FEUP Treasure Seek

Relatório Final

Sistemas Distribuídos

Mestrado Integrado em Engenharia Informática e Computação

João Dias Conde Azevedo up201503256@fe.up.pt
João Pedro Furriel de Moura Pinheiro up201104913@fe.up.pt
Leonardo Manuel Gomes Teixeira up201502848@fe.up.pt
Mariana Lopes da Silva up201506197@fe.up.pt

Docentes:
Pedro Alexandre Guimarães Lobo Ferreira Souto
Hélder Fernandes Castro

28 de Maio de 2018

Conteúdo

1 Introdução

No âmbito da unidade curricular de Sistemas Distribuídos, desenvolvemos uma aplicação do tipo Cliente-Servidor, seguindo o paradigma REST. Esta aplicação baseia-se na localização de artefactos distribuídos pela Faculdade de Engenharia. Semelhante a uma "caça ao tesouro", após encontrar um destes, o jogador deve resolver o enigma contido no artefacto em questão. A aplicação terá dois tipos de clientes, o administrador e o jogador, com as seguintes funcionalidades:

1. Cliente Jogador
   (a) Envia a resposta a um enigma e recebe mensagem de confirmação do servidor, isto é, se a resposta que inseriu está ou não correta.
   (b) Consulta o ranking com a sua pontuação atual
   (c) Pode subscrever a notificações relativas a novos tesouros escondidos
2. Cliente Administrador
   (a) Coloca enigmas em determinados locais no interior da Faculdade de Engenharia.

Interface do cliente jogador:

