



Universidade do Minho

# Sistemas Distribuídos

## Relatório do Projeto Prático

Mestrado Integrado em Engenharia Informática

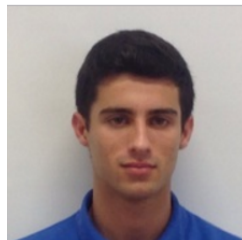
Ano Letivo 2016/2017



A73674 – Alexandre Lopes Mandim da  
Silva



A73825 – Diogo Mendes Gomes



A74219 - Hugo Alves Carvalho



A74260 - Luís Miguel da Cunha Lima

## 1. Introdução

No âmbito da unidade curricular de Sistemas Distribuídos do 3º ano do Mestrado Integrado em Engenharia Informática, foi proposta a implementação de uma aplicação distribuída que permita a gestão de um serviço de leilões, podendo ser acedida por vendedores e compradores.

Neste relatório irá ser explicado como implementamos o cliente e o servidor, bem como o protocolo existente entre si. Também serão explicadas as funcionalidades propostas no enunciado do trabalho prático: registo de utilizador, iniciar um leilão, listas leilões, licitar um item e indicar o fim do leilão de um item.

## 2. Cliente

Neste tópico vamos discutir sobre os métodos que foram implementados no cliente, de modo a respeitar todos os requisitos apresentados no enunciado.

Para começar iremos explicar as *threads* criadas, que servem para o utilizador desta aplicação comunicar com o servidor, e vice-versa. Para complementar também iremos apresentar o protocolo Cliente-Servidor, que permite ao servidor compreender o que o cliente pretende e ao cliente compreender o que o servidor responde.

Em seguida segue-se uma ilustração que visa representar como o Cliente funciona e comunica com o servidor. Esta ilustração tem como objetivo entender melhor a implementação do Cliente que será explicada nos tópicos seguintes.

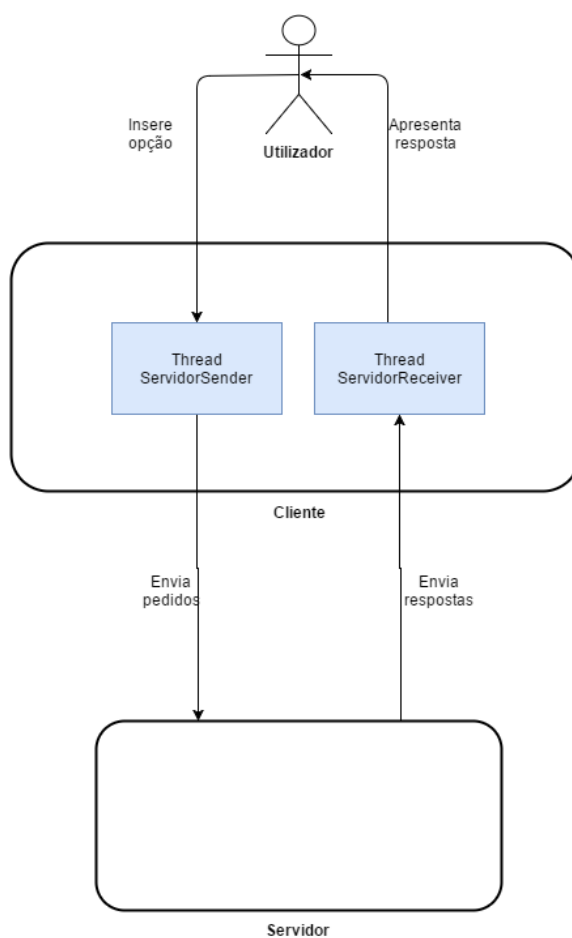


Ilustração 1 Ilustração da implementação do Cliente

## 2.1 Threads

O nosso cliente é composto por duas *threads* e uma classe auxiliar, sendo que uma *thread* envia um pedido para o servidor (ServidorSender), e a outra *thread* recebe respostas do servidor (ServidorReciever), o seu funcionamento será explicado de seguida. A classe auxiliar (Util) serve para registar se o utilizador já realizou login ou não, esta é composta por dois métodos. O método `estaLogado` retorna um boolean que indica se o utilizador já fez login ou não, e o método `setLogado` vai colocar o utilizador como “logado”, ou seja vai indicar que o utilizador já iniciou sessão.

### 2.1.1 ServidorReciever

Quando o utilizador corre esta aplicação, a primeira ação tomada pela *thread* ServidorReciever será verificar se este utilizador já fez login ou não.

Caso este ainda não esteja “logado” será mostrado ao utilizador um primeiro menu, através do método `mostrarMenu`, que permite registar um utilizador, ou iniciar sessão. Caso estes já tenham feito login, então será mostrado um segundo menu, através do método `mostrarSegMenu`, que contém as funcionalidades pedidas pelo enunciado.

