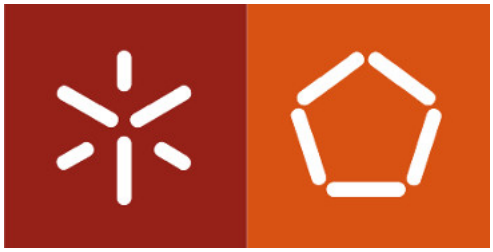

Universidade do Minho
Licenciatura em Engenharia Informática

# Sistemas Distribuídos

TP - Grupo 20

André Silva - A87958 Armando Silva - A87949
Joana Oliveira - A87956 João Nunes - A87972

Janeiro 2022

# Índice

# 1 Introdução

Neste trabalho prático da UC de Sistemas Distribuídos foi-nos proposto a implementação de uma plataforma de reserva de voos, com a obrigatoriedade da utilização de *sockets* para estabelecer a ligação entre cliente-servidor e também do desenvolvimento desta aplicação num ambiente *multithreading*. Para a concretização deste projeto, foi necessário aplicar vários conceitos aprendidos e praticados durante as aulas teóricas e práticas, nomeadamente o controlo da concorrência, exclusão mútua, secção crítica, entre outros.

# 2 Servidor

O Servidor inicializa o seu *socket* na porta '12345' onde serão realizadas todas as trocas de mensagens com os vários Clientes. Depois da conexão ter sido estabelecida, o Servidor vai tratar dos dados que recebeu do Cliente e executar os métodos necessários para a sua realização. Para tal, vai recorrer à classe Model que contém todos os dados fundamentais para a execução das funcionalidades pedidas pelos diferentes utilizadores do programa.

# 3 Cliente

Para o estabelecimento da ligação do Cliente com o Servidor, este conecta-se ao *socket* criado pelo Servidor, mencionado na secção acima. Após esta conexão ter sido definida, vai ser mostrado o menu inicial com as opções de registo e autenticação. Depois de autenticado, o Cliente vai ter várias opções de pedidos.

De modo a cumprir com os requisitos colocados no enunciado, desenvolvemos dois tipos de clientes: cliente normal e administrador. A diferença entre os dois está principalmente nos distintos menus criados para cada um. Como o nome de cada tipo indica, o cliente normal tem acesso às principais funcionalidades do programa, como agendar ou cancelar um voo. Já o administrador possui opções para a gestão do programa em si, como finalizar um dia ou adicionar um novo voo no sistema.

# 4 Conexão: Cliente ↔ Servidor

Para a conexão entre o Cliente e o Servidor foi necessário recorrer a duas classes, Demultiplexer e TaggedConnection, estas que foram definidas durante as aulas práticas. Explicamos a seguir a metodologia para as duas direções possíveis nesta conexão.

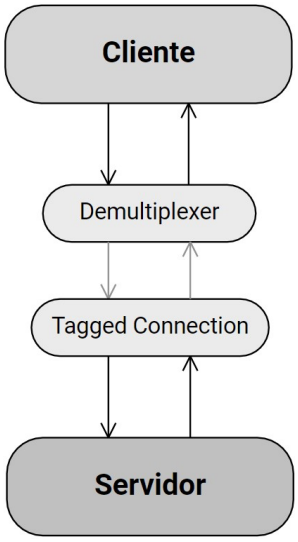


**Figura 1.** Modelo que demonstra a conexão entre Cliente e Servidor

## 4.1 Cliente → Servidor

Quando um Cliente tenta enviar alguma informação para o Servidor, este utiliza o método 'send' da classe *Demultiplexer* com uma *tag* específica da opção que escolheu. O Servidor para receber esta informação invoca o método 'receive' da classe *TaggedConnection* e dependendo da *tag* que recebeu vai executar os métodos necessários para a sua conclusão.

