Trabalho Prático 2

Sistemas Distribuídos

João Ricardo Lourenço, Nº 2011151194
Joaquim Pedro Bento Gonçalves Pratas Leitão, Nº 2011150072

8 de Dezembro de 2013

Relatório

1 Introdução

No presente projecto pretende-se criar uma aplicação responsável pela gestão e organização de ideias. Nesta aplicação, diferentes utilizadores podem efectuar o seu registo e autenticação, criando ideias que, estando organizadas por tópicos, são comercializadas num mercado interno à aplicação.

A fim de permitir a comercialização das ideias criadas pelos utilizadores, cada ideia possuí 100 000 *shares*, que serão transaccionadas no mercado de ideias disponibilizado na nossa aplicação.

O processo de criação de uma ideia requer um investimento por parte do seu criador, sendo que o seu valor inicial no mercado está directamente relacionado com o valor investido pelo utilizador. Sempre que o utilizador pretender, ou sempre que comprar *shares* de uma ideia, pode definir um novo valor pelo qual pretende disponibilizar as *shares* no mercado.

De entre todos os utilizadores existe um conjunto reduzido de utilizadores com permissões especiais, os utilizadores *root*, que podem a qualquer instante adquirir todas as shares de uma ideia ao preço da sua última transação, retirando essa ideia do mercado. Uma vez retirada do mercado, uma ideia é colocada na *Hall of Fame* da aplicação, onde poderá ser consultada por todos os utilizadores da aplicação.

2 Arquitectura Interna Web-Based

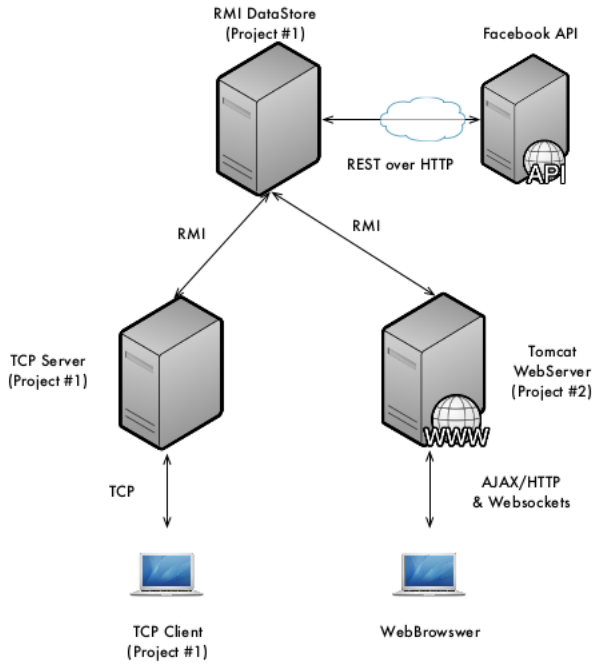

Figura 1: Arquitectura Interna do Serviço

2.1 Introdução

Para o presente projecto foram realizadas algumas alterações ao servidor RMI desenvolvido para o projecto anterior, bem como desenvolvido um servidor web, responsável por receber pedidos de clientes web (browsers).

Adicionalmente, a aplicação apresenta integração com a API, *Application Programming Interface*, oficial da rede social *Facebook*, permitindo a um utilizador realizar um registo com a sua conta da rede social ou alternativamente, associar a uma conta já criada na nossa aplicação a sua conta do *Facebook*.

Assim, sempre que este criar ou comprar *shares* de uma ideia será realizada por ele uma publicação na rede social, indicando a acção realizada. Da mesma forma, se o utilizador apagar uma ideia que aquando da sua criação foi publicada na rede social, essa mesma publicação será removida da rede social.

À semelhança da aplicação desenvolvida para o projecto 1, estão ainda disponíveis o cliente e o servidor TCP, que permitem a um utilizador autenticar-se e criar e transacionar ideias.