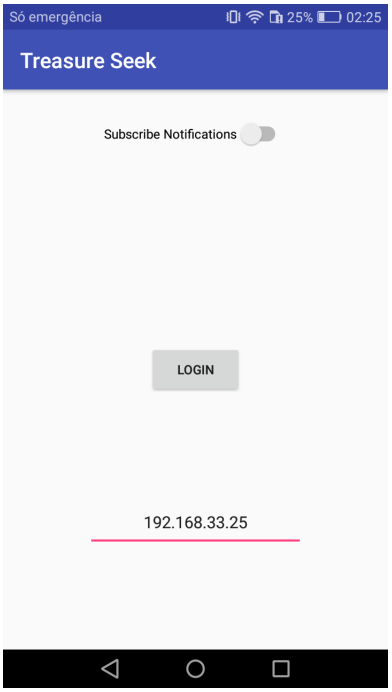

Figura 1: Login activity

Figura 2: Main Map activity

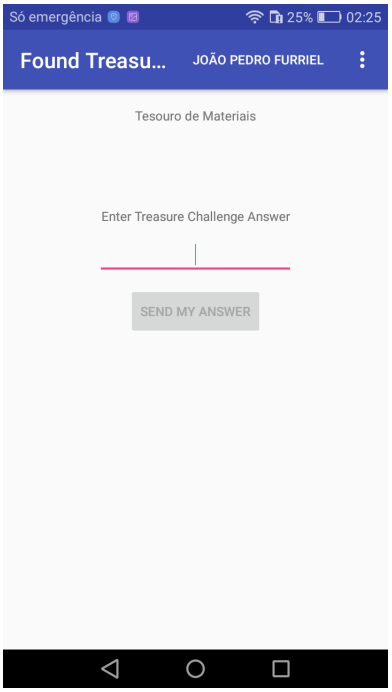

Figura 3: Answer activity

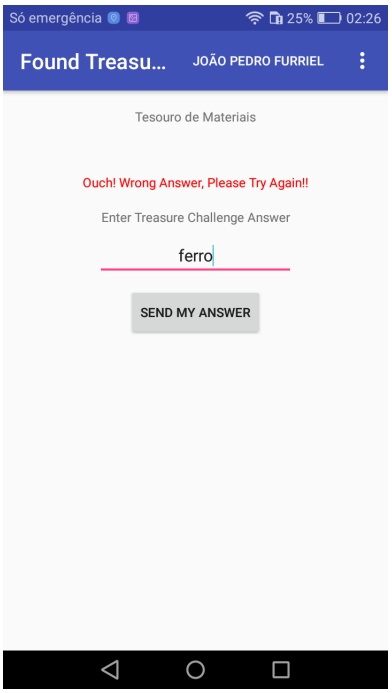

Figura 4: Wrong Answer

Figura 5: Correct Answer

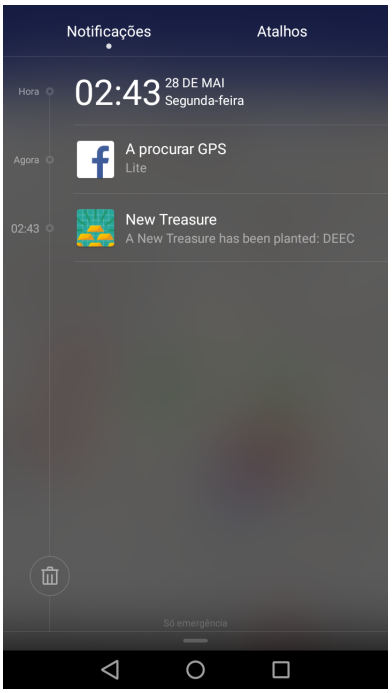

Figura 6: Notification

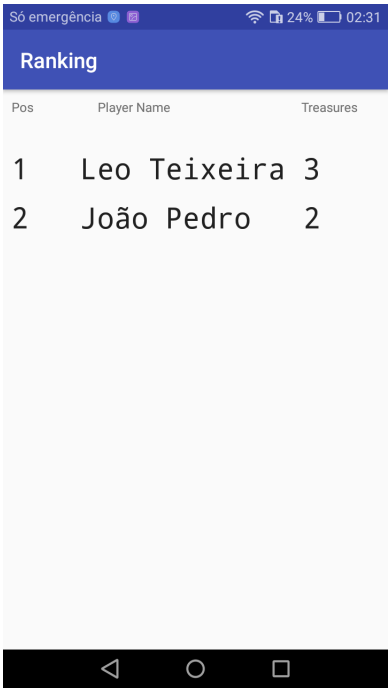

Figura 7: User Ranking

Interface do administrador (para além do login):

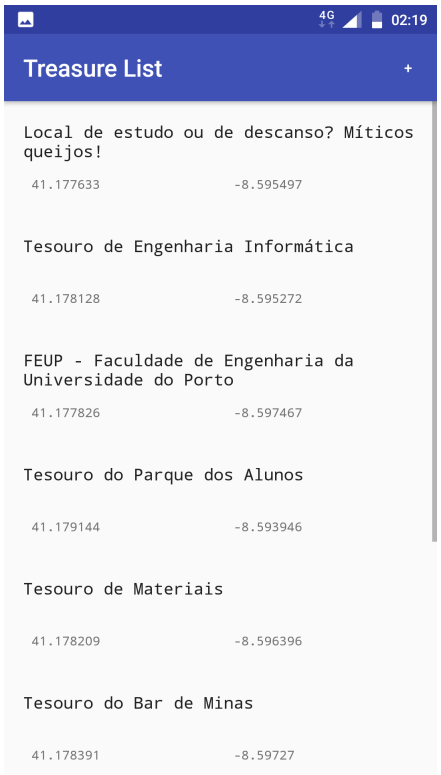

Figura 8: Treasure List

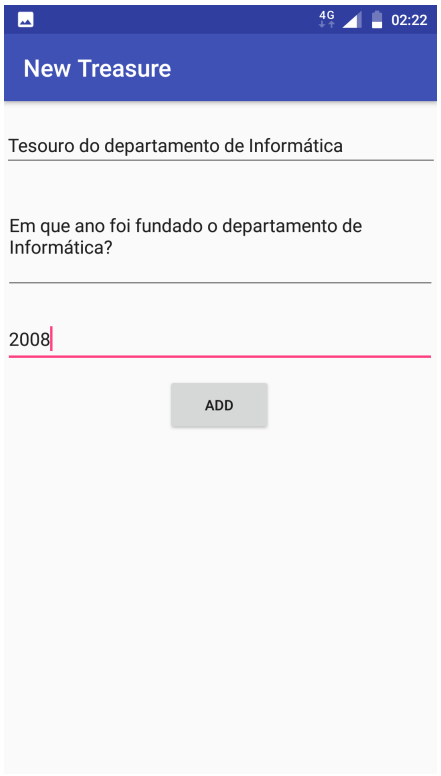

Figura 9: Add Treasure

De modo a especificarmos o modo de elaboração deste projeto a estrutura do relatório foi dividida em diversas secções, sendo que as que se seguem a esta introdução são:

1. **Arquitetura**, na qual são apresentados os principais componentes da aplicação, a sua interação, serviços fornecidos por terceiros e uma descrição detalhada dos protocolos de comunicação definidos entre os diversos componentes.

