Sistemas Distribuídos

Trabalho Prático: iBei

Meta II

João Feliciano - jamaia@student.dei.uc.pt - Nº 2009112631

Luís Coimbra - lcoimbra@student.dei.uc.pt - Nº 2014232313

Luís Silva - lfgsilva@student.dei.uc.pt - Nº 2008103138

18 de Dezembro de 2016

Arquitectura de Software

Servidor RMI

O servidor RMI cria as suas próprias *Threads* quando os métodos são invocados pelos servidores TCP e pelo webserver. Este é o responsável pela comunicação com a base de dados.

Nesta meta adicionamos métodos adicionais de modo a incorporármos os utilizadores com Facebook na base de dados.

Servidor WEB

Ao usármos o *Model-View-Controller*(MVC) 1 da Struts2 temos o nosso código dividido por *layers* permitindo assim a reutilização e manutenção de código e separação de conceitos. O nosso Model é composto pelos BEANS e RMI, a nossa View pelos JSP's e o nosso *Controller* pelo conjunto de ACTIONS criadas de modo a termos um fluxo de controlo na aplicação. A nossa *View* tem o propósito de apresentar a devida informação ao utilizador e o *Model* permite-nos ter persistência dos dados e acesso ao RMI, que por sua vez comunica com a base de dados.

Além de seguirmos o modelo MVC, utilizamos as *Tags* de Struts2 *struts-tags* e as *JSTL*.

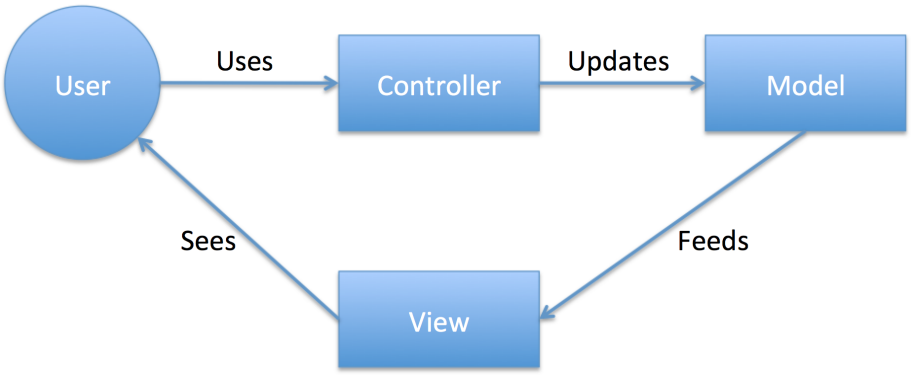

Figura 1: Modelo MVC

Seguimos a arquitetura proposta no enunciado do projeto 2, criando um servidor *web* que responde aos clientes *web* através de menssagens HTTP, além da interligação com as API's do Facebook e do Ebay.

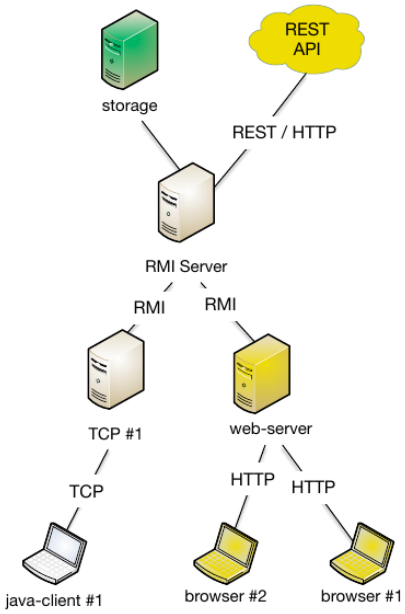

Figura 2: Arquitetura

Structs e integração com Servidor RMI

A integração do *structs* com o servidor RMI é realizada nos *Beans* e utilizada nas *Actions*. Na criação do bean *AuthenticationBean* efetuamos o *Lookup* do RMI de modo a termos a referência remota da interface implementada pelo RMI. A partir do momento que conseguimos obter esta referência simplesmente invocamos os métodos mapeados nessa interface. Esta informação é inserida na sessão de cada utilizador no processo de autenticação e depois é utilizada em várias *Actions*, dependendo da ação do utilizador.

A integração de *CallBacks* foi implementada criando a classe *Callback*. Esta classe implementa a interface do servidor *TCP* da primeira meta do projeto, dando *Override* ao método e contém um objeto *WebSocket*, passado no construtor, que vai ser utilizado para enviar mensagens em tempo real ao cliente quando o servidor RMI faz o *Callback*. As notificações enviadas ao cliente através de *WebSocket* são mostradas ao mesmo através de *alerts* na página do *Browser*.