Com esta estrutura e aplicações apresentadas, pretendemos disponibi-

lizar soluções multi-plataforma para a organização e validação de ideias, oferencendo aos seus utilizadores métricas para a comparação da importância e influência de diferentes ideias, através do seu valor monetário e da sua eventual presença na *Hall of Fame* das ideias do sistema.

2.2 Estrutura Interna

A arquitectura interna do Servidor Web assenta num modelo MVC (Model, View Controller), suportado pelo framework Struts2 (da Apache Foundation). Utilizando as funcionalidades de Java Servlets, este framework simplifica a tarefa de interligar componentes, oferecendo um ambiente de programação eficiente e rápido.

Por forma a obter uma clara superação das diferentes lógicas (Presentation, Business e Data), o modelo MVC é utilizado, sendo de certa forma até imposto pelo Struts2.

2.2.1 View

A View está materializada em ficheiros JSP que utilizam HTML, CSS, Javascript, jQuery e Boostrap para apresentar a informação no ecrã. Esta informação, frequentemente dependente da acção requerida ou dos dados de um dado cliente, é armazenada em java beans e que são acedidos através de Tags JSTL/Struts2. Deste modo, procurou-se minimizar o código nas views e movê-lo para o lugar onde pertencem (O Controller/as Actions do Struts2).

Destacamos o facto de utilizarmos AJAX, com JSON para proceder a actualizações dinâmicas da página, tendo tido o cuidado de dividir também esta lógica em ficheiros javascript carregados à parte das páginas, não sendo demasiado intrusivos na view.

A nossa View é responsive, algo que é garantido pela framework Bootstrap e, para além de beneficiar do mecanismo Push-To-Client brevemente descrito, sofre alterações a parte do seu conteúdo com base nas tecnologias AJAX que utilizamos. Por exemplo, ao comprar shares de uma ideia somos notificados em resposta (por JSON) de quantas shares temos dessa ideia, entre outras alterações. No entanto, os mecanismos de AJAX implementados são essencialmente para não forçar o utilizador a ter de alterar de View (JSP) frequentemente, o que também nos possibilitou focar o desenvolvimento mais em código e menos em design.

2.2.2 Controller

O Controller está materializado em Actions (métodos de classes em Java) invocadas pelo Struts (através de um pedido do utilizador) e que podem obter parâmetros via GET ou POST e definir parâmetros diversos das Views.

Por exemplo, para obter a listagem de ideias, uma Action é despoletada com o tipo de listagem que se requer e, após obter estas Ideias, armazena-as numa estrutura de colecção acessível à View. Na View, utilizando tags JSTL, a colecção é iterada e o código HTML/JS/CSS é gerado dinamicamente. No Controller não há qualquer lógica de apresentação, apenas de negócio.

Existe, pois, um trabalho de algum parsing dos dados inseridos pelo utilizador, bem como de verificação da validade da sua sessão, para enviar os dados para o Model.

2.2.3 Model

Materializado em Java Beans auto-contidos. O Controller interage com o Java Bean para efectuar alterações no seu estado interno. Por exemplo, existe um método login(), do Bean, que altera o estado deste para reflectir o utilizador que acabou de se autenticar.

Este Model acaba por agir como uma camada de ligação ao servidor RMI onde os dados estão verdadeiramente armazenados e a lógica de acesso à Base de Dados (e verificação de consistência, manutenção de filas de espera, etc) é tratada.

O único Java bean existente no projecto é um Bean que contém as informações do cliente (userid, username, username no facebook, estado do login...) e cuja maior parte dos métodos apenas redirecciona para métodos equivalentes do RMI. Este Bean é armazenado na sessão para ser acessível entre vários pedidos e representa o núcleo duro da representação de uma sessão do cliente. O próprio acesso ao RMI é gerido por este bean, sendo totalmente desconhecido do Controller, uma vez que mesmo que a Base de Dados e o servidor RMI fossem trocados por uma outra tecnologia equivalente, a interface de programação do Controller se manteria intacta.