2. **Implementação**, onde se abordam os detalhes de desenvolvimento identificando as bibliotecas e estruturas externas utilizadas, bem como o porquê do seu uso, e uma descrição detalhada de como a concorrência é tratada entre cliente e servidores, bem como é feito o tratamento de mensagens.

3. **Questões relevantes**, onde serão abordados aspetos funcionais que consideramos importantes, bem como a descrição da estratégia de implementação desses aspetos com referência a extratos de código. Tais aspetos incluem a segurança nas nossas comunicações, a tolerância a falhas do sistema e a consistência dos dados entre diferentes componentes. Será também descrita a nossa implementação seguindo o paradigma REST.

4. **Conclusão**, na qual está presente uma avaliação crítica do trabalho e possíveis melhorias.

2 Arquitetura

2.1 Módulos principais e interações

A aplicação desenvolvida apresenta uma arquitetura complexa, tendo como principais componentes o **Cliente**, o **Servidor de Aplicação**, o **Servidor de Base de Dados** e o **Balanceador de Carga**.

Os referidos módulos dizem respeito, respetivamente, às classes java Client (no Android), AppServer, DBServer e LoadBalancer.

O cliente utiliza o nosso serviço através da aplicação Android desenvolvida para mobile.

O Balanceador de Carga é responsável por gerir a carga de trabalho de cada servidor de aplicação, distribuindo os diferentes pedidos dos diferentes clientes pelos mesmos, usando um algoritmo simples de distribuição. O balanceador de carga possui uma lista de servidores de aplicação, atualizada cada vez que um novo servidor é iniciado ou que um já existente é desligado. Para selecionar, escolhe aquele que está no início da lista, passando-o logo de seguida para o fim da mesma (Round-robin). Depois de feita a seleção do servidor de aplicação, o balanceador de carga comunica ao cliente qual o endereço IP e a porta do servidor de aplicação que será responsável por atender o seu pedido.

Cada servidor de aplicação recebe a lista de servidores de base de dados existentes aquando do seu lançamento. Escuta e processa pedidos de clientes e comunica com apenas um servidor de base de dados, obtendo confirmação do sucesso da operação e eventuais dados necessários para responder ao cliente.

Cada servidor de base de dados é responsável por ler ou escrever nas bases de dados e responder adequadamente aos servidores de aplicação, replicando a informação pelos restantes servidores de base de dados, mantendo assim a consistência das mesmas. Relativamente ao armazenamento de dados está ser utilizado um modelo de base de dados relacional, gerido pelo SQLite.

A acrescentar ainda que todas as comunicações, incluindo em RMI, são encriptadas com mecanismo de chaves assimétricas baseadas em SSL, excetuando a comunicação entre o cliente e o balanceador de carga e as notificações efetuadas pelo servidor aos clientes subscritos a tais.

2.2 Funcionamento

O processamento típico do sistema envolve um pedido feito pelo cliente. Primeiramente comunica com o balanceador de carga, pedindo um servidor de aplicação disponível. Este, envia ao respetivo cliente a indentificação do servidor, ou seja, um IP e uma porta. De seguida, o cliente envia uma mensagem ao servidor de aplicação anteriormente escolhido. Este servidor é responsável por escolher um servidor de base de dados e comunicar a intenção do cliente, retornando para o mesmo uma mensagem de sucesso no fim da operação, podendo esta conter informação caso o pedido do cliente tenha sido para leitura de dados. O servidor de base de dados é responsável por efetuar a leitura ou escrita na

sua base de dados, conforme o pedido do cliente e, caso seja uma operação que a altere, replicá-la pelos restantes servidores de base de dados. Retorna para o servidor de aplicação a confirmação de sucesso da operação (ou não) e os dados desejados pelo cliente, se houver.

O procedimento descrito acima é regularmente efetuado. Porém, é por vezes minimamente diferente. No caso do login na nossa aplicação, o cliente comunica primeiramente com o facebook, onde faz login e recebe um token de sessão. Esse mesmo token é posteriormente enviado aos servidores de aplicação e base de dados para validação do mesmo com o facebook. Não só mas também, o procedimento é diferente no caso das notificações, onde ao ser inserido um novo tesouro por um administrador o servidor de aplicação encarrega-se de notificar os clientes subscritos às mesmas.

Para uma melhor compreensão, o esquema apresentado abaixo ilustra as principais estruturas utilizadas nesta arquitetura e respetivas relações.