Esta *thread* está constantemente à espera que o servidor lhe envie respostas. Sempre que uma resposta chega do servidor, esta *thread* cria um *StringTokenizer* sobre a resposta e envia para uma função auxiliar (`analisaResposta`) que vai analisar a resposta e apresenta-la ao cliente, após ser apresentado ao cliente a resposta esta *thread* volta a ficar a espera que o servidor lhe envie outra resposta.

### 2.1.2 ServidorSender

Esta *thread*, ServidorSender, como dito anteriormente, servirá como uma estrada de um sentido, do utilizador para o servidor. Quer isto dizer que esta *thread* serve unicamente para o utilizador enviar pedidos para o servidor.

Esta *thread* irá transmitir o pedido do utilizador para o servidor, para este ser processado. Esta tarefa será feita através do método `analisaEscolha`, que com o protocolo Cliente-Servidor, que está explicado a seguir, permite ao servidor saber o que o utilizador quer.

## 2.2 Protocolo Cliente-Servidor

Pedido (Cliente)	Resposta (Servidor)	Descrição
registarUtilizador <code>jose password</code>	registarUtilizador sucesso	Registado com sucesso
	registarUtilizador erro	Erro ao registar
iniciar <code>jose password</code>	iniciar sucesso	Iniciou sessão
	Iniciar erro	Não iniciou sessão
iniciarLeilao <code>Vendo citroen</code>	iniciarLeilao 1	Iniciou o leilão com id 1
listarLeiloes	listarLeiloes ( <code>informação dos leilões</code> )	Devolve todos os leilões separados pelo caracter ‘ ’
licitar 1 12000	licitar sucesso	Licitou com sucesso
	licitar valorBaixo	Valor muito baixo
	licitar erro	Erro ao licitar

terminarLeilao 1	terminarLeilao 1 quim 12000	O quim ganhou o leilão 1 por 12000
	terminarLeilao erro	Não foi possível terminar o leilão
erro		Ocorreu algum erro

Tabela 1- Protocolo

O protocolo entre o cliente e o servidor, baseado em texto e orientado à linha é o que podemos ver na tabela em cima. Na primeira coluna temos os pedidos que o cliente envia ao servidor, ou seja, mensagens que são enviadas da Thread ServidorSender para o Servidor, para que este analise. Este após interpretar os pedidos vai responder uma das seguintes mensagens que se encontram na segunda coluna, que serão enviadas para o cliente onde vão ser lidas pela *thread* ServidorReciever. Na terceira coluna da tabela temos uma breve descrição das possíveis respostas do servidor.

Nota: as palavras a azul claro significam que são variáveis.

### 3. Servidor

Neste tópico vamos detalhar como foi implementado a parte do servidor de modo a respeitar todos os requisitos apresentados no enunciado.

Inicialmente vamos começar por explicar quais foram as classes criadas e qual o seu papel na representação do problema em questão. Em seguida vamos falar das *threads* que criamos de modo a permitir que os vários clientes se conectem e interajam com o servidor. É também explicado como é feito o controlo de concorrência das várias *threads* e como o servidor analisa e gere as respostas a enviar aos clientes (por exemplo, quando um leilão termina).

Em seguida segue-se uma ilustração de como o servidor foi implementado e como é constituído. Esta ilustração servirá de apoio à compreensão da implementação ao longo dos restantes tópicos.

