Projeto de "bate-papo distribuído"

Feito por Eduardo Tavares Leão, 117063913.

Introdução

O objetivo deste Laboratório é **projetar uma aplicação de "bate-papo distribuído"** para aplicar os conceitos estudados até aqui. Na próxima semana, o exercício de laboratório será implementar e avaliar essa aplicação.

Para o contexto deste exercício, um **bate-papo distribuído** e uma aplicação distribuída que permite que **pares de usuários** remotos troquem mensagens de texto.

Requisistos gerais da aplicação

A solução implementada deverá permitir que o usuário, ao entrar na aplicação, sinalize que está pronto para receber mensagens e tenha acesso à relação de outros usuários que também estão ativos. Daí, o usuário poderá enviar mensagens para qualquer outro usuário ativo e, ao mesmo tempo, receber e responder mensagens de outros usuários. Quando desejar sair da aplicação, o usuário deverá sinalizar que não mais estará ativo.

Não é necessário criar janelas distintas para cada conversa. Pode-se usar, por exemplo, um rótulo no início de cada mensagem para indicar o remetente ou o destinatário da mensagem. Dessa forma, todas as mensagens trocadas podem ser exibidas na mesma janela.

Atividade 1

Objetivo: Projetar a interface de usuário da aplicação.

Roteiro: Considere as funcionalidades desejadas listadas abaixo e elabore o menu da aplicação.

1. Indicar que o usuário está ativo (apto a receber mensagens).
2. Indicar que o usuário está inativo (não está apto a receber mensagens).
3. Visualizar os usuários que estão ativos.
4. Iniciar e manter uma conversa com um usuário que está ativo.
5. Visualizar as mensagens recebidas de outros usuários.
6. Encerrar a aplicação.

Menu

- Para indicar que o usuário está ativo haverá um botão que alterará o `status` do usuário entre `online` e `offline`.
- No menu também haverá uma parte com a lista de todos os usuários ativos no serviço, além de um `(conectado)` ao lado do nome caso eles estejam em uma conversa.

- Ao clicar no nome de um usuário ativo, um botão será disponibliziado para realizar conexão ou encerrar conexão.
- Haverá outro espaço mostrando todas as conexões atuais do usuário.
- Para enviar uma mensagem, o usuário precisará selecionar o destinatário na lista de conexões atuais.
- Todas as mensagens trocadas serão vistas na tela principal.
- Para encerrar a aplicação basta usar o botão padrão da janela de encerramento.
- A visualização se dará da seguinte forma:

  ```
  (Username:) <Corpo da mensagem>
  ```

  Ou seja, o nome do usuário que envia a mensagem seguido da própria mensagem.

Atividade 2

Objetivo: Projetar a arquitetura de software da aplicação.

Roteiro: Considerando as funcionalidades listadas acima, projete os componentes da aplicação (funcionalidades e interfaces). Use os estilos arquiteturais estudados como
ponto de partida.

Arquitetura de software

A arquitetura escolhida como base foi a de arquitetura em camadas. Serão desenvolvidas as seguintes camadas:

- **Interface do usuário:** será responsável por montar e apresentar o Menu e receber as requisições do usuário para enviá-las à camada abaixo.
- **Camada de processamento / aplicação:** responsável pela manutenção de conexões ativas e processar as requisições do usuário.
- **Camada de dados:** responsável por armazenar a lista de clientes ativos no sistema e a lista de conexões ativas de cada cliente.

Atividade 3

Objetivo: Projetar a arquitetura de sistema da aplicação.

Roteiro: Escolha uma das arquiteturas de sistema estudadas (cliente-servidor, par-a-par, híbrida) e descreva como a arquitetura de software projetada na Atividade 2 será instanciada em uma arquitetura de sistema, definindo:

1. como os componentes de software serão agrupados ou particionados;
2. onde os componentes de software serão implantados;
3. como os componentes de software deverão interagir.

Arquitetura de sistema

Será um modelo híbrido com cliente/servidor em que o papel do servidor é manter um registro de conexões ativas e de usuários ativos e a interação entre os usuários será P2P.

Como dito, teremos duas aplicações: cliente e servidor.
A parte de **interface do usuário** será totalmente concentrada nas máquinas dos clientes. Já a camada de processamento será dividida entre cliente e servidor da seguinte forma:

- **Cliente:** responsável por estabelecer conexões com o servidor e com outros clientes, processar as requisições do usuário e realizar a troca de mensagens P2P. Aqui será feito a manutenção da lista de conexões com outros usuários.
- **Servidor:** responsável por receber conexões de todos os clientes e fazer uma manutenção da lista de clientes ativos.