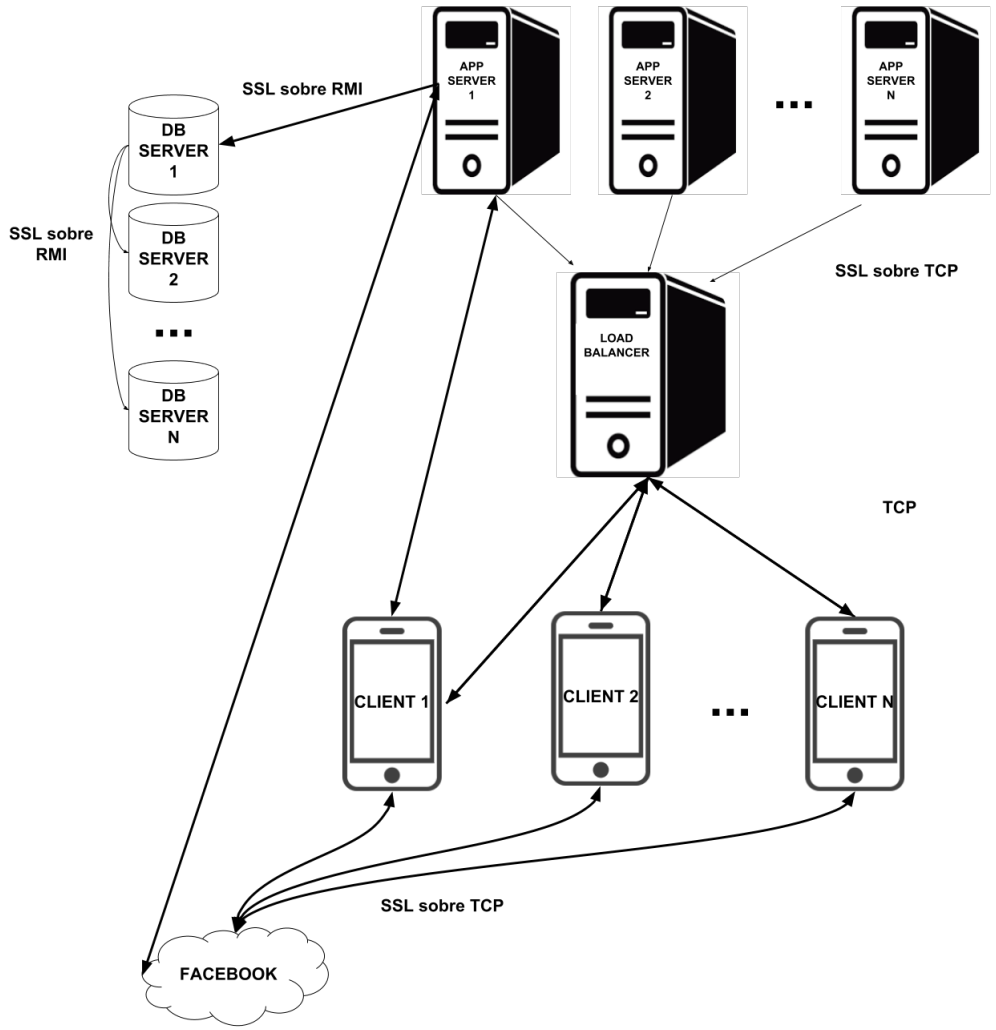

Figura 10: Esquema representativo da arquitetura do projeto

2.3 Protocolo de Comunicação

O protocolo de mensagens é definido no package communications, nos ficheiros Message.java e ReplyMessage.java.

O protocolo de mensagens assenta no paradigma REST. Assim, cada mensagem possui uma ação, assegurando operações CRUD (CREATE, RETRIEVE, UPDATE, DELETE), e um identificador de recurso, por exemplo TREASURE, USER ou FOUND_TREASURE.

A estrutura geral de uma mensagem de pedido será do tipo:
<ACTION><RESOURCE><BODY>

Por outro lado, a estrutura de uma mensagem de resposta será do tipo:
<STATUS><BODY>

O <STATUS>pode ser do tipo OK, UNAUTHORIZED, RESOURCE_NOT_FOUND ou BAD_REQUEST.

Para a construção do <BODY>é usada uma biblioteca externa que permite criar e manipular JSONArray's e JSONObject's.
Assim, numa mensagem de pedido o <BODY>é um JSONObject e numa mensagem de resposta será um JSONArray.

2.3.1 Sub-Protocolo Cliente/Servidor de Aplicação

Este protocolo foi definido para implementar a comunicação entre o cliente e o servidor de aplicação.

