.
      - “logout”: método para desconectar o usuário do sistema. No campo “data” deve ser enviado o “userName” e o “password” do usuário remetente a ser desconectado ou inativado.
    - **Métodos Usuário-Usuário**
      - "sendMessage": método para indicar o envio de mensagens. No campo “data” terá o subcampo “message” guardará a mensagem a ser enviada e também o “userName” do usuário que enviou a mensagem.

- **“data” (Dicionário):** define os dados a serem passados para esse método caso tenha. Cada campo do dicionário pode ser de valor **String** ou **Inteiro**. Vale ressaltar que o campo “data” pode assumir **None**.

- **Resposta de Sucesso**
  - **“status” (String)**: “success”, indicando que tudo ocorreu bem.
  - **“method” (String)**: o nome do método pedido na mensagem de requisição (as possibilidades são as mesmas das mensagens de requisição).
  - **“data” (Dicionário)**: guarda os dados retornados pelo método em questão caso tenha, do contrário pode retornar que o método foi executado com sucesso. Cada campo do dicionário pode ser de valor **String** ou **Inteiro**.
- **Resposta de Erro**
  - **“status” (String)**: “error”, indicando que houve um problema.
  - **“method” (String)**: o nome do método pedido na mensagem de requisição (as possibilidades são as mesmas das mensagens de requisição).
  - **“data” (Dicionário)**: guarda os dados retornados pelo método em questão caso tenha. Neste caso o dado seria a mensagem de erro gerada.

Regras

Ao acessar a aplicação pela primeira vez, será necessário que o usuário realize o cadastro no servidor. Para isso, será preciso que o cliente envie uma **mensagem de requisição** ao servidor **(createAccount)** passando o nome e a senha desejados, para a qual o servidor irá responder com uma mensagem de sucesso ou erro. Em seguida, o usuário deve realizar o login no servidor, com o cliente enviando uma **mensagem de requisição (authAccount)** passando o usuário e senha criados anteriormente, além de sua porta, e o servidor deverá responder aprovando ou negando o acesso. Vale ressaltar que após a realização do cadastro o servidor irá salvar essas informações, de modo que não haja a necessidade de novos cadastros em acessos posteriores. Além disso, caso o mesmo deseje descadastrar-se poderá enviar uma **mensagem de requisição (deleteAccount)** informando suas credenciais. Por fim, após estar conectado ao servidor, o cliente estará apto para realizar as funcionalidades do bate papo e iniciar conversas com outros clientes.

Após o usuário estar logado na aplicação, o mesmo poderá requisitar ao servidor a lista de usuários cadastrados. Para isso, o cliente envia uma **mensagem de requisição** ao servidor (**getUsers**). Ao fim do processamento dessa requisição, o servidor responde ao cliente com uma mensagem contendo a listagem dos usuários cadastrados no sistema ou então uma mensagem de erro, caso ocorra algum problema.

O usuário também pode fazer a mudança do seu status ao seu objeto cliente associado enviar uma **mensagem de requisição** ao servidor (**setMyStatus**). O servidor, então, responde com uma mensagem informando o atual estado do usuário ou uma mensagem de erro, caso ocorra algum problema.

Caso o usuário queira conversar com alguém, é necessário que ele digite o comando para abrir o chat do parceiro em questão. Neste momento, para que as mensagens digitadas do usuário sejam entregues, é necessário que o cliente envie uma **mensagem de requisição**

(sendMessage) com o conteúdo que quer passar ao destinatário e o seu “userName”. Durante toda a conversa, serão trocadas entre remetente e destinatário **mensagens de requisições do tipo “sendMessage”**, até que um dos usuários pare de enviar mensagens. Para que um usuário A envie uma mensagem para o usuário B, não é necessário que o chat entre eles esteja aberto em ambos os lados ao mesmo tempo.

Quando o usuário desejar se desconectar do sistema, o cliente enviará uma **mensagem de requisição do tipo “logout”** para o servidor e, quando for confirmada a sua desconexão, os clientes aos quais mantinham uma conexão com ele perderão esta conexão. É passado como parâmetro o seu próprio user name. Uma mensagem de requisição do tipo “logout” também é enviada aos usuários com conexão aberta quando o agente deseja mudar seu status de “ativo” para “inativo”. Caso o logout seja realizado com sucesso, uma **mensagem de resposta de sucesso** será retornada; caso contrário, uma **mensagem de resposta de erro** virá como resposta.

ANEXO I

Mensagens Cliente-Servidor

Requisição	{
	"method":"authAccount",
	"data":{
	"userName":"fulano",
	"password":“123456”,
	“port”: 5002
	}
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"authAccount",
	"data":{
	"message":"logado com sucesso"
	}
	}

Resposta de Erro	{
	"status":"error",
	"method":"authAccount",
	"data":{
	"message":"<motivo do erro>"
	}
	}

Requisição	{
	"method":"createAccount/deleteAccount",
	"data":{
	"userName":"fulano",
	"password":“123456”
	}
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"createAccount/deleteAccount",
	"data":{
	"message":"criado/deletado"
	}
	}

Resposta de Erro	{
	"status":"error",
	"method":"createAccount/deleteAccount",
	"data":{
	"message":"<motivo do erro>"
	}
	}

Requisição	{
	"method":"logout",
	"data":{
	"userName":"fulano",
	"password":“123456”
	}
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"logout",
	"data":{
	"message":"desconectado com sucesso"
	}
	}

Resposta de Erro	{
	"status":"error",
	"method":"logout",
	"data":{
	"message":"<motivo do erro>"
	}
	}

Requisição	{
	"method":"getUsers",
	"data":"None"
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"getUsers",
	"data":[
	{
	"userName":"Eduardo",
	"ip":“192.168.0.1”,
	"port":“5569”,
	"status":1/0
	},
	{
	"userName":"Tainá",
	"ip":“192.167.0.2”,
	"port":“5488”,
	"status":1/0/-1
	}
	]
	}

Resposta de Erro	{
	"status":"error",
	"method":"getUsers",
	“data”:{
	"message":"motivo do erro"
	}
	}

Requisição	{
	"method":"getMyStatus",
	"Data": {
	"userName":"fulano"
	“password”:”123456”
	}
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"getMyStatus",
	"data":{
	"message":1/0
	}
	}

Resposta de Erro	{ "status":"error", "method":"getMyStatus",

	“data”:{
	"message":"motivo do erro"
	}
	}

Requisição	{
	"method":"setMyStatus",
	"data":{
	"userName":"fulano"
	“password”:”123456”
	"status":1/0 (1 é ativo e 0 inativo)
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"setMyStatus",
	"data":{
	"message": “Status 1/0 definido com sucesso.”
	}
	}

Resposta de Erro	{
	"status":"error",
	"method":”setMyStatus",
	“data”: {
	"message":"motivo do erro"
	}
	}

Mensagens Usuário-Usuário

Requisição	{
	"method":"sendMessage",
	"data":{
	"userName":"fulano",
	"message":"Oie fulano",
	}
	}

Resposta de Sucesso	{
	"status":"success",
	"method":"sendMessage",
	"data":"None"
	}

Resposta de Erro	{
	"status":"error",
	"method":"sendMessage",
	"data":"None"
	}