Projeto Redes de Computadores

Alunos: José Rui Roque da Silva Fernandes
Pedro Augusto de Almeida Costa Tenório

Repositórios

Servidor: https://github.com/ChatAppRedes/chat-app-server-python
Cliente: https://github.com/ChatAppRedes/ChatAppJavaFX

Funcionalidades

O ParrotChat é um app o qual permite troca de mensagens por usuários que estiverem no servidor, ele possui um interface de menu, que contém a logo do aplicativo e um campo de entrada para o nome do usuário, caso ele deixe o input em branco, será conduzido ao chat como “guest”, na tela do chat, no canto inferior, o usuário poderá enviar mensagens e elas aparecerão acima e em ordem, contendo sempre o nome do usuário que a enviou.

Foram implementadas duas camadas:

- Cliente (Java com JavaFX);
- Servidor (Python 3);

Algumas coisas poderiam ter sido implementadas para deixar o app mais completo, como por exemplo, o suporte a caracteres latinos, como acentuação, o envio de imagens e arquivos, de emojis, e de mensagens de voz, bem como algumas pequenas melhorias na interface poderiam ter sido feitas.

Cliente

O cliente foi implementado com JavaFX e possui duas conexões com o servidor:

- Client Sender (ao enviar uma mensagem);
- Cliente Receiver (escuta a todo momento novas mensagens vindas do servidor).

Servidor

O servidor é multithread e possui uma fila de endereços de usuários. Ao receber uma nova mensagem de determinado usuário, ele manda para todos os usuários dessa fila. Caso não obtenha sucesso, ou seja, determinado usuário desconectou, o servidor remove-o da fila.

Protocolo de comunicação entre Cliente e Servidor:

Criamos um protocolo baseado em JSON. Ele possui os seguintes cenários:

- Ao entrar no chat (Client Sender):
  - Cliente envia somente o username: **{ “username”: “Rui” }**;
  - Servidor responde a todos com um “welcome” (palavra reservada do protocolo): **{ “message”: “welcome”, “username”: “Rui” }**;
- Ao entrar no chat (Client Receiver):
  - Cliente envia somente o receiver: **{ “receiver”: true }**

- Ao enviar uma mensagem:
    - Cliente envia username e message: **{ “username”: “Rui”, “message”: “Hello World” }**;
    - Servidor responde a todos com essa mesma estrutura;
- Ao sair (Client Sender):
    - Cliente envia username e message com um “quit” (palavra reservada do protocolo): **{ “message”: “quit”, “username”: “Rui” }**;
    - Servidor responde a todos com essa mesma estrutura;
- Ao sair (Client Receiver):
    - Cliente envia o username e uma mensagem com um “quit”: **{ “message”: “quit”, “receiver”: true }.**

Dificuldades:

As principais dificuldades encontradas foram no lado do cliente. A primeira delas foi fazer com que o cliente escutasse em tempo real as mensagens vindas do servidor (ou seja, as mensagens de todas as pessoas), porque o JavaFX é executado em single-thread. Caso esse “listener” fosse executado na thread do JavaFX, o programa ficaria bloqueado. Dessa forma, houve a necessidade de criar uma thread só para escutar. No entanto, outras threads não conseguem utilizar as funções do JavaFX dentro delas (nesse caso, a função de adicionar o texto da nova mensagem quando o servidor enviasse). O problema foi resolvido criando um IMessageHandler (uma interface entre a thread do JavaFX e a thread do listener), para abstrair a comunicação entre essas duas camadas. Além disso, a Task que rodava na thread de listener tinha sua propriedade de mensagem associada à propriedade de texto do conteúdo do chat (TextArea do JavaFx), ou seja, toda vez que uma mensagem chegava, o message handler a incrementava numa string e adicionava à propriedade de mensagem da Task. Com isso, a propriedade de texto do conteúdo do chat era sempre atualizada.

Outra dificuldade foi encontrar uma forma de, ao fechar a aplicação, parar a thread de listener, porque, quando a aplicação era fechada pelo usuário, a thread continuava rodando e impedia a aplicação de fechar completamente. Resolvemos esse problema com um interceptador do JavaFX que é executado ao fechar o programa. Quando o usuário fecha o programa, uma mensagem de desconectar é enviada ao servidor e o loop da thread de listener é quebrado.

Instruções:

1. Clonar o repositório do servidor https://github.com/ChatAppRedes/chat-app-server-python
2. Acessar a pasta do código fonte: cd chat-app-server-python/src
3. Executar o servidor com o comando: python3 main.py
4. Baixar o executável do cliente (.jar) em: https://drive.google.com/file/d/1Ni-Es3W4TBeaN3VR8Nw5UeCd03Ozq4QF/view?usp=sharing
5. Caso não tenha o Java 11 instalado, fazer a instalação
6. Executar o cliente (arquivo .jar)