De uma maneira geral as mensagens do cliente para o servidor tem a estrutura:
<ACTION><RESOURCE><BODY>
sendo que o BODY é opcional e apenas usado nas operações de inserção de um RESOURCE

Por outro lado, as mensagens do servidor para o cliente tem a estrutura:
<STATUS><BODY>
sendo que o BODY é opcional e apenas usado nas operações de leitura de um RESOURCE

Como exemplo de um possível par de mensagens pedido e resposta temos:

Pedido: UPDATE USER/44 {"name": "John Doe"}
Resposta: OK

As ACTION disponíveis neste sub-protocolo são CREATE, RETRIEVE, UPDATED e DELETE bem como duas especiais: LOGIN e LOGOUT.

2.3.2 Sub-Protocolo Cliente/Balanceador de Carga

Este protocolo foi definido para implementar a comunicação entre o cliente e o balanceador de carga.

A única mensagem enviada do cliente para o balanceador é
RETRIEVE_HOST à qual a resposta é

OK {"host": 172.30.0.88, "port": 2500}

ou, caso não haja nenhum servidor de aplicação online,

BAD_REQUEST

2.3.3 Sub-Protocolo Balanceador de Carga/Servidor de aplicação

Este protocolo foi definido para implementar a comunicação entre o balanceador e o servidor de aplicação.
As mensagens são unidirecionais, apenas do servidor para o balanceador, sendo que estas podem ser ou
NEW_SERVER {"host": 172.30.0.88, "port": 2500} ou
SHUTDOWN_SERVER {"host": 172.30.0.88, "port": 2500}

Em baixo explicitamos as mensagens trocadas no nosso sistema:

1. **LOGIN** para dar entrada de um novo utilizador. Esta mensagem é enviada pelo cliente para o servidor de aplicação contendo como recurso USER e enviando os dados necessários para login.

2. **LOGOUT** para a saída de um utilizador. Esta mensagem é enviada pelo cliente para o servidor de aplicação contendo como recurso USER.

3. **CREATE** para criar um novo tesouro ou quando o utilizador encontra um tesouro, dependendo do tipo de recurso. No caso de inserção de novo tesouro a mensagem será do tipo CREATE TREASURE. Caso um novo tesouro seja encontrado a mensagem será do tipo CREATE FOUND_TREASURE.

4. **RETRIEVE** para obter os tesouros encontrados pelo utilizador e o respetivo ranking. No caso de obtenção de todos os tesouros a mensagem será do tipo RETRIEVE TREASURE e no caso de rankings será do tipo RETRIEVE USER.

5. **RETRIEVE_HOST** para selecionar o servidor de aplicação responsável por atender o serviço. Esta mensagem é enviada pelo cliente para o balanceador de carga.

6. **NEW_SERVER** para informar o balanceador de carga da entrada de um novo servidor de aplicação. Esta mensagem é enviada pelo servidor de aplicação para o balanceador de carga.

7. **SHUTDOWN_SERVER** para avisar que o servidor de aplicação já não está disponível para atender pedidos. Esta mensagem é enviada pelo Servidor de Aplicação para o balanceador de carga.

3 Implementação

3.1 Módulos

A componente cliente é destinada a dispositivos mobile para Android e foi implementada em Java no Android Studio, usando como API's externas a Facebook Graph API e a Maps Android API. A Facebook Graph API permite ao utilizador fazer login na aplicação, tornando cada utilizador como único. Através da Maps Android API, foi possível obter o mapa da Faculdade de Engenharia e atualizar em tempo real a posição do utilizador.

As componentes dos servidores e balanceador de carga são aplicações implementadas em Java, para PC/Mac. A componente do servidor de base de dados utiliza como biblioteca externa o SQLite para armazenamento dos dados. O grupo resolveu escolher esta biblioteca externa não só pela sua performance e utilidade prática a ler e escrever num banco de dados, geralmente mais rápido do que ler e gravar ficheiros individuais no disco, como também pela confiabilidade num sistema menos provável de falha do que leituras e escritas em ficheiros customizados. Para além das razões mencionadas, o código é assim mais portável entre sistemas operativos e o conteúdo pode ser acedido e atualizado usando query's SQL concisas, em vez de rotinas processuais longas e propensas a erros. Outra razão da escolha reside no facto de vários programas, implementados em diferentes linguagens de programação, poderem aceder ao mesmo ficheiro da aplicação sem preocupações de compatibilidade.