Torna-se claro que existe muita Business Logic no nosso model, e em particular no nosso local de armazenamento de dados (o RMI). No entanto, a justificação para este facto é simples: esta é uma aplicaçãp focada nos dados (Ideias e Shares) e na manipulação destes (criação, compra e venda). Numa outra aplicação menos "data-centric", alguma desta lógica poderia ser movida para fora dos nossos dados e colocada em camadas mais superiores, como as servlets.

2.2.4 Conclusão

Assim, a nossa View está programada em JSP, com Javascript (autocontido em ficheiros próprios), CSS, jQuery, as frameworks Bootstrap e Noty, dependendo apenas da existência e de uma interface bem-definida do Model e de alguns parâmetros do Controller (por exemplo, no já referido caso da iteração de ideias, é necessário que esta colecção exista). A alteraçã. Os noss

Controllers estão programados inteiramente em Java e interactuam com o Bean, responsável por fazer a ponte com o servidor RMI, isto é, com o armazenamento dos dados.

Relativamente à integração com o Facebook, recorrendo à API do Facebook, a maior parte da lógica é mantida no lado de armazenamento dos dados. Considera-se que 'um post no Facebook" pertence ao modelo de dados e deve, portanto, estar isolado da View e do Controller.

Relativamente aos Websockets, é utilizada a API de Tomcat 7 (agora em desuso) em Java e em Javascript.

3 Integração com o Projecto 1

As alterações nos requisitos entre projectos e até as diferentes interpretações que existiram do projecto original levaram a alterações profundas a nível de funcionamento interno. O Projecto 1 apresentava uma quantidade significativa de Business Logic no servidor TCP e que acabou por ser desenvolvida de novo, no servidor RMI e de maneira diferent do que originalmente. Tais alterações tornaram o projecto inicial virtualmente obsoleto, permitindo apenas que a comunicação entre o servidor e o cliente TCPs se mantivesse sem grandes alterações. A comunicação entre servidor TCP e RMI, bem como a maioria das funções internas sofreram, pois alterações.

Cingimo-nos apenas aos requisitos exigidos na checklist para minimizar o nosso esforço de adaptação do código. Se, por um lado, a nossa interpretação original do enunciado era totalmente distinta no que dizia respeito a adquirir shares, por outro, a quantidade de alterações do lado do servidor TCP forçar-nos-ia a desenvolver quase dois projectos. Assim, o cliente TCP apenas suporta Login, Comprar Shares, Criar e Ver Ideias.

Relativamente à funcionalidades mantidas, como referido, o protocolo TCP mantém-se praticamente inalterado (alguns inteiros foram alterados para floats...e um conjunto de parâmetros no que diz respeito á relação entre ideias foi retirado).

O Servidor, que possuía um mecanismo interno de filas de espera, foi totalmente modificado e readaptado para o novo servidor RMI.

Como as notificações e a maior parte da business logic está centrada no servidor RMI, funcionam de forma transparente, independentemente da origem dos pedidos.

4 WebSockets

Para os Websockets foi utilizada a API do Tomcat 7, actualmente em desuso.

4.1 Servlet adicional

Foi criada uma servlet à parte, independente do Struts2, que também interage com o RMI. Para, na servlet dos Websockets, poder saber qual o cliente que acabou de se ligar, de forma segura, implementamos um mecanismo recorrendo a cookies e passando informação gerada no Struts2.

4.2 Interligação com Struts2 (manutenção da sessão activa)

É gerada uma hash interna (Um Encoded UID), no Java bean do Cliente, armazenada pelas Actions do Struts2 em cookies. Quando um Websocket é aberto e um novo cliente é adquirido, a Servlet dos websockets acede aos cookies do cliente e lê este Encoded UID, procedendo ao seu mapeamento aos UIDs armazenados no servidor RMI. Assim, a única forma de este sistema ser de algum modo "enganado" será se um cliente mal-intencionado tiver acesso aos cookies (ou aos dados neles armazenados, i.e: ao Encoded UID) de outro utilizador.

5 Push-to-client (RMI Callback) / Notificações

Para implementar mecanismos push-to-client (vastamente superiores na maior parte das situações a mecanismos client-triggered como polling), recorremos a RMI Callbacks.

