Reserva de Voos

Licenciatura em Engenharia Informática

Cristiano Pereira, João Martins, Jorge Lima, Rúben Santos
Universidade do Minho
Sistemas Distribuídos
Grupo 14
2021/2022

Abstract

No presente documento é apresentada a descrição da nossa implementação de uma plataforma de reserva de voos sob forma de um par cliente-Servidor, em Java, utilizando *sockets* e *threads*. A plataforma permite aos seus utilizadores aceder aos voos disponíveis (utilizando filtros), reservar viagens constituídas por vários voos e cancelar estas viagens. O administrador do sistema tem ainda a possibilidade de adicionar novos voos, entre outras.
Esta solução parte do princípio que é sempre possível fazer escalas, sendo que se existir um voo A → B e outro de B → C, num dado dia, então é possível para um passageiro fazer A → B → C, se houver disponibilidade de lugares em ambos os voos. Os mesmos voos repetem-se todos os dias.

1. Arquitetura da Solução

Como proposto no enunciado, o sistema está dividido em dois componentes: Cliente e Servidor.

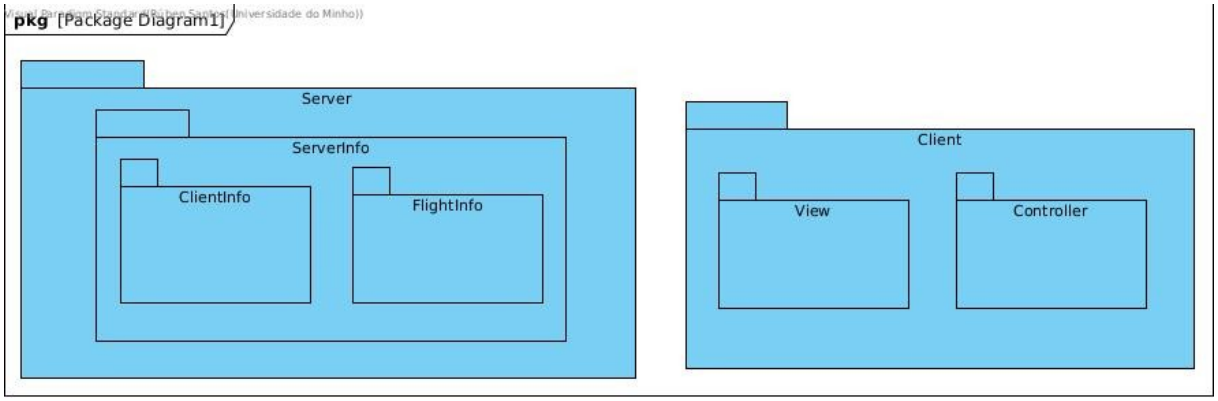

Figure 1: Diagrama de Pacotes

1.1. Servidor

O objetivo principal do servidor é receber, processar e responder, concorrentemente, aos pedidos dos clientes do serviço implementado. Para cumprirmos com este objetivo, o Servidor cria uma *thread* para processar e responder cada pedido que recebe.
Para armazenarmos as informações do Servidor precisamos de uma estrutura de dados: o package *ServerInfo* contém dois componentes que, por sua vez, fazem a gestão da informação de clientes e voos.
Para gerir a informação dos clientes temos o package *ClientInfo*, que permite guardar todas as informações pessoais dos clientes, incluindo também as reservas que fizeram.
A gestão da informação dos voos é feita em *FlightInfo*. Aqui é possível gerir os voos e reservas de todo o sistema, nomeadamente, fazer reservas no intervalo de tempo entre o dia atual e os próximos noventa dias. Este deteta e faz automaticamente a manutenção da passagem de um dia. Também disponibiliza a procura de um percurso entre duas localizações e a proibição de reservas/cancelamentos num dia específico.
Ambos estes *packages* têm implementados mecanismos de gestão de concorrência, dado que serão acedidos por várias threads ao mesmo tempo.

1.2. Cliente

O Cliente, fornece uma interface que possibilita o registo de reservas do utilizador e do administrador. As reservas são realizadas por uma *thread* paralela à principal, que trata da parte gráfica do sistema. Deste modo, é permitido ao utilizador, por exemplo, fazer várias reservas e, mais tarde, verificar o seu resultado.

2. Implementação

Todas as funcionalidades apresentadas estão implementadas com *ReentrantReadWriteLocks*, para permitir que vários clientes consultem e alterem a informação de forma concorrente, tendo como vantagem que o acesso não é bloqueado a várias *threads* que apenas pretendam ler dados.