Quando o utilizador carrega uma pagina JSP no browser irá realizar ações que estão mapeadas no ficheiro *structs.xml*. As ações são iniciadas com a interação do utilizador, como por exemplo, inserir dados nos campos disponíveis,

clicar em botões ou mesmo seguir um processo de navegação.

Através do ficheiro anterior, quando é iniciada uma ação, seguimos para uma determinada *Action*. Nesta fase efetuamos o controlo que pretendemos, como por exemplo, verificar as credências de um utilizador invocando, através do *AuthenticationBean*, os métodos remotos do RMI.

Além do *Bean* anterior temos outros *Beans*, como o *AuctionBean* e o *AdminBean* que estão relacionados com os leilões e controlo do admininistrador. Como o próprio nome sugere, no *Controller* efetuamos o controlo da informação inserida, por exemplo um utilizador não poderá deixar campos em branco no processo de registo ou autenticação. A imagem seguinte 3 mostra a interação descrita.

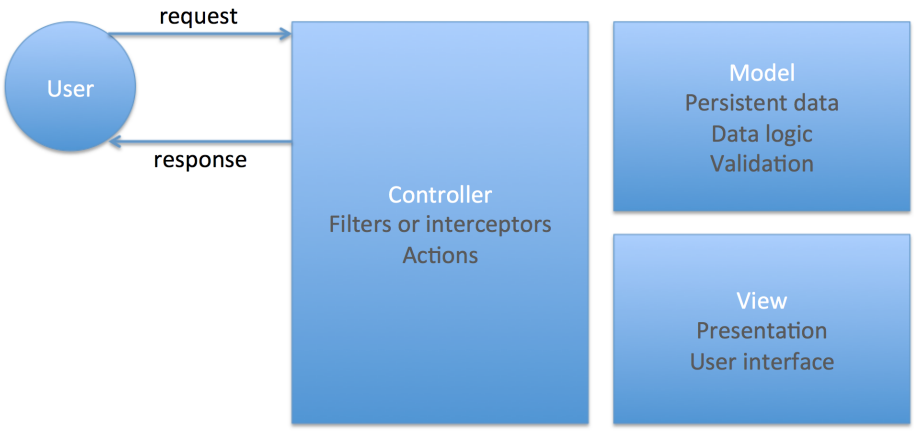

Figura 3: Interação do Utilizador

Quando uma *Action* interage com o(s) bean(s) retorna o resultado em forma de string. Este resultado está mapeado no ficheiro *structs.xml* que efetua um redirecionamento para o JSP pretendido (*View*). Guardamos, no caso de sucesso, alguns dados na sessão de utilizador de modo a efetuar verificações e a mostrar dados nos JSPs, como por exemplo o *username* do utilizador autenticado na página *Home*. Utilizamos deste modo a sessão de um utilizador para passar e apresentar dados temporários.

Implementámos um único *interceptor* para a autenticação de um utilizador criando a classe *MyInterceptor*. Com este, a verificação de autenticação de um utilizador fica mais fácil, pois ele intercepta todas as actions de modo a verificar se um utilizador está autenticado na aplicação ou não. A figura 4 seguinte demonstra o papel do *interceptor*, além de demonstrar a estrutura interna da Struts2 com mais detalhe, demonstra também a interação entre os vários componentes ou intervenientes.

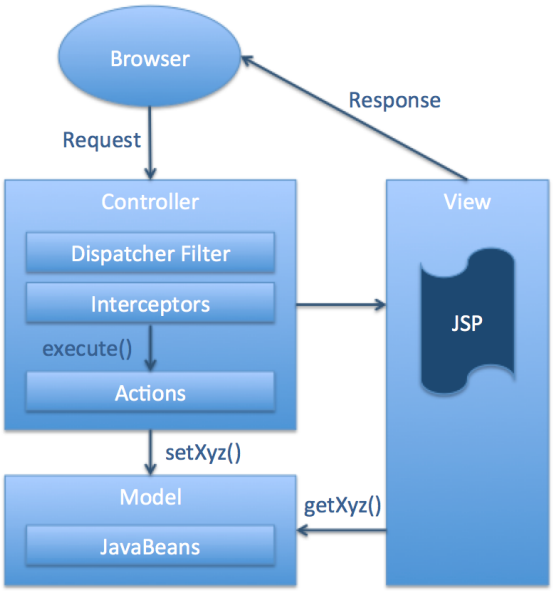

Figura 4: Estrutura interna do Struts2 com interação do Utilizador

Implementação de WebSockets e sua integração com o servidor RMI

Para termos o nosso servidor RMI responsivo, isto é, para que este possa notificar o cliente em tempo real, utilizámos o *WebSockets* (*/ws*). O nome usado foi inserido no *container* do *Tomcat* usando *@ServerEndpoint*.