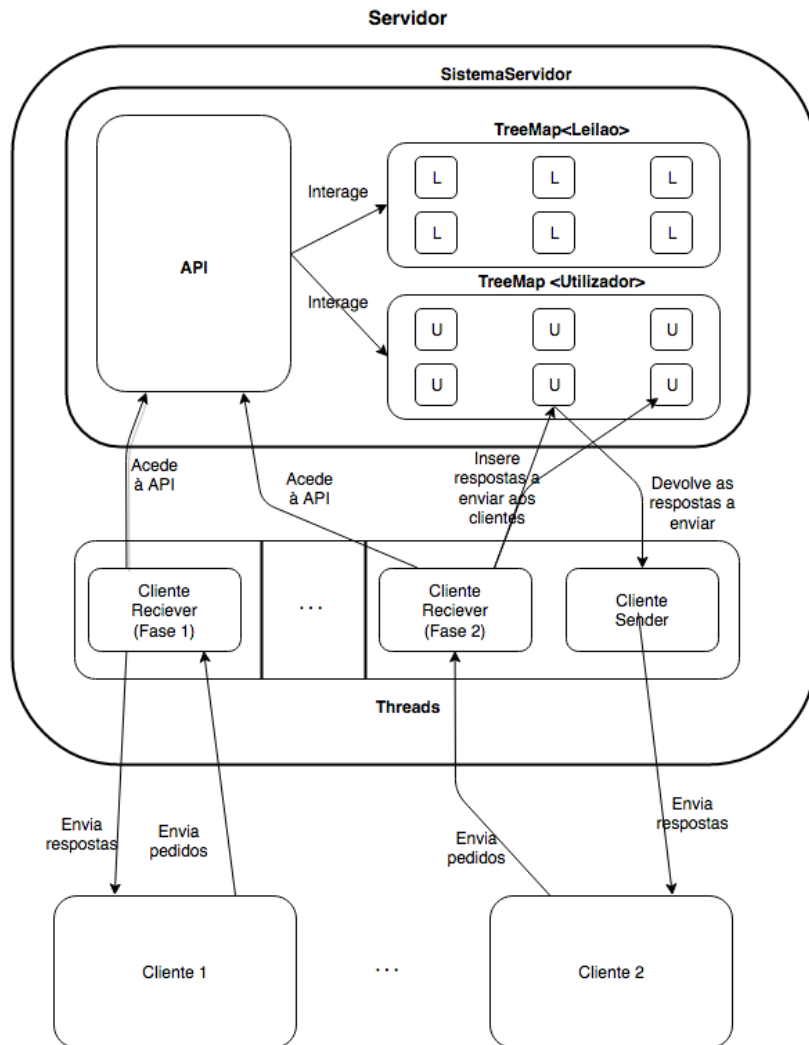


Ilustração 2 - Ilustração da implementação do servidor

### 3.1 Classes: SistemaServidor, Utilizador e Leilao

Em seguida vamos apresentar e explicar as classes do Servidor: SistemaServidor, Utilizador e Leilao:

#### 3.1.1 SistemaServidor

A classe SistemaServidor é a classe que gere todos os leilões e todos os utilizadores e é através da API desta classe que os vários clientes (*threads* no servidor) vão interagir com os objetos Leilao e/ou Utilizador.

Esta classe implementa a interface Sistema.java, interface esta que possui todos os métodos de modo a respeitar todos os requisitos apresentados no enunciado: registar utilizadores, iniciar sessão, iniciar um leilão, listar leilões ativos, licitar num leilão e terminar um leilão.

De modo a guardar todos os leilões e utilizadores foram criados dois *TreeMap*'s: um *TreeMap* <*String*, *Utilizador*> para os utilizadores em que a chave é o respetivo nome e um *TreeMap* <*Integer*, *Leilao*> para os leiloes em que a chave é o id do leilão.

Como todos os métodos desta classe operam sobre estes dois *TreeMap*'s e sobre os objetos contidos neles (Leilões e Utilizadores) e como várias *threads* vão estar constantemente a correr estes métodos é necessário controlar esta concorrência, garantido exclusão mutua na execução de cada um destes métodos. Deste modo foi instanciado um *ReentrantLock* que garante a exclusão mutua na execução das seções críticas (métodos).

### 3.1.2 Utilizador

A classe Utilizador é a classe que vai representar os utilizadores do sistema de leilões. Cada utilizador tem um nome e password para poderem se autenticar no sistema, além disso esta classe tem mais três atributos: um *Lock* e uma *Condition* associada a este *Lock* e um *ArrayList*<*String*> que representam as respostas do Servidor para o Cliente (utilizador). A utilização destes três últimos atributos irá ser melhor explicada mais à frente neste tópico 3.1.2 e posteriormente nos tópicos 3.2 *Threads*.

Esta classe possui um método *mayLogin*(*String* pass) que é responsável por responder ao SistemaLeiloes se é possível ou não aquele Utilizador realizar login, se for possível a thread marca a flag logado como verdadeira. Este método tem um papel importante pois é o responsável por não deixar que mais quem um Cliente esteja logado com o mesmo Utilizador.

Temos também os métodos *inserirResposta*(*String*) e o *lerResposta*(). O primeiro método é responsável por inserir uma resposta no *ArrayList* das respostas, enquanto que o segundo é responsável por devolver sempre a resposta mais antiga e remove-la do *ArrayList*.

Nestes métodos temos controlo de concorrência com *ReentrantLock*'s pois várias *threads* podem tentar executar o método *inserirResposta* (caso em que vários leilões terminam e é necessário avisar este utilizador desse termino) e também existe sempre uma *thread* que vai estar permanentemente a correr o método *lerResposta*() (*Thread* responsável por enviar as respostas ao Cliente deste Utilizador).

