Relatório Trabalho Prático - Sistemas Distribuídos

Grupo 33:
António Luís de Macedo Fernandes (a93312)
José Diogo Martins Vieira (a93251)
João Silva Torres (a93231)
Ricardo Lopes Santos Silva (a93195)

Jan 16, 2022

Contents

1 Introdução

No âmbito do desenvolvimento do projeto da unidade curricular Sistemas Distribuídos foi-nos proposto desenvolver uma plataforma de reserva de voos sob a forma de um par Cliente-Servidor, em Java, utilizando *sockets* e *threads*.

Ora, ao longo deste relatório iremos abordar a forma como pensamos e projetamos todas as funcionalidades básicas e adicionais que dão vida à nossa aplicação. Também vamos explicar com mais detalhe as principais classes deste projeto e a forma como as mesmas foram implementadas.

Na última secção do relatório faremos uma análise crítica sobre todo o trabalho desenvolvido pelo grupo.

2 Resumo

Inicialmente, na realização deste projeto, começamos por definir classes que caracterizam os principais objetos deste trabalho. Sendo elas, *Viagem*, *Voo*, *Local*, *User*, *Admin*, *Contas* e *Reservas*.

De seguida, foi preciso implementar o nosso *Cliente* e *Servidor*. Para tal e com ajuda dos recursos disponibilizados pelos professores, definimos uma classe *Demultiplexer* que como o própio nome indica irá fazer a desmultiplexagem da informação enviada entre o servidor-cliente, e vice-versa. Também foi necessário criar a classe *Connection* que irá auxiliar a gestão desta informação enviada.

Posteriormente, tivemos de implementar métodos que permitissem responder aos objetivos propostos no enunciado e para tal utilizamos tags(int) que permitem distinguir as diferentes funcionalidades do nosso projeto. Para isso, foi também preciso fazermos a diferenciação da autenticação entre um user normal e um administrador pois estes possuem funcionalidades diferentes no nosso sistema que serão melhor abordadas neste relatório.

3 Principais Classes Desenvolvidas

3.1 Servidor

O servidor tem como função processar pedidos provenientes de clientes. Nesse sentido, foi implementado um servidor Threaded-per-connection (tal como abordado no guião 8), isto é, por cada conexão com cada cliente existe uma thread associada. Essa thread é responsável por esta conexão com o cliente, recebe, processa e envia a informação necessária para o cliente.

3.2 Cliente

Esta classe será o nosso *Client*, ou seja, funcionará para enviar sockets com informação para o nosso servidor. Dependendo do que o utilizador escrever no terminal, resultará de uma resposta pelo servidor. Esta informação é entiquetada pelas tags que terão valores diferentes dependendo da opção que escolher.

3.3 Demultiplexer

A classe *Demultiplexer* tem como objetivo agrupar as respostas provenientes do servidor de acordo com a sua tag de forma a distribuí-las pelas threads que enviaram o pedido respetivo. Para cumprir com esse objetivo, existe um thread que está à "escuta" de novas mensagens do socket de input do lado do cliente que após a sua receção irá "acordar" os threads que estejam à espera de resposta da tag da mensagem recebida. (Como foi feito no guião 8).

3.4 Connection

Esta classe *Connection* é responsável pela gestão da troca de mensagens entre cliente e servidor e vice-versa recorrendo a recursos de etiquetamento e serialização de mensagens. Para tal, implementamos métodos de envio e receção de mensagens para o socket, recorrendo à serialização e escrita em binário (DataInputStream e DataOutputStream). Como esta classe foi utilizada nas classes cliente e servidor, existem métodos de implementação diferentes de envio e receção particulares ao cliente e ao servidor. Nesta classe estão implementadas as classes FrameCliente e FrameServidor que tem o intuito de armazenar a informação recebida pelo servidor e cliente, respetivamente.

Para as diferentes funcionalidades da aplicação foram criadas tags para destinguir a diferente informação enviada pelo cliente:

Como funcionalidades comuns tanto ao admin como um user temos:

- Iniciar Sessao = 0;
- Terminar Sessao = 10;

Funcionalidades do user:

- Criar Conta = 1;
- Reservar Viagem = 2;
- cancelar Viagem = 3;
- Ver Voos Existentes = 4;
- Ver Viagens Reservadas = 7;
- Ver todos os Percursos = 8;

Funcionalidades do admin:

- Inserir Voo = 5;
- Encerrar Dia = 6;

3.5 Reservas

Nesta classe faz-se a gestão dos diferentes voos, viagens e locais. Para tal contém como atributos, um mapa com todos os locais disponíveis para fazer viagens, uma lista de voos com todos os voos diários, dois mapas que associam o dia às viagens e voos existentes.

Utilizamos estes atributos para se tornar mais fácil a realização do que foi pedido no enunciado. Também é importante realçar o uso do reentrant lock nesta classe para permitir que haja exclusão nos vários métodos implementados nesta classe.

