FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Sistemas Distribuídos

Relatório do Projeto

"IdeaBroker – Idea Management and Trading"

Trabalho realizado por:

Andreia Sofia Oliveira Cruz, nº2011154038

Bruno José Borges Madureira, nº2011161942

Arquitectura Interna Web

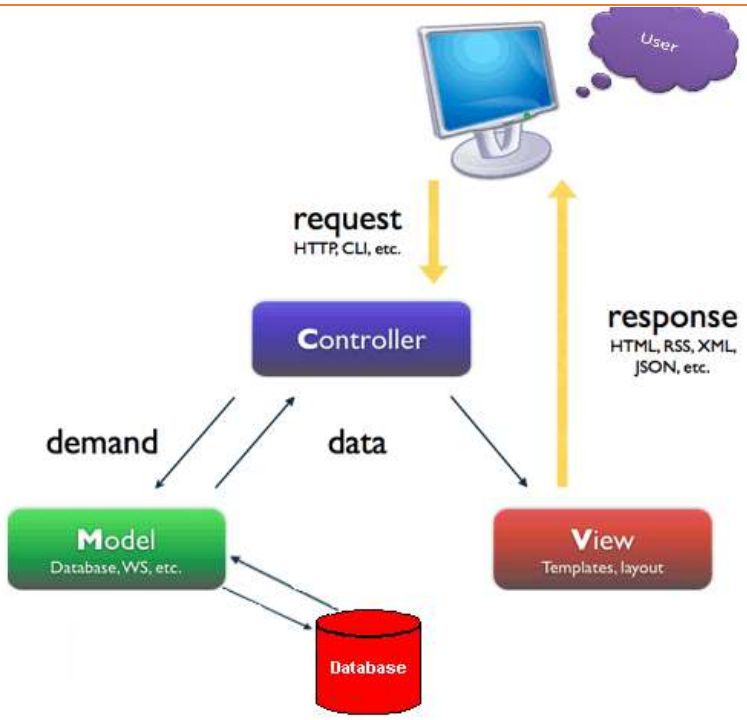

Figura 1 – Exemplo de arquitetura MVC

Como se pode observar na figura 1 utilizámos uma arquitectura **MVC** (Model-View-Controller) que é uma ideia para desenvolvimento de aplicações que consiste em separar as 3 "peças" principais da aplicação nos seus ambientes isolados. O *modelo*, a *visão* e o *controlador*.

No nosso caso usamos **servlets** para controlarem todo o fluxo da nossa aplicação para tratarem todos os pedidos provenientes da visão tratando esses pedidos interagindo com os modelos apropriados, beans na nossa aplicação. As servlets contêm as beans declaradas e quando necessitam de ir buscar qualquer informação à base de dados chamam a função setattribute passando-lhe como parâmetros uma string, que vai ser o nome/identificador da bean, e a variável declarada anteriormente.

Como modelo/s usámos **beans** que realizam todas as interacções com a base de dados. O modelo é onde os dados do controlador e por vezes da visão são recebidos, enviados e manipulados. Nesta aplicação as servlets fazem pedidos aos beans para estes retirarem todo o tipo de informação da base de dados, essa informação é então enviada para a servlet que irá carregar na visão a página, correspondente à acção realizada pelo utilizador que despoletou o pedido de acesso à base de dados, com os dados recebidos.

A visão é exactamente o que o nome indica: a interface visível ao utilizador com a qual este interage, mostrando botões, dados informativos, formulários etc. Para a nossa aplicação fazemos uso de **Java Server Pages** (JSP's) que possibilitam que o html, que apresenta o layout da página, esteja num lugar separado do código em java utilizado nas servlets ao mesmo tempo que permite o uso de algum java através de por exemplo JSTL(JavaServer Pages Standard Tag Library). Algo muito importante que se deve ter em conta é o facto de quando nalguma página é necessário ter acesso a algum bean, ao inicializar esse bean convém que o id seja igual ao nome dado na servlet para que a servlet consiga chamar o bean correto.

Concluindo o controlador carrega a visão depois de interagir com o modelo, que é o que reúne a informação necessária para mostrar na visão correspondente. Portanto as servlets tratam dos pedidos provenientes das jsp's (exemplo: utilizador carrega num botão), pedem às beans para ir buscar as informações necessárias à base de dados e carregam-nas na jsp correspondente.

Integração com servidor do projecto 1

Uma vez que o servidor RMI é um servidor que implementa objectos remotos por isso é sempre o mesmo. Para conseguirmos ligar este ao servidor web basta ter a mesma implementação da interface remota da base de dados e classes comuns nos dois projectos para encapsulamento de dados necessários para o correcto funcionamento da aplicação.