Para a implementação dos *WebSockets* começamos por implementar o demo 7, perceber o seu funcionamento e como o poderiamos usar como base de desenvolvimento para resolvermos o problema das notificações em tempo real.

Finalizado este passo, criámos um *script* de javaScript (*wsScript.js*) para criar e iniciar o *webSocket* do lado do cliente e mostrar a informação ao mesmo.

Criámos uma classe, *WebSocketAnnotation.java* para a implementação do *webSocket*. Nesta guardamos o nome e o ID do utilizador e utilizamos um *CopyOnWriteArraySet*, uma vez que é *Thread Safe* para podermos enviar actualizações para dos utilizadores online na aplicação e dos utilizadores a visualizar cada leilão.

Por fim uma classe *Callback.java* é iniciada pelo *WebSocket* no servidor e implementa a *tcpInterface* criada na primeira meta. Assim quando o servidor RMI necessita de notificar o cliente invoca o método do *Callback.java* e este trata de enviar através da referência que tem para um *webSocket* do lado do servidor uma mensagem para o *webSocket* do lado do cliente.

Já neste lado final vai ser criado um *pop-up* no *Browser* para cada mensagem que o cliente recebeu enquanto esteve ausente, para quando recebe mensagens

em tempo real e quando recebe notificações de licitações em leilões em que já licitou.

Integração de Serviços REST

Facebook

O uso da API do Facebook com autênticação envolveu o registo de uma aplicação na zona *developers* deles. O processo de autenticação e autorização foi alcançado com o protocolo OAuth. Tivemos de guardar a *apiKey* e a *apiSecret* específicas da nossa aplicação para depois usarmos na criação de um *URL* de autorização. Este *URL* permite ao utilizador dar autorização à nossa aplicação para aceder a informação da sua conta.

Para isto tivemos de mapear no ficheiro *struts.xml* o seguinte resultado. *<result name="redirect"type="redirect">% {authorizationUrl} </result>*. Escolhendo *callback("http://localhost:8080/fbLoginAction")* nos campos do *OAuthService service* para o *callback* para a *action fbLoginAction*.

Depois desta autorização, é criado um código de verificação, pelo Facebook, sendo este passado por método *GET* no *URL* de *callback* referido acima. Utilizamos esse código para pedir um código de acesso.

O último passo é pedir esse código de acesso, específico para o Utilizador e aplicação, que serve para assinar os pedidos HTTP enviados ao Facebook.

Depois de todo este processo estamos em condições de efetuar os pedidos pretendidos ao Facebook, usando para isso verbos, por exemplo, *request = new OAuthRequest(Verb.GET, PROTECTED_RESOURCE_URL, service)* .

Quando criamos um *Post* no Facebook, inserimos o parâmetro "*link*"no HTTP Request com o valor *"http://eden.dei.uc.pt/˜amaf/echo2.php?redirect= http://localhost:8080/DetailAuction?Id=<ID_do_leilao_criado>"* de modo a conseguirmos visualizar o leilão criado. Esta visualização é conseguida através do *redirect* embutido no link anterior.

É possível um utilizador efetuar o login com a conta do Facebook. Se esta não estiver associada a uma conta na base de dados, o processo de autênticação é anulado. Primeiro o utilizador tem de efetuar um registo na aplicação e depois de efetuar a autênticação é dado a opção a esse utilizador de efetuar a associação da conta local com a sua do Facebook num botão ao lado do botão de *Logout*, na barra de naveção.

Ebay

O uso da API do Ebay não envolveu nenhuma autênticação por parte do utilizador. Usamos em concreto a Finding API como aconselhado no enunciado do projeto. Foi necessário efetuar o registo de uma aplicação no Ebay de modo a obtermos uma *EBAY_APP_ID*. Com este token podemos efetuar pedidos ao

Ebay, basta preencher os campos do *url EBAY_FINDING_SERVICE_URI* de modo a estabelecer a conexão. Os campos principais são os seguintes:

- *GLOBAL-ID=EBAY-ES* - escolhemos uma loja europeia, mais concretamente o Ebay espanhol, por ter os preços em euros.
- sortOrder=PricePlusShippingLowest - para encontrar o preço mais baixo, como pedido no enunciado do projecto.
- *OPERATION-NAME=findItemsByKeywords* - modo de pesquisa.
- *RESPONSE-DATA-FORMAT=XML* - formato da resposta.
- *SECURITY-APPNAME=LuisSilv-sd2016te-PRD-a45f0c74d-746c7147* - o ID da aplicação.