Atividade 4

Objetivo: Projetar o protocolo de camada de aplicação da aplicação.

Roteiro: Projete um protocolo de camada de aplicação, considerando as decisoes tomadas nas Atividades 2 e 3, definindo:

1. protocolo de camada de transporte;
2. tipos de mensagens trocadas (ex., requisição, resposta);
3. sintaxe/formato de cada tipo de mensagem (campos da mensagem com seus tamanhos e tipos de dados);
4. regras que determinam quando e como um processo envia/responde mensagens.

Pode-se optar por usar algum formato de troca de mensagem (ex., JSON).

Protocolo camada de aplicação

1. O protocolo da camada de transporte será o TCP.
2. O tipo das mensagens trocadas será o de requisição/resposta no relacionamento entre cliente/servidor. Já no relacionamento P2P será aproveitada a conexão TCP para permitir um fluxo contínuo de troca de mensagens. A verificação será feita indicando o tamanho da mensagem no cabeçalho da mesma.
3. O formato de troca de mensagens usado será JSON do seguinte formato para requisições:

```
{
    "type": type,
    "source" : source,
    "destiny" : destiny,
    "command" : command,
    "message": message,
}
```

A tabela a seguir mostrará as opções de cada argumento:

Argumento	Valor	Descrição
type	`"request"`	Sinal de que a mensagem é uma requisição. Possível no cenário cliente/servidor
type	`"response"`	Sinal de que a mensagem é uma resposta. Possível no cenário cliente/servidor
type	`"p2p"`	Sinal de que a mensagem é enviada para em cenários P2P.
source	`string`	Endereço de remetente da mensagem. Ex: `192.168.0.1:8000`
destiny	`string`	Endereço de destinatário da mensagem. Ex: `192.168.0.1:9000`
command	`open`	Mensagem para que o servidor incremente o cliente na lista de usuários ativos
command	`close`	Mensagem para que o servidor remova o cliente da lista de usuários ativos
command	`show`	Mensagem para que o servidor mostre a lista de usuários ativos
command	`update`	Mensagem para atualizar a lista de usuários ativos na interface
command	`connect`	Mensagem para estabelecer conexão P2P. Possível com `type` sendo `p2p`
command	`send`	Sinal de que há uma mensagem a ser enviada. Possível com `type` sendo `p2p`
command	`end`	Requerimento para fim de conexão P2P. Possível com `type` sendo `p2p`.
command	`finish`	Requerimento para término de aplicação.
body	`string` / `dict`	Corpo da mensagem. (*Opcional*)

4. De acordo com as funcionalidades do sistema, temos os seguintes cenários:
    - Para indicar que o usuário está ativo basta que ele envie o comando `open` para o servidor principal, que irá **incrementar** o usuário na lista de dicionários de usuários ativos, utilizando o

nome inserido pelo usuário na camada de interface e que será passado em `body`. Como exemplo de lista, temos:

```
[
    {
        "user": "Joao",
        "address:": "192.168.0.1:8010"
    },
    {
        "user": "Maria",
        "address": "192.168.0.1:8020"
    },
    {
        "user": "Pedro",
        "address": "192.168.0.1:8030"
    },
    ...
]
```

- Para indicar que o usuário está inativo basta que ele envie o comando `close` para o servidor principal, que irá **retirar** o endereço desse usuário da lista de dicionários de usuários ativos.
- Para visualizar os usuários ativos a interface enviará o comando `show` e o servidor responderá com a lista de dicionários de endereços ativos. Em caso de mudança na lista, o próprio servidor enviará o comando `update` seguido da lista.
- Para iniciar uma conversa com outro usuário será enviado o comando `connect` seguido da mensagem em `body`.
- Para encerrar uma conversa com outro usuário, será utilizado o comando `end`.
- Para encerrar a aplicação temos o comando `finish`. Assim, o cliente será removido da lista de usuários ativos, caso esteja nela e encerrará o programa. Caso haja conexões ativas, ele receberá a mensagem:

```
Error: All connections must be closed.
```

- Já no lado do servidor, ele só poderá ser encerrado se a lista de clientes ativos estiver vazia. Assim, caso haja usuários ativos, ele responderá com:

```
Error: Wait until there are no active users.
```

Caso contrário, o comando `finish` encerrará a aplicação do servidor.