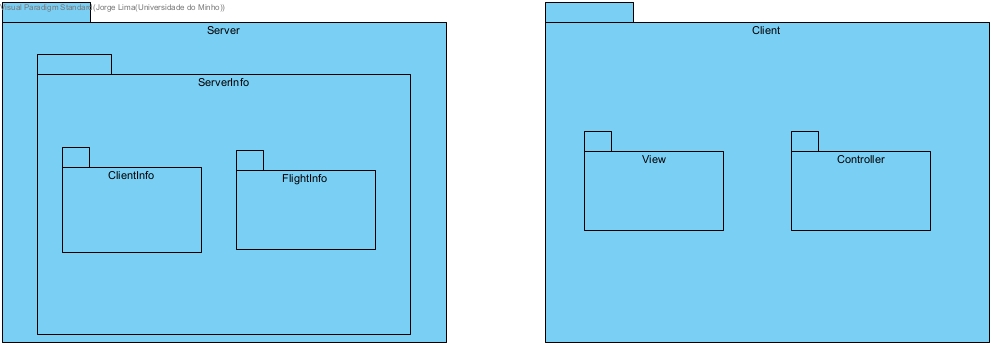
Arquitetura



Como era de esperar, dividimos o sistema em duas grandes partes: cliente e servidor.

O objetivo do servidor é responder aos pedidos do cliente. Deve permite vários clientes fazer pedidos ao mesmo tempo e para isto cria uma “thread” para responder a cada cliente. Para este efeito necessita de uma estrutura de dados para armazenar informações. Para isso criamos o package “ServerInfo” que contém dois componentes que fazem a gestão da informação sobre os clientes e sobre os voos.

Para gerir os clientes o package “ClientInfo” permite guardar todas as informações básicas, sobre estes, e também os voos aos quais fizeram reservas. O armazenamento dos voos é feito em “FlightInfo”. Aqui é possível gerir os voos e reservas de todo o sistema, permitindo guardar as reservas do intervalo de tempo entre o dia atual e os próximos noventa dias e isso implica também fazer a manutenção da passagem de um dia. Também disponibiliza a procura de um percurso entre duas localizações e a proibição de ações num dia específico.

Ambos estes componentes vão ter em atenção que irão ser acedidos por várias “threads” e irão fazer esta gestão.

Por outro lado, o cliente, além de fazer os pedidos ao servidor, vai permitir a utilização da aplicação, fornecendo interfaces que registam os pedidos do utilizador e do administrador e paralelamente os transmite ao servidor, usando uma “thread” para comunicar com o servidor e outra para a interface. Assim permitimos ao utilizador, por exemplo, fazer várias reservas e, mais tarde, verificar o seu resultado.

Implementação

Servidor:

A classe “Server” do nosso projeto está sempre a ouvir pedidos em TCP por parte do cliente. Assim sempre que recebe um pedido de conexão apenas cria uma nova “thread”, ou seja, um “ClientHandler”. Este sim vai trocar informação com o cliente. Para não existirem demasiadas “threads” que não trabalham se nas primeiras trocas de informação o “login” ou o “signup” falharem a “thread” desliga-se. Caso contrário vai manter a conexão estando sempre a espera do próximo pedido do cliente. Usa os componentes “IFlightFacade” e “IClientFacade” para obter e registar a informação.

O componente “IClientFacade” vai conter um conjunto de clientes e toda a sua informação. Além das informações básicas como username, password,etc permite também associar o id de uma reserva a um cliente, que vai ser útil no caso de este decidir a cancelar. Implementa um sistema de “ReentrantReadWriteLocks” para permitir o acesso de várias “threads”.

O componente “IFlightFacade” é um pouco mais complexo. Cada “Flight” contém além da origem e destino, o número de reservas para os próximos 90 dias. Logo a classe “FlightFacade” vai fazer esta gestão, verificando sempre que passa um dia e, usando a biblioteca “LocalDate”, atualizando a lista com os novos dias e removendo os anteriores. Também contém uma lista dos dias em que os administradores bloquearam reservas e cancelamentos de reservas. Permite o cálculo de um caminho entre duas localizações usando um algoritmo de procura iterativa limitada em 3 níveis de profundidade. Quando estão reunidas todas as condições permite a reserva e o cancelamento de reservas. No caso da reserva tenta encontrar sempre um caminho usando o mesmo algoritmo “SUPRAMENCIONADO”, ou seja, se um cliente pretendia ir de Lisboa ao Porto e o sistema apenas o consegue fazer se existir uma paragem intermediaria em Coimbra, este regista a reserva como dois voos, Lisboa a Coimbra e Coimbra ao Porto. Todas estas funções estão a ser feitas com o uso de “ReentrantReadWriteLocks” para permitir que varias clientes consultem e alterem a informação.

Ambos estes componentes usam exceções quando acontece algum erro, permitindo assim ao servidor transmitir a mensagem de erro. Também permitem a escrita das suas estruturas em ficheiros em binário, para o servidor sempre que iniciar poder carregar toda a sua informação.

Cliente:

A iniciar o cliente, o utilizador tem duas opções de dar “sign in” ou “sign up”. Estas duas opções funcionam de maneira muito semelhante, o utilizador insere as suas credenciais, nome de utilizador e palavra-passe, apos isto é inserido numa lista de pedidos localizado na classe “ClientWorker” e uma “thread” com o “RequestWorker” é inicializada e a “thread” principal adormece a espera da resposta. O “RequestWorker” vai verificar se existe algum request na lista de pedidos envia para o servidor e aguarda resposta, no “sign in” o servidor verifica se as credencias existem e se estão corretas, no “sign up” o servidor vai inserir as credenciais no “ClientInfo”, após isto, envia a resposta para o cliente novamente. O “RequestWorker” quando receber a resposta do servidor verifica a resposta e insere a resposta na lista de respostas localizado no “ClientWorker” e dá “signal” á “thread principal”, esta verifica a lista de respostas e entra para o menu principal e o “RequestWorker” adormece.

No menu principal, o utilizador tem as diversas funcionalidades descritas no guião, sempre que quer utilizar uma funcionalidade é gerado um pedido e inserido na lista de pedidos, no caso da lista estar vazia, dá signal ao “RequestWorker” para este tratar do pedido, desta forma temos a possibilidade de uma thread estar a resolver o pedido e outra no interpretador do cliente.

Conclusão

* O trabalho correu bem, cumprindo todos os objetivos que desejamos.
* Ajudou a perceber melhor a utilização de locks, conditions e a comunicação em TCP.
* Coisas a melhorar? Base de dados? Faz sentido nesta cadeira?