A *Condition* é a responsável por fazer adormecer a *thread* que está permanentemente a correr o método *lerResposta*() quando não existem mais respostas a enviar ao Cliente. Por outro lado, sempre que alguma *thread* executa o método *inserirResposta*() é realizado *signal*() para despertar, se necessário, a *thread* responsável por responder ao cliente.

### 3.1.3 Leilao

Esta classe é a classe mais simples do Servidor. As informações referentes a cada leilão que são necessárias guardar são as seguintes: o id do leilão, a sua descrição, um estado que determina se está ou não terminado, o dono do leilão, quem é o licitador maior e a respetiva quantia e todas as licitações que já foram feitas até então (é necessário guardar este histórico pois quando um leilão é terminado é necessário notificar todos os utilizadores que nele participaram). Desta forma, estes são os atributos desta classe.

Existe também um método desta classe que achamos relevante realçar neste relatório, o método *leiloar*. Este método é o responsável por representar o ato de leiloar por parte de um qualquer utilizador.

Nesta classe não temos qualquer tipo de controlo de concorrência porque os atributos e métodos desta classe apenas são acedidos pelos métodos do SistemaServidor, métodos esses

em que já é assegurado o controlo de concorrência, ou seja, não mais que uma *thread* ao mesmo tempo executa estes métodos ou acede aos atributos dos leilões.

## 3.2 *Threads*

Em seguida será explicado o papel das duas *threads*: ClienteReceiver e ClienteSender. A primeira *thread* executa dois papeis: um antes do cliente ter feito login em que recebe pedidos e envia respostas ao cliente, e outro papel de apenas receber pedidos por parte do cliente após este ter feito login com sucesso. A outra *thread* (ClienteSender) apenas é executada quando um cliente se autenticou no servidor e é responsável por enviar todas as respostas ao cliente em questão.

Em seguida será explicada com mais pormenor o funcionamento destas duas *threads*:

### 3.2.1 ClienteReceiver

A *thread* ClienteReceiver é a *thread* que é executada quando um Cliente se conecta ao servidor.

Tal como mencionado no tópico anterior, esta *thread*, executa dois papeis: um quando o cliente ainda não se autenticou (apenas envia pedidos ao servidor de registar utilizador ou realizar login) e outra após um utilizador estar autenticado (o utilizador envia pedidos para realizar as várias tarefas como licitar, encerrar leilão, etc).

No primeiro estado da *thread* esta recebe os pedidos do cliente, executa-os no SistemaServidor e responde ao cliente. Quando um cliente se autentica com sucesso, esta *thread* cria uma nova *thread* ClienteSender e passa a executar o segundo estado em que apenas recebe os pedidos do cliente e a resposta é feita através da nova *thread* criada.

Na execução do segundo estado desta *thread*, ela está constantemente à espera de pedidos vindos por parte do cliente (através do *socket*). Quando um pedido chega, a *thread* analisa o pedido (através do protocolo já mencionado anteriormente no tópico 2.2 deste relatório) e executa o pedido no SistemaServidor. Em seguida insere a resposta deste pedido no ArrayList<String> do Utilizador. A *thread* ClienteSender é que será a responsável por enviar a resposta (através de *socket*) ao cliente.

### 3.2.2 ClienteSender

Cada objeto Utilizador que esteja logado está sempre a ser “observado” por uma *thread* ClienteSender.

Estas *threads* estão sempre a correr a função lerMensagem() à procura de uma resposta para enviar ao cliente. Sempre que esta função retorna uma mensagem a *thread* envia, através do *socket*, a resposta ao cliente. Se o cliente não tiver respostas para lhe enviar esta *thread* fica adormecida no método lerMensagem() e só é acordada quando forem adicionadas novas mensagens no ArrayList ou quando o cliente faz logout (em que é preciso acordar, sair do método, fechar a conexão e terminar a *thread*).



## 4. Conclusão

Este trabalho foi bastante importante e enriquecedor pois permitiu que os membros do grupo compreendessem e assimilassem melhor os conceitos da matéria como a programação concorrente e os sistemas distribuídos.

Compreendemos os mecanismos para a criação de *Threads* bem como o seu funcionamento, pois estas permitem a cooperação eficiente via memória. Também foi importante perceber a suspensão ou retoma da execução de secções críticas através das variáveis de condição e ainda os *Locks*, mecanismos estes que ajudam a resolver os problemas de exclusão mutua.

Abordamos e exploramos mais profundamente o modelo cliente-servidor e a comunicação via *Sockets* TCP e a criação de servidores *multi-threaded*, envolvendo estado partilhado e o respetivo controlo de concorrência.

A realização deste projeto permitiu-nos consolidar os conhecimentos adquiridos na unidade curricular.