2.1. Servidor

2.1.1. Server

A classe *Server* do nosso projeto está perpetuamente à espera de receber pedidos TCP de clientes. Sempre que recebe um pedido de conexão cria uma nova instância da classe *ClientHandler* que corre numa *thread* criada para o efeito. Esta é a classe que vai trocar informação com o cliente.
Para que não existam demasiadas *threads* inativas, foi decidido que, se o *login* ou o *signup* falharem, a *thread* termina.

2.1.2. IClientFacade

O componente *IClientFacade* é utilizado para obter e registar informação dos clientes. Além de armazenar as informações pessoais como *username* e *password*, permite também associar o *id* de uma reserva a um cliente, que é utilizado para, posteriormente, cancelar esta reserva, se necessário.

2.1.3. IFlightFacade

O componente *IClientFacade* é utilizado para obter e registar informação de voos. Cada *Flight* contém, além da origem e destino, o número de reservas de cada voo para os próximos 90 dias.
A classe FlightFacade vai fazer a gestão dos Flight's, sendo que fica encarregue de atualizar o dia atual, utilizando a biblioteca LocalDate. A lista com os novos dias vai sendo também atualizada e os dias anteriores removidos. É nesta classe implementado o seguinte conjunto de funcionalidades:

- Cálculo de um caminho entre duas localizações usando um algoritmo de procura iterativa limitada em 3 níveis de profundidade, assim limitando cada viagem a duas escalas (três voos).

- Efetuar reservas e o cancelamento das mesmas.
- No caso de uma reserva, o caminho é calculado com o algoritmo acima, sendo que, se um cliente pretende viajar de Porto - Paris, e não existe disponibilidade num voo direto, o sistema procura uma alternativa e regista, se disponível, uma reserva como dois voos: Porto – Madrid, Madrid – Paris.

Ambos os componentes (*IclientFacade, IflightFacade*) fazem uso de exceções quando ocorre algum erro, permitindo assim ao servidor emitir e resolver o problema. Também permitem a escrita das suas estruturas em ficheiros binários, para que o servidor, sempre que iniciar, carregar toda a informação anterior.

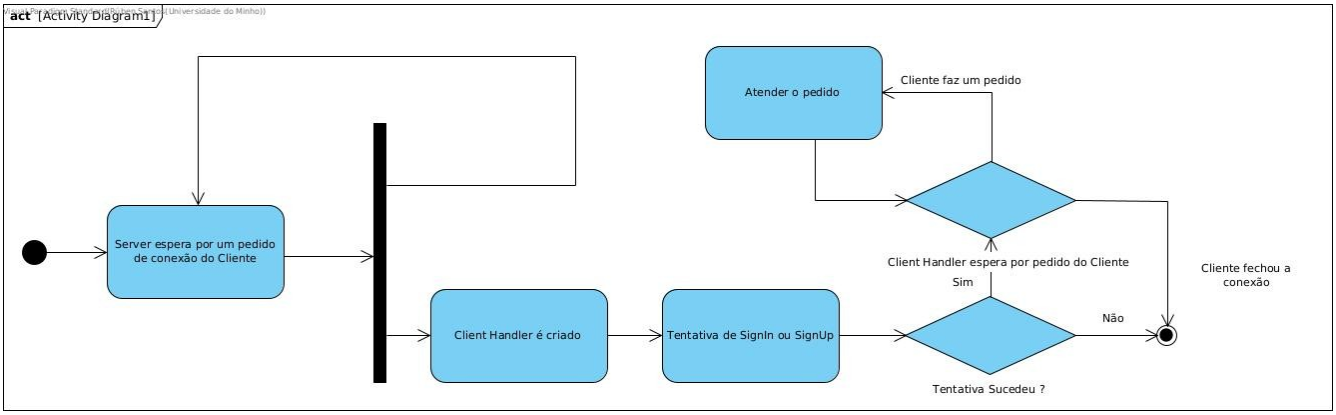

Figure 2: Diagrama de Atividade (Server)