Aquando de um início bem-sucedido da ligação com um cliente (i.e., após correcta verificação do Cookie), é criado um Bean equivalente ao que existe no Struts2. Esta "cópia"/"duplicação" do Bean permite melhorar a manutenção do código, pois teríamos de reimplementar funcionalidades no que diz respeito à interligação ao RMI. Com este Bean, é criado um objecto remoto que vai fazer a ponte entre o Websocket (Inbound) e o servidor RMI. Este objecto remoto é passado (i.e: registado) ao servidor RMI por forma a ser mapeado a um user id numa hashtable. Assim, o RMI possui, de forma eficiente, um mapeamento entre cada userid e o seu websocket mais recente.

Quando se torna necessário notificar um cliente, o servidor RMI itera esta hashtable, verifica se o objecto remoto (criado na Servlet dos websockets) ainda existe e, se existir, invoca métodos remotos que vão levar ao envio de mensagens (por parte da servlet) por websockets. Estas notificações são recebidas do lado do cliente web e interpretadas em javascript.

Utilizamos o mecanismo Push-to-client para transmitir actualizações dos valores de mercado das ideias (preço da venda mais recente) e para

transmitir diversas notificações.

6 Rest Web Service

Para tornarmos a nossa aplicação mais relevante nos dias de hoje, interligámos esta com a *Application Programming Interface* pública do Facebook, que utiliza *REST (REpresentational State Transfer)*.

Esta API pressupõe a criação e registo de uma aplicação no Facebook, a qual pode ser utilizada e interligada com a nossa aplicação.

Assim, sempre que um utilizador se regista na nossa aplicação com uma conta do Facebook, ou sempre que associa à sua conta da aplicação uma conta do Facebook, é-o facebook encarrega-se de lhe fazer um pedido para conceder as permissões necessárias (que especifcamos em javascript) para que a aplicação por nós criada no Facebook possa realizar a actividade prevista em nome do utilizador.

Sempre que um utilizador se pretende autenticar na nossa aplicação com as suas credenciais do Facebook, é estabelecida uma ligação aos servidores do Facebook, para que se confirme a identidade do utilizador em questão, sendo-nos permitido acesso a todas as funcionalidades referidas anteriormente.

Uma vez efectuado com sucesso a autenticação do utilizador no Facebook, é-nos fornecido um *Access Token*, que concede à nossa aplicação um acesso seguro e temporário às acções para as quais esta obteve permissão da parte do utilizador.

Quando um utilizador pretende registar uma nova conta com as suas credenciais do Facebook, ou quando pretende associar uma conta já existente à sua conta no Facebook, é utilizada a API da rede social, em Javascript que, dadas as permissões solicitadas ao utilizador, se encarrega de abrir (ou não) uma nova janela de forma a autenticar a aplicação com o utilizador e providenciar a nossa aplicação com um *Access Token*. Este *Access Token* é em seguida utilizado para, utilizando *OAuth*, ser efectuado um pedido ao Facebook, de forma a validar a identidade do utilizador que se acabou de registar. Como resultado deste processo, o Facebook fornece à nossa aplicação um conjunto de dados acerca do utilizador em questão, dos quais armazenamos na base de dados da nossa aplicação o identificador único (ID) do utilizador no Facebook.

O processo anteriormente descrito é também utilizado quando o utilizador pretende associar a sua conta na nossa aplicação a uma conta válida do Facebook, bem como quando este pretende efectuar a sua autenticação na nossa aplicação utilizando as suas credencias do Facebook.

Sempre que um utilizador realiza a sua autenticação perante a rede social, o *Access Token* recebido é armazenado na sua sessão, sendo utilizado sempre que a aplicação pretende realizar acções em nome do utilizador.