## 4.2 Servidor → Cliente

Quando um Servidor pretender enviar dados para um Cliente, este utiliza o método 'send' da classe *TaggedConnection* que envia a informação e a *tag* respetiva. Como a aplicação está num ambiente de *multithreading* existe a possibilidade de informação ser mandada para Clientes errados. Para evitar isso o Cliente utiliza, como já mencionado, a classe *Demultiplexer* que é responsável por organizar as mensagens recebidas com a respetiva *tag* num *buffer*. Assim, quando o Cliente quer receber os dados, este invoca o método 'receive' que vai procurar o conteúdo correspondente à *tag* pretendida no *buffer*. Caso este esteja vazio, então a *thread* vai ficar à espera que os dados cheguem.

# 5 Funcionalidades

Para a implementação desta aplicação, como já foi referido na secção acima, foram considerados dois diferentes tipos de clientes. Cada um destes tipos possui funcionalidades únicas. O Cliente normal pode reservar uma viagem nova, cancelar a reserva de alguma viagem que tenha pedido e obter a lista de todos os voos presentes no sistema. Já o administrador pode adicionar novos voos ao sistema e terminar um dia, isto para que o sistema comece a rejeitar pedidos de reserva para o dia encerrado pelo administrador. De salientar que, antes destas funcionalidades estejam disponíveis é necessário o Cliente se registar e autenticar.

## 5.1 Autenticação e registo

Para o **Cliente** ganhar acesso às suas funcionalidades tem de efetuar o seu registo no sistema (opção 1 no menu). Caso este já esteja registado tem a opção de se autenticar (opção 2 no menu). Ambas as opções pedem um id e uma palavra passe. De salientar que, é possível um novo administrador se registar, para isso é necessário que saiba o id especial para o seu registo, esta que é a única maneira de se registar um administrador. Por fim, vai ser enviado em binário para o Servidor a informação introduzida com a *tag* 0 ou 1.

No lado do **Servidor** este vai converter os dados recebidos nos tipos correspondentes e no caso de um novo registo, vai verificar se o id é especial ou não e se é único, se sim vai adicioná-lo ao mapa que guarda todos os utilizadores. Já no caso do Cliente tentar efetuar a sua autenticação, o Servidor vai verificar se os dados que recebeu são os corretos e permitir o acesso à conta ou não. Vai ainda modificar a sua variável de autenticado para verdadeiro, caso tenha obtido acesso. De seguida, envia para o Cliente a resposta obtida da tentativa de registo ou autenticação.

Para concluir este processo, o **Cliente** vai receber a resposta do Servidor. No caso de ser um registo, vai receber a validação do registo. Se for uma autenticação bem sucedida é permitido o acesso às funcionalidades respetivas, se não é mostrada uma mensagem de erro.

## 5.2 Adicionar informação sobre voos

Funcionalidade apenas disponível para um **Administrador**. Este precisa de inserir o local de partida e de chegada e a capacidade máxima de clientes num único voo. Vai depois ser enviado para o Servidor esta informação com a *tag* 5.

O **Servidor** vai receber esta informação, converter para os tipos correspondentes e caso a nova rota ainda não esteja no sistema, vai adicioná-la na lista que guarda todos os voos. No caso da rota já existir, esta não vai ser guardada no sistema. Vai enviar para o Administrador o resultado da adição do voo.

Para terminar, o **Administrador** vai receber o resultado da adição da rota que inseriu, se foi bem sucedida ou não.

## 5.3 Encerrar um dia

Funcionalidade apenas disponível para um **Administrador**. Este não necessita de inserir qualquer informação, pois o sistema automaticamente encerra o dia atual. É enviado para o Servidor a *tag* 6.

No outro lado, o **Servidor** vai atualizar o dia atual para o seguinte, impedindo que novas reservas possam ser feitas para o dia terminado. Vai ser enviado para o Administrador o resultado da operação.

Por fim, o **Administrador** vai receber o resultado da operação que o Servidor realizou, se foi realizada com sucesso ou não.

### 5.4 Reservar uma viagem

Funcionalidade unicamente disponível para um **Cliente normal**. Necessita de inserir todos os locais da escala que deseja realizar, separado por '-', por exemplo Porto-Barcelona-Paris. Também precisa de inserir um intervalo de datas, separado por '/', que indica ao sistema um intervalo de datas que o Cliente deseja reservar a viagem. Envia para o Servidor esta informação com a *tag* 2.