2.2. Cliente

Abaixo segue uma descrição textual do fluxo de informação proveniente da autenticação do cliente.
A iniciar o *Cliente*, o utilizador tem a opção de efetuar *login* ou, caso não possua uma autenticação, *signup*. Estas duas opções funcionam de maneira muito semelhante, sendo que o utilizador fornece apenas as suas credenciais, nome de utilizador e palavra-passe.
A informação da autenticação é então inserida numa lista de pedidos, pertencente à classe *ClientWorker*. É criada a *thread* que irá executar o *RequestWorker*. A *thread* principal adormece à espera da resposta. O *RequestWorker* verifica se existe algum pedido na lista de pedidos: neste caso temos um pedido de autenticação, que enviamos de seguida para o servidor. A resposta é recebida e processada ainda nesta classe. O conteúdo desta resposta é colocado numa lista de respostas, pertencente à classe *ClientWorker*. A inserção de elementes nesta lista dá um sinal à *thread* principal, que verifica de seguida a lista de respostas, e volta para o menu principal. O *RequestWorker* adormece, até que seja necessário enviar mais algum pedido.
No menu principal, o utilizador tem todas as funcionalidades pedidas no guião do trabalho prático. Sempre que o utilizador decidir utilizar uma funcionalidade é gerado um pedido, que é inserido na lista de pedidos. O *RequestWorker* acorda e processa o pedido. Desta forma temos a possibilidade de uma *thread* estar a resolver um pedido de reserva enquanto que o utilizador tem acesso à interface.

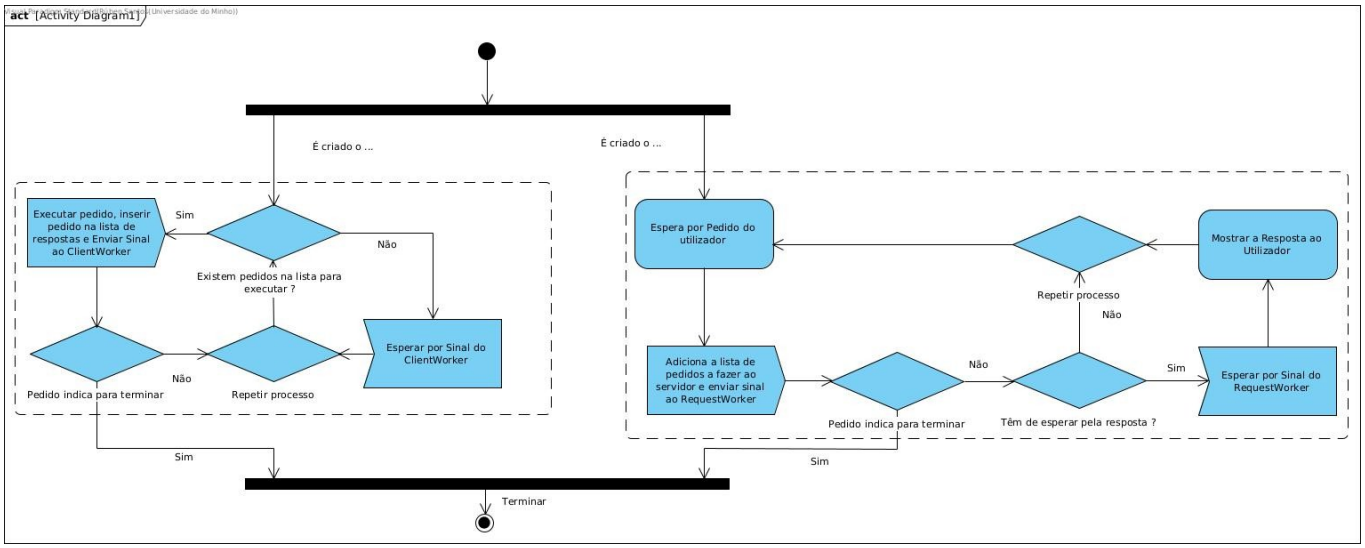

Figure 3: Diagrama de Atividade (Cliente)

3. Conclusão

Analisando o projeto final, conseguimos dizer que atingimos todos objetivos desejáveis. A implementação deste sistema ajudou-nos a complementar os conceitos teóricos da utilização de *Locks*, *Conditions* e Comunicação em TCP. A programação em sockets TCP, e os conhecimentos que advém dela, são algo que iremos certamente continuar a usar em projetos futuros.
Achamos que o projeto podia ter algumas pequenas alterações, mas que seriam irrelevantes para o propósito de sistemas distribuídos (p.ex interface gráfica, base de dados, etc.).

Referências

M. van Steen and A.S. Tanenbaum, Distributed Systems, 3rd ed., distributed-systems.net, 2017.
ReentrantLock - Oracle
Condition - Oracle