Quando esse utilizador adiciona uma nova ideia ao sistema a nossa aplicação recorre a OAuth e ao *Access Token* do cliente para enviar um pedido à rede social que visa efectuar uma publicação na *feed* de notícias do

utilizador em questão. Para isso, e tal como requerido pela API, é fornecido o *Access Token* do utilizador em questão (e que foi previamente armazenado na sua sessão aquando da sua autenticação), o endereço onde queremos efectuar a publicação (e que neste casos será o endereço do mural do utilizador no Facebook) e a mensagem a publicar. Ao fornecermos estes parâmetros através de um pedido REST POST estamos a indicar à API do Facebook que a acção por nós pretendida é a da criação de uma nova publicação.

Como resposta a este pedido, a API do Facebook fornece-nos um identificador único da publicação realizada, que será armazenado na base de dados do sistema, a fim de podermos, futuramente, realizar comentários à publicação criada, ou simplesmente apagar a publicação.

Sempre que um utilizador compra *shares* de uma ideia que, aquando da sua criação levou a uma publicação no Facebook, a nossa aplicação realiza um processo semelhante ao anteriormente descrito, no sentido de realizar um comentário na publicação original da ideia. Nesta situação, é necessário fornecer os mesmos dados que na situação anterior, sendo de salientar que o endereço onde queremos efectuar a publicação é agora o endereço que nos permite aceder à publicação original da ideia (no qual está incluído o identificador único da publicação, previamente armazenado na base de dados da nossa aplicação).

Por fim, quando um utilizador detém a totalidade das shares de uma ideia e pretende eliminar essa ideia do sistema, caso se encontre autenticado com credenciais do Facebook, e caso a ideia que pretende apagar seja da sua autoria, a publicação na rede social, efecutada aquando da criação da ideia, será eliminada.

O processo de remoção de uma publicação do Facebook é bastante semelhante ao processo de criação de uma nova publicação na rede social, diferindo no facto do pedido realizado ser REST DELETE, enquanto no processo anterior o pedido era enviado por REST POST. Para além disso, nesta situação não é necessário enviarmos qualquer corpo da mensagem, dado que não pretendemos realizar uma publicação, mas sim remover uma publicação já realizada.

Nas diferentes acções descritas, caso o *Access Token* armazenado para o utilizador em questão não se encontre válido, ou caso o utilizador termine a sua sessão no Facebook, a acção em questão (criação de uma publicação, criação de um comentário a uma publicação ou remoção de uma publicação) não será realizada.

No entanto, em qualquer situação, a acção realizada pelo utilizador será mantida no sistema, uma vez que a sessão do utilizador se mantém.

Deste modo, recorrendo a um conjunto de primitivas GET, POST, PUT e DELETE simples (em HTTP), bem como ao processamento das respostas em JSON, somos capazes de criar interligação com um sistema totalmente independente do nosso. Trata-se, pois, de uma correcta articulação com um sistema verdadeiramente distribuído – a Internet.

7 Testes Realizados

Teste Realizado	Estado
Servidores executam em máquinas separadas	Passou
Registo de Utilizadores	Passou
Login de Utilizadores	Passou
Procura e Listagem de Ideias	Passou
Listagem de Tópicos	Passou
Criação de um novo Tópico	Passou
Criação de uma nova Ideia	Passou
Remoção de uma Ideia	Passou
Definir o preço de venda das shares de uma Ideia	Passou
Compra shares de uma Ideia	Passou
Root (Administrador) compra Ideias e coloca-as no Hall of Fame	Passou
Visualização do Histórico de Transações de um utilizador	Passou
Cliente e Servidor desenvolvidos no Projecto 1 permitem a criação de Ideias	Passou
Cliente e Servidor desenvolvidos no Projecto 1 permitem transação de Ideias	Passou
Associação de uma conta do Facebook a uma conta existente na aplicação	Passou
Registo com uma conta do Facebook na aplicação	Passou
Criação de novas Ideias dão origem a posts no Facebook	Passou
Remoção de Ideias originam a remoção dos respectivos posts no Facebook	Passou
Compra de shares de uma Ideia origina um comentário ao post no Facebook	Passou

Figura 2: Testes Realizados