3.6 Contas

A classe Contas irá armazenar os vários utilizadores e contas criadas no nosso sistemas. Para tal definimos como atributo um mapa com todos os users. Relativamente aos métodos desta classe também foi necessário implementarmos locks.

4 Funcionalidades

Quando o programa inicia, o utilizador tem a opção de iniciar sessão ou criar conta. Caso já tenha conta criada, indica o username e a password que serão envidas ao servidor para serem validadas. Caso não tenha conta, o utilizador cria uma fornecendo um nome e uma palavra-passe e essa inforamção será devidamente guardada.

Caso se trate de um utilizador cliente, será direcionado para o menu cliente, onde terá várias opções por onde escolher:

- **Reservar viagem** - é perguntado ao cliente para indicar o local de origem da viagem, e as sucessivas escalas até ao destino final. Após isso, o cliente indica também o intervalo de datas onde quer que a viagem se realize. Essa informação é enviada ao servidor que em primeiro lugar verifica se para os destinos indicados é possível realizar a viagem. Caso seja, verifica se, para o intervalo de datas mencionado, existe disponibilidade de voos (voo não está cancelado e existem lugares). É dada preferência às datas mais recentes para a marcação da viagem. Caso seja possível realizar, a viagem é marcada e é enviada ao cliente o id da reserva. Caso não seja, o cliente é informado.

- **Cancelar viagem** - é perguntado ao cliente para indicar o id da viagem que pretende cancelar. Essa informação é enviada ao servidor que verifica se o utilizador tem uma viagem com esse id marcada. Caso tenha, a viagem é cancelada e é desmarcado o lugar nos respetivos voos. É enviada a confirmação do cancelamento ao cliente.

- **Voos existentes** - não é necessária informação adicional introduzida pelo cliente. O servidor envia a lista de todos os voos que se realizam diariamente.

- **Ver Viagens Reservadas** - Também não é preciso informação adicional do cliente. O servidor vai buscar todas as viagens reservadas do cliente que efetuou o pedido, e envia informação para cada viagem, relativamente ao id de reserva, à data, estado e aos voos.

- **Todos os percursos** - é perguntado ao cliente para indicar a origem e o destino. Essa informação é enviada ao servidor, que vai para o local de origem verificar todos os percursos que são possíveis realizar, com um máximo de duas escalas, ou seja, três voos. O servidor armazena todos os locais destintos dos percursos. E associa esse nome a um short. É enviada então primeiramente a lista dos destinos fazendo-se acompanhar do short. Seguidamente, para cada percurso, em vez de ser enviado o nome do local, é enviado apenas o short associado ao nome. Assim pretende-se minimizar a quantidade de dados transferidos.

Caso se trate de um utilizador Admin, será direcionado para o menu Admin, onde terá as seguintes opções:

- **Inserir voo** - o administrador introduz a origem, o destino e a capacidade do voo que pretende adicionar. Esta informação é enviada ao servidor que verifica se existe algum voo com a mesma origem e destino, e caso não exista, este é introduzido no sistema.

- **Encerrar dia** - o adminstador indica um o dia que pretende encerrar. O servidor vai cancelar todos os voos desse dia, tal como as viagens e não serão aceites mais reservas para esse dia.

Tanto o utilizador Admin como Cliente têm ainda a opção de terminar sessão. O servidor vai alterar o estado do utlizador para logout, a thread associada ao utilizador vai deixar de receber e vai ainda ser escrita para ficheiro a informação das classes Reservas e Contas. Assim, caso o Servidor seja finalizado, consegue ler a informação a partir desse ficheiro.

5 Conclusão

Nesta fase final do projeto constatamos que conseguimos cumprir com tudo o que nos foi pedido.

Consideramos que foi um projeto bastante interessante e desafiante que, numa fase inicial, nos obrigou a fazer as funcionalidade mais básicas e, à medida que nos fomos familiarizando com o mesmo, procedemos à realização das restantes.

Efetuamos, então, uma funcionalidade adicional: a obtenção de uma lista com todos os percursos possíveis para viajar entre uma origem e um destino, limitados a duas escalas (três voos).

Enquanto grupo, conseguimos distribuir bem o trabalho entre todos. Ajudamo-nos mutuamente e, de forma geral, o grupo teve um aproveitamento positivo.

Contudo, achamos que se o projeto fosse escalar teríamos alterado o Servidor para $Thread - Pool$, que achamos que seria a melhor forma de implementar, mas devido à pequena escala do projeto, e às limitações do tempo, optamos por $Thread - per - Connection$.

Concluindo, este trabalho ajudou-nos a desenvolver novas aptidões e a consolidar toda a matéria lecionada em aula, nomeadamente sobre *threads* e *sockets*.