Na implementação da base de dados, e na sua interface, tiveram de ser criados novos métodos para que as servlets, através dos beans, consigam aceder aos dados necessários para fazer o carregamento da página correcta, apresentando-as ao utilizador, com as informações retribuídas da base de dados.

Integração de Websockets

WebSocket é um protocolo que permite comunicações full-duplex entre dois pares sobre o protocolo TCP. Na nossa aplicação o websocket é inicializado na right.jsp assim que a página é carregada pelo browser chamando a função initialize, através de javasript. O websocket é conectado à websocket servlet, que está implementada na classe **ChatWebSocketServlet**, através da função connect que também implementa os event listeners, que por sua vez irão estar à escuta de acções no socket.

Assim que o socket é aberto, provocado pela inicialização da aplicação por um utilizador que faça login, é enviado para o ecrã o username do servidor que se logou, informação essa conseguida com auxilio da bean do user e de jstl's como já explicado anteriormente. Quando o socket é fechado é impressa uma mensagem de aviso ("WebSocket closed") no ecrã.

Todas as mensagens são escritas ou enviadas para o ecrã chamando a função **writeToHistory** que recebe como parâmetro o texto a ser enviado. Esta função vai inicialmente procura na página o elemento com o id correspondente ao do local onde irá ser mostrada a mensagem. Através da propriedade innerHTML o texto passado como parâmetro é guardado numa variável que irá ser adicionado usando o appendChild às informações anteriormente mostradas.

Foi criada uma interface remota **Notificacoes,** implementada pela **Notificacoes_implementation,** que é colocada dentro da servlet **ChatWebSocketServlet** para assim ter acesso aos métodos desta. Com isto faz-se um callback à base de dados e no final da transação a função **notificate_users**, que está implementada em Notificacoes_implementation e recebe como parâmetro a mensagem a enviar**,** é chamada e esta por sua vez chama a função broadcast que envia a mensagem para todos os utilizadores a informar que uma transacção entre dois utilizadores acabou de acontecer e o preço a que foram vendidas/compradas as acções.

REST

A classe **FacebookRestClient** é implementada no servidor web e no servidor RMI pois é através deste último que a aplicação faz os pedidos ao servidor. Quando o utilizador carrega no botão do facebook no menu é chamado o método **autenthicate** que irá pedir à API do facebook um token de acesso às informações do utilizador em questão.

Assim quando um utilizador insere uma ideia, e está ligado ao facebook, é chamado o método **post_on_facebook**, que recebe como parâmetro o texto que vai ser inserido, e é guardado na base de dados o id da ideia inserida e o id do post do facebook. Isto para que quando um utilizador comprar acções dessa ideia, e esteja ligado ao facebook, seja inserido um comentário no post dessa ideia, chamando o método **post_comment_on_facebook** que recebe como parâmetros o texto a escrever e o id do post onde será inserido o comentário, a indicar que esse utilizador comprou acções dessa ideia e o preço que pagou. O id do post é retirado, usando um String Tokenizer, da string que é recebida da API do facebook através do método **getBody**.

Quando o utilizador apaga uma ideia na aplicação, se este estiver ligado ao facebook e essa ideia tiver sido postada no seu facebook quando foi inserida, o post respectivo no facebook é apagado utilizando o método **delete_on_facebook** que recebe como parâmetro o id do post que irá ser apagado.

Descrição de testes

Teste	Passou/Falhou
É inserido post no facebook do utilizador, se este estiver conectado, quando ele insere ideia	✓
É apagado o post da ideia quando esta é apagada da aplicação	✓
Login com conta de facebook em vez de credencias normais	X
É postado um comentário no post da ideia quando utilizador compra acções dessa ideia	X
Websockets enviam informação para browser assim que transacção acontece	✓
Informação actualizada no que diz respeito ao preço por acção da última transacção	X
Utilizador consegue registar-se	✓
Utilizador consegue fazer login com os seus dados	✓
Utilizador consegue criar um tópico	✓
Utilizador consegue ver uma listagem de tópicos previamente criados	✓
Utilizador consegue inserir uma ideia	✓
Utilizador consegue apagar uma ideia	✓
Utilizador consegue visualizar uma listagem com um histórico de transações realizadas	✓
Utilizador consegue mudar o preço das ações de uma ideia sua	✓
Utilizador consegue comprar ações de uma ideia	✓
Servidores e clientes correm em máquinas diferentes	✓
Utilizador consegue por ideias na sua watchlist	✓
Utilizador consegue ver o seu portfolio de ideias inseridas	✓
Utilizador consegue ver hall of fame	✓
Root consegue adquirir ideias	✓

Tabela 1