**Universidade do Minho**
Mestrado Integrado em Engenharia Informática

# Sistemas Distribuídos

Ano letivo de 2018/2019

# Cloud Servers

## Grupo 25

A71407 - Maurício Salgado

A76270 - Joana Cruz

# Introdução

O objetivo deste trabalho é desenvolver um serviço de alocação de servidores na nuvem e de contabilização do custo incorrido pelos utilizadores recorrendo às competências desenvolvidas na unidade curricular de Sistemas Distribuídos. A reserva de um servidor pode ser feita a pedido ou em leilão. Quando um utilizador reserva um servidor a pedido o servidor fica-lhe atribuído até o próprio o decidir libertar, sendo cobrado o preço por hora correspondente ao tempo utilizado. Enquanto na reserva de um servidor em leilão o utilizador propõe o preço que está disposto a pagar por esse tipo de servidor, e quando o seu preço proposto for o maior entre todas as licitações pelos diversos utilizadores, o servidor é lhe atribuído. Por último, as reservas de servidores a pedido têm prioridade sobre as reservas de servidores em leilão, logo uma reserva de servidor em leilão poderá ser cancelada pela nuvem, para poder satisfazer uma reserva a pedido. Tantos os programas Cliente e Servidor foram escritos em Java e usando threads e a comunicação é feita via sockets TCP, sendo orientada à linha. O endereço é localhost e a porta utilizada é a 12346.

# Cliente

O programa CloudServersClient pode comunicar com o servidor através de sockets TCP para satisfazer uma das seguintes ações:

- registo de um novo utilizador
- efetuar autenticação de um utilizador
- visualizar a lista de tipos servidores disponíveis pela nuvem, assim como o seu preço por hora, o número de servidores livres, ocupados a pedido e ocupados em leilão
- reservar um servidor a pedido
- reservar um servidor em leilão, fornecendo o preço que pretende pagar por hora
- visualizar a sua dívida atual
- visualizar a sua lista de reservas que estão ativas, ou seja que um servidor já lhe foi atribuído, assim como o tempo e preço atual
- visualizar as suas reservas a pedido pendentes
- visualizar as suas reservas em leilão pendentes
- cancelar tanto uma reserva a pedido como uma reserva em leilão
- desalocar um servidor

O cliente é composto pelas seguintes classes:

- CloudServersClient - Função *main* do cliente, e métodos para interação entre menus
- ClientReader – thread responsável pela leitura de mensagens vindas do servidor, e distinguir se é uma resposta a um pedido ou uma notificação devido a um cancelamento de uma reserva em leilão pela nuvem
- ConnectionResources – disponibiliza os métodos de escrita e de leitura para um servidor, e de instanciar e fechar um socket
- UI – disponibiliza todos os menus e opções de cada menu
- Menu e Option – classes auxiliares para construção de um menu

Todas as mensagens são enviadas para o servidor via socket TCP da seguinte maneira:
*"<código_operação> + <argumentos_necessários_operação>"*

## Menu principal

Quando o utilizador inicia a aplicação é lhe apresentado um menu em que pode escolher registar-se, efetuar autenticação, ou sair da aplicação.

### Registo de um utilizador

Caso o utilizador escolha registar-se é-lhe pedido que indique o email e a palavra-passe pretendida. O utilizador fica registado se o email estiver disponível, ou seja, se não houver mais nenhuma conta com o mesmo email. Caso corra tudo bem, o servidor retorna uma mensagem de sucesso, senão retorna que o utilizador deve escolher outro email para se poder registar.
*Mensagem enviada: register + email + password*

### Efetuar autenticação

Caso o utilizador escolha fazer login é lhe pedido o seu email e a sua palavra-passe. Se email não for válido o servidor irá responder com uma mensagem de erro a dizer que o email não existe, e caso seja a password incorreta o servidor irá responder com uma mensagem de erro explicando o tipo de erro. Caso as credencias sejam válidas o servidor responde com uma mensagem de sucesso, e assim entra para o menu disponível para o utilizador.
*Mensagem enviada: login + email + password*

## Menu Utilizador

Visualizar a lista dos servidores disponibilizados pela nuvem

Caso o utilizador escolha esta opção ele poderá ver os servidores disponibilizados pela nuvem, assim como quais os seus preços, número de disponíveis, ocupados a pedido e ocupados em leilão.

*Mensagem enviada: serversList*

Visualizar a lista dos seus servidores e reservas a pedido em espera

Caso o utilizador escolha esta opção ele poderá ver os seus servidores que lhe foram atribuídos(reservas que estão ativas), assim como as suas reservas que estão à espera de algum servidor do tipo pedido disponível. O servidor irá responder com a tabela correspondente aos seus servidores e reservas a pedido à espera.

*Mensagem enviada: myServers*

Visualizar a lista das suas reservas em leilão em espera

Caso o utilizador escolha esta opção ele poderá ver as suas reservas de servidores em leilão(reservas que estão à espera). O servidor irá responder com a tabela correspondente às suas reservas em leilão à espera.

*Mensagem enviada: bidsList*

Visualizar dívida atual

*Mensagem enviada: currentDebt*

Fazer LogOut

*Mensagem enviada: logout*

## Menu Servidores Disponibilizados

Reservar um servidor a pedido

Caso o utilizador reserve um servidor a pedido, tem que fornecer o tipo de servidor que quer reservar. O servidor irá responder com o número da reserva que lhe foi atribuída, ou com uma mensagem de erro caso o tipo de servidor que o utilizador pediu não existe no sistema.