Já o **Servidor** vai transformar toda esta informação nos tipos correspondentes. Caso todos os dias que o Cliente pediu a reserva estejam já encerrados, a viagem não é registada. Já no caso da reserva ser válida, é obtido um código de reserva aleatório único e incrementado o número de lugares ocupados no voo (ou voos). Esta informação é depois inserida num mapa responsável por guardar todos os códigos de reserva e os respetivos voos associados. É ainda guardado no Cliente o código da reserva. Depois de terminados todos os passos necessários para a execução do pedido, o Servidor vai enviar para o Cliente o código de reserva obtido ou um código nulo que representa a falha da reserva da viagem.

Este processo é encerrado com o **Cliente normal** a receber o respetivo código da reserva ou uma mensagem de erro.

### 5.5 Cancelar uma reserva

Funcionalidade exclusiva a **Clientes normais**. Este tem de introduzir o código da reserva que deseja cancelar. É enviado para o Servidor o código e a *tag* 4.

O **Servidor** vai receber o código da reserva e verificar se este existe. Se existir então vai ser verificado se o dia já foi encerrado ou não, pois se tiver sido encerrado não vai ser possível cancelar a reserva. Se o dia não tiver sido encerrado então vai ser verificado se o Cliente possui o código e se sim este vai ser removido da lista de reservas do Cliente e o número de lugares ocupados nos voos vai ser decrementado. Terminada a execução deste método, o Servidor vai enviar ao Cliente se o cancelamento da reserva foi realizado com sucesso ou não.

O **Cliente normal** vai receber o resultado do cancelamento da reserva.

### 5.6 Obter lista de voos

Funcionalidade apenas disponível para um **Cliente normal**. Não é necessário introduzir nenhuma informação para o sistema executar este pedido. É enviado para o Servidor a *tag* 3.

O **Servidor** vai construir a *String* com todas as rotas presentes no sistema. Vai enviar para o Cliente toda esta informação.

Por fim, o **Cliente normal** vai receber e expor toda esta informação de maneira agradável e simples.

### 5.7 Terminar sessão

Funcionalidade disponível tanto para o **Cliente normal** como para o **Administrador**. Não precisam de inserir nada, pois apenas vão terminar a sua sessão. É enviado para o Servidor a *tag* 7.

O **Servidor** apenas vai alterar a variável do Cliente, responsável por identificar se este está atualmente autenticado ou não, para falso. Envia para o Cliente a validação da operação.

Por último, o **Cliente** vai perder acesso às suas funcionalidades e vai voltar para o menu inicial do programa.

### 5.8 Funcionalidade Adicional 2

Funcionalidade implementada com a adição de uma *thread* iniciada depois do Cliente ter pedido a opção 'Reservar uma viagem'. Assim, essa *thread* fica responsável por pedir a reserva de uma viagem e de receber o resultado do mesmo. Permitindo que o Cliente faça outras operações enquanto o sistema trata da reserva da viagem.

## 6 Conclusão

Dado por concluído este projeto, consideramos importante realizar uma análise crítica, considerar possibilidades para trabalho futuro, e ainda, realizar uma avaliação final do trabalho realizado.

O facto de todas as funcionalidades pedidas estarem implementadas e funcionais é um ponto positivo do nosso trabalho. Além disso, a utilização correta de *threads*, permitindo a utilização concorrente de vários utilizadores, sem que exista incoerência de dados, também é um ponto fulcral do nosso projeto. O facto de nem todos os possíveis casos de verificação terem sido feitos pode criar erros que poderiam ter sido prevenidos, sendo este o ponto onde o grupo concorda que poderia ter feito mais e melhor.

De forma a completar ainda mais o projeto, seria uma boa ideia adicionar mais funcionalidades, tanto para os administradores como para os clientes. Implementar uma interface muito mais agradável e interativa seria também um excelente trabalho para o futuro.

Para concluir, o grupo concorda que o trabalho desenvolvido foi positivo uma vez que responde a todas as tarefas pedidas.