São feitos pedidos HTTP procurando produtos com as *keywords* do título do leilão da nossa aplicação, vindo a resposta em formato XML. Em seguida é feito o *parsing* da resposta e é procurado os campos que nos interessam, principalmente o do preço.

Propriedade Intelectual Externa

Para a realização deste Trabalho usamos um imagem para servir de icon que pode ser encontrada gratuitamente em `https://www.iconfinder.com/icons/532644/acquisitions_auction_finance_gavel_hammer_justice_law_icon#size=48`.

Utilizamos também o Bootstrap para a parte de design das páginas *web*, pode ser acedido em `http://getbootstrap.com/getting-started/#download`.

Para o uso da API do Ebay baseamo-nos, além dos exemplos dados pelos docentes, nos seguintes exemplos:
`http://stackoverflow.com/questions/2960265/implement-ebay-finding-feedback-api`
`http://developer.ebay.com/Devzone/finding/HowTo/GettingStarted_JS_NV_JSON/GettingStarted_JS_NV_JSON.html`

As figuras 1 a 4 foram retiradas dos apontamentos teóricos da Unidade Curricular.

Como Executar

Nesta meta não foi possível integrar o nosso ficheiro *iBei.proprieties* com o *webServer* pelo que temos que colocar a máquina onde vai correr o webserver com IP 192.168.1.3, pois este está estatico no .war.

Em seguida é necessario configurar o ficheiro *iBei.proprieties* pois o TCP e RMI *Servers* irão correr com esta configuração.

Feito isto podemos iniciar o RMI server, seguindo-se do servidores TCP (caso pretenda algum) e por fim o *webServer* por meio do ficheiro .war.

Distribuição de tarefas

A distribuição de tarefas foi efectuada da seguinte forma:

- João Feliciano
  - Idealização do design e da navegação da página Web
  - Criação dos JSPs, Actions e Bean
  - Criação do WebSocket
  - Interligação dos callbacks entre o WebSocket e o RMI
  - Criação de novos metodos necessários no RMI
  - Testes à plataforma
  - Criação do relatório
- Luís Coimbra
  - Idealização do design e da navegação da página Web
  - Criação dos JSPs, Actions e Bean
  - Criação de novos metodos necessários no RMI
  - Alterações necessárias na Base de Dados
  - Criação do REST
  - Testes à plataforma
  - Criação do relatório
- Luís Silva
  - Idealização do design e da navegação da página Web
  - Criação dos JSPs, Actions e Bean
  - Criação do WebSocket
  - Criação do REST
  - Testes à plataforma
  - Criação do relatório

Testes à plataforma

Requisitos Funcionais	**Resultado**
Registar novo utilizador	OK
Login protegido com password	OK
Criar um leilão (incl. Integração com a meta 1)	OK
Pesquisar leilões por código EAN/ISBN	OK
Ver todos os detalhes de um leilão (incl. histórico de licitações)	OK
Listar todos os leilões em que o utilizador está/esteve ativo	OK
Licitar um valor num leilão (incl. integração com a meta 1)	OK
Editar propriedades de um leilão, guardando versões anteriores	OK
Escrever mensagem no mural de um leilão (incl. integração com a meta 1)	OK
Leilão termina corretamente na data, hora e minuto marcados	OK
Grupos de 3: cancelar um leilão (-3)	OK
Grupos de 3: banir utilizador, cancelando leilões e licitações (-3)	OK
Grupos de 3: mostrar estatísticas de atividade (-3)	OK
WebSockets	**Resultado**
Notificação imediata de licitação melhor de outro utilizador	OK
Utilizadores online leem novas mensagens imediatamente	OK
Utilizadores offline leem novas mensagens assim que se ligam	OK
Listar utilizadores online	OK
Grupos de 3: Contador em tempo real de utilizadores em cada leilão (-5)	OK
REST	**Resultado**
Associar conta ao Facebook	OK
Login com o Facebook	OK
O leilão deve ser partilhado no Facebook assim que é criado, pelo autor	OK
A página do leilão deve mostrar o preço mais barato no ebay	OK
Grupos de 3: deve ser possível desassociar um utilizador do facebook (-5)	OK
Extra	**Resultado**
Login em Smartphone ou Tablet	OK

Anexos

ER da Base de Dados