*Mensagem enviada: serverDemand + serverType*

Reservar um servidor em leilão

Caso o utilizador reserve um servidor a pedido, tem que fornecer o tipo de servidor que quer reservar.
O servidor irá responder com o número da reserva que lhe foi atribuída, ou com uma mensagem de erro caso o tipo de servidor que o utilizador pediu não existe no sistema.
*Mensagem enviada: serverAuction + serverType*

### Menu meus servidores e reservas

Desalocar um servidor que já lhe foi atribuído
Caso o utilizador escolha desalocar um servidor, tem que inserir o número da reserva que esse servidor está alocado. O servidor irá responder com uma mensagem de sucesso, dizendo que o servidor foi desalocado, ou com uma mensagem de erro caso o número de reserva inserido não exista
*Mensagem enviada: deallocate + numberReservation*

Cancelar uma reserva que se encontra à espera
O servidor irá responder com uma mensagem de sucesso, dizendo que a reserva foi cancelada ou com uma mensagem de erro caso o número de reserva inserido não exista
*Mensagem enviada: cancel + numberReservation*

### Menu minhas reservas em leilão

Cancelar uma reserva
O servidor irá responder com uma mensagem de sucesso, dizendo que a reserva foi cancelada ou com uma mensagem de erro caso o número de reserva inserido não exista
*Mensagem enviada: cancelBid + numberReservation*

## Servidor

O servidor está encarregado de fazer toda a gestão dos pedidos dos vários clientes e comunica com o cliente via *sockets* TCP baseado em texto, mantendo em memória a informação relevante descrita em cima. Cada cliente tem associado a si duas *threads:* uma para responder aos pedidos dos clientes e outra para notificar clientes. Cada *thread* não escreve em mais do que um *socket*.

O programa CloudServers é composto pelas seguintes classes:

- CloudServers - classe que contêm a main do programa

- **Classes de excepção -** EmailAlreadyTakenException, InexistingReservationNumberException, InexistingServerTypeExeption, NotExistantUserException
- **User -** classe utilizador com as seguintes variáveis de instância, email e password, assim como todas as suas reservas ativas e que terminaram
- **UserDAO -** singleton com o HashMap que guarda os utilizadores do sistema, o HashMap que a cada identificador de utilizador guarda as suas notificações, e a variável de condição *hasNotifications*
- **ServerInstance** – classe servidor que tem como atributos o tipo, id, a reserva que esta alocado de momento, o preço por hora, o estado do servidor
- **ServerState** – estados em que um servidor se pode encontrar FREE, BUSY_DEMAND, BUSY_SPOT
- **ServerInstanceDAO** – singleton com o HashMap que guarda para cada tipo de servidor os servidores disponibilizados pelo sistema, assim como o HashMap que guarda para cada tipo de servidor as suas instâncias disponíveis. Aqui temos a variável de condição *serversAvailable* que será usada no *Allocator*
- **Reservation** – classe com os seguintes atributos o identificador da reserva, o utilizador associado a essa reserva, o servidor alocado a essa reserva, o estado da reserva, o preço por hora(pois este pode ser modificado caso a reserva seja em leilão), a hora no momento em que foi alocado, a hora no momento em que foi desalocado, o tipo de reserva
- **ReservationState** – estados que uma reserva pode tomar WAITING, ATIVE, FINISHED
- **ReservationDAO** - Aqui temos a variável de condição *hasReservations* que será usada no *Allocator*
- **DemandDAO** - contem a lista com todas as reservas a pedido que estão à espera de algum servidor para lhe ser atribuído
- **BidsDAO** – contêm a lista com todas as reservas em leilão que estão à espera de um servidor disponível para lhe ser atribuído
- **ReservationID** – classe responsável pelo incremento dos identificadores das reservas
- **Allocator** – thread responsável pela alocação de servidores para reservas que se encontram à espera, sejam reservas a pedido ou em leilão
- **Notificator** – thread responsável pelo envio de uma notificação ao cliente caso a sua reserva em leilão tenha que ser cancelada para dar prioridade a uma reserva a pedido
- **Server** – thread responsável pela gestão dos pedidos do cliente, dependendo da mensagem enviada do utilizador, irá encaminha para o respetivo método

A thread *Allocator* faz toda a gestão de alocação de servidores a reservas à espera, dando sempre prioridade às reservas a pedido, e só depois de todas as reservas a pedido tiverem sido tratadas, esta thread irá tratar das reservas em leilão caso ainda haja servidores disponíveis. O Allocator não irá só verificar os servidores que se encontram livres no caso de ser uma reserva a pedido, mas sim também aqueles que se encontram ocupados para uma reserva em leilão. Caso encontre algum ocupado em leilão faremos um signalAll para a variável de condição *hasNotifications,* para o utilizador cuja sua reserva foi cancelada ser notificado. Caso não existam reservas, ou não existam servidores disponíveis ela irá para o estado em que se encontra "a dormir". Quando o servidor recebe uma mensagem do tipo "deallocate", faremos um signalAll para a variável de condição *serversAvailable*, assim a reserva que há mais tempo se encontra à espera irá ter um servidor atribuído. Quando o servidor recebe uma mensagem do tipo "serverDemand"(reserva a pedido) ou "serverAuction"(reserva em leilão) faremos um signalAll para a variável de condição *hasReservations*.

# Conclusão

Com o desenvolvimento deste trabalho prático conseguimos ter um conhecimento mais sólido e uma maior destreza na utilização de threads, a capacidade de entender claramente como gerir concorrência e acesso a dados e uma visão geral de como as ligações entre um servidor e os vários clientes que a ele se querem conectar funcionam na vida real.

Foram cumpridos todos os requisitos do sistema, sendo estes implementados com mecanismos de controlo de concorrência tanto nos processos de escrita como nos de leitura.