De forma global, quer no lado do cliente quer no lado dos servidores, é também usada uma biblioteca externa que permite armazenar a informação numa estrutura JSON, sendo possível criar JSONArrays e JSONObjects, facilitando e definindo uma estrutura rígida de transmissão de dados nas mensagens.

3.2 Concorrência e processamento de mensagens

De forma a gerir a concorrência e a evitar que os pedidos do clientes não sejam atendidos, o sistema é capaz de possuir várias réplicas quer dos servidores de aplicação quer dos servidores de base de dados do sistema e da entidade balanceador de carga para regular a carga dos servidores de aplicação.

Cada porto de escuta, quer no cliente quer nos servidores, é processado numa thread à parte, não bloqueante, sendo o processamento de cada mensagem feito numa outra thread, de forma a rapidamente processar cada pedido e rapidamente responder ao mesmo.

4 Questões relevantes

4.1 Tolerância a falhas

Com a utilização de múltiplos servidores, quer os de aplicação, quer os de base de dados, pretendemos que case um ou outro falhe existam servidores capazes de atender os clientes. Assim, com várias réplicas do estado do sistema temos maior redundância, evitando assim que caso um servidor se desligue os clientes sejam atendidos e os seus dados mantidos.

Efetivamente, a questão da tolerância a falhas foi uma preocupação primária do grupo e intensivamente testada, e orgulhamo-nos de poder dizer que, de facto, o sistema é bastante robusto, aceitando a disconexão de servidores de base de dados e servidores de aplicação.

Não só mas também, é possível re-iniciar um servidor de base de dados, cujo o estado é de extrema importância, e sincronizar o mesmo com o restante sistema.

4.2 Consistência

Com o uso de múltiplos servidores de base de dados existe a possibilidade de inconsistência entre o estado mantido por cada um destes. Assim, para garantir que tal não acontece, desenvolvemos um protocolo que garante que quando é efetuada uma alteração num servidor, a mesma é replicada pelos restantes. No package main, na classe DBServer.java, a função replicateData() é responsável por replicar a informação de um servidor de base de dados para todos os outros.

4.3 Segurança

Para troca de mensagens de forma segura implementamos um sistema baseado em chave pública e privada, com base no protocolo SSL usando a API de Java JSSE (Java Secure Socket Extension). Excetuam-se a comunicação entre o cliente e o balanceador de carga não é protegida, porque apenas é trocada a informação do IP e da porta do servidor de aplicação, e as notificações dos servidores aos clientes subscritos. A comunicação entre o cliente e o servidor de aplicação é encriptada com mecanismo de chaves assimétricas, tal como pode ser visto na classe LoadBalancer.java na função serverDispatcher() e na classe AppServer.java nas funções announceToLB() e kill(). O mesmo acontece na comunicação em RMI entre o servidor de aplicação e o servidor de base de dados, na classe AppServer.java na função chooseDB() e na classe DBServer.java na função initRMIInterface() e na comunicação, também em RMI, entre os servidores de base de dados na classe DBServer.java, na função replicateData(). A comunicação entre o cliente e o servidor de aplicação também é protegida, e pode ser percecionado na classe UserController.java, na funções loginToTreasureSeek(), logoutFromTreasureSeek(),sendFoundTreasure(), insertNewTreasure(), requestAllTreasures() e getRanking().

4.4 Escalabilidade

Com esta arquitetura conseguimos aumentar o número de servidores, quer de aplicação quer de base de dados, em tempo de execução, e assim servir mais clientes, dependendo apenas da infra-estrutura (hardware).

4.5 Notificações

Os clientes subscritos recebem notificações do servidor de aplicação quando há a inserção de um novo tesouro por um administrador. No ficheiro AppServer.java, a thread NotifyClient() da linha 563 à 571, é responsável por enviar uma notificação ao cliente.

5 Conclusão

A realização desta aplicação, do tipo Cliente/Servidor com notificações, concorrente, tolerante a falhas e consistente, baseada em múltiplos threads, contribuiu não só para a consolidação dos conceitos leccionados, como também para compreender ainda melhor o modo de funcionamento de um sistema distribuído. A notar a importância do docente no esclarecimento de dúvidas que iam surgindo no decorrer do trabalho.

Uma das possíveis melhorias a implementar neste projeto seria tratar um caso de concorrência em que, por exemplo, dois admins inseririam o mesmo tesouro.

Assim e analisando o trabalho realizado, considera-se que o objetivo foi cumprido, pois foram desenvolvidas todas as exigências propostas inicialmente, bem como foram considerados aspetos como a segurança, a tolerância a falhas e consistência.