Universidade de Coimrba

• U  C •

Universidade de Coimbra

Sistemas Distribuídos

Relatório de Projeto
iBei: Leilões Invertidos
Meta 2

Alexandre Ferreira Costa
2014206463 - afc@student.dei.uc.pt

Ana Inês Mesquita Fidalgo
2013134819 - aimf@student.dei.uc.pt

Pedro Filipe Matos Godinho Gabriel Coelho
2009116949 - pfcoelho@student.dei.uc.pt

18 de Dezembro de 2016

Conteúdo

Capítulo 1

Introdução

Este projecto foi realizado no âmbito da cadeira Sistemas Distribuídos (SD) inserida no plano de estudos da Licenciatura em Engenharia Informática da Universidade de Coimbra, lecionada pelos Professores Doutores Raul André Brajczewski Barbosa e Alcides Miguel Cachulo Aguiar Fonseca, no ano lectivo de 2016/2017.

O projecto acima referido tem como objetivo mostrar a aplicabilidade de um Sistema *Remote Method Invocation*, ou RMI, de modo a criar uma camada de persistência de dados, e aceder a estes dados através de uma camada Web (Servlets) onde é utilizado o Structs para ir buscar a informação introduzida pelos utilizadores na realização de pedidos.É-nos também pedido para fazer comunicação entre clientes, para a qual usamos WeSocket.

1.1 Contexto

No propósito do trabalho é-nos pedido para implementarmos um sistema de Leilão Invertido, onde um vendedor declara qual o preço máximo que está disposto a pagar por um artigo e os outros utilizadores licitam o preço mais baixo que aceitariam receber por esse mesmo artigo.

A par deste método de Leilão é-nos pedido para implementarmos várias funcionalidades que permitam a construção desta plataforma como a criação do servidor RMI (em parte já implementado na 1ª meta do trabalho), a utilização de WebSockets para receber notificações offline e online e a utilização do rest para associar as contas dos utilizadores a uma conta de Facebook. Implementamos ainda as funcionalidades de administradores - dado que somos

um grupo constituído por 3 pessoas - e todas as eventuais funções necessárias para o decorrer do trabalho.

Capítulo 2

Arquitectura de Software

O nosso projecto está dividido em várias packages: RMI, Websockets, FacebookApi e uma package para cada funcionalidade diferente implementada. O objectivo principal do nosso projecto consiste em: é realizado um pedido pelo utilizador numa Servlet, e através da sua action e da sua bean, a qual se vai fazer lookup ao rmi e chamar as classes genéricas que estão guardadas no RMIServer.

2.1 RMIServer

Sempre que o RMIServer é iniciado vai verificar a informação contida nos ficheiros para preencher os arraylists de leilões, users e admins. Para além disto, a partir de uma thread, é verificado contantemente o término dos leilões para mantendo os ficheiros actualizados.

2.1.1 Login

Quando se recebe um pedido de login do por um utilizador, é criada uma resposta a partir da classe LoginBean que irá realizar o lookup ao RMI e chamar a função login, presente no servidor RMI, para obter uma String[] onde estarão colocados os parâmetros de resposta necessários para apresentar a página correspondente ao utilizador. Na função login da classe RMIserver procuramos um user no ArrayList users do RMI que contenha o mesmo username que o desejado e por fim da-nos uma resposta de true ou false caso encontre um user válido ou não. Este User que fez login é adicionado a um

ArrayList de Users ligados até acabar a ligação.

2.1.2 Registar User normal e Admin

Sempre que é recebido um pedido do tipo registar, tal como na classe de Login, é criada uma instância da classe Registar, onde é feito lookup ao RMI e é gerada uma lista de strings de resposta para os comandos a partir da função Registar ou RegistarAdmin, ambos presentes no RMIServer. Quando são registados, estes são adicionados ao ArrayList de Users e/ou Admin respectivamente, e no final os ficheiros são actualizados.

2.1.3 Criar Leilao

Na função get do CriarLeilaoBean é realizado um split da string do pedido para para obtermos os dados necessários à criação da resposta. O processo é o mesmo das classes de login e registo, temos uma classe CriarLeilaoBean onde é realizado o look up ao RMI para obtermos uma resposta ao pedido. Depois disto é enviada uma resposta. Quando é criado um leilão este é adicionado ao ArrayList global de leilões do RMI e são actualizados os ficheiros.

2.1.4 Procurar Leilão

A procura de um leilão é feita através do seu código. Este código não é único, sendo apenas o código do artigo a leiloar, podendo existir vários leilões com o mesmo código. Na classe SearchLeilaoBean é realizado o lookup, e chamada a função SearchLeilao do RMI, onde vamos procurar no ArrayList leiloes um leilão com o id que foi inserido. É criada ainda uma lista de strings onde estão os outputs a aparecer no comando de resposta e, de seguida, é enviada a resposta utilizador, que é redirecionado para a página com a informação correspondente.

2.1.5 Detalhe de Leilão

Esta procura é semelhante à anterior em termos de implementação mas é uma procura por id, o atributo único do leilão. Além disto, devolve ao utilizador uma resposta mais detalhada deste.

2.1.6 Listar as Licitações de um User

O RMI contem uma função chamada ListarLicitacoesUser() que irá percorrer o ArrayList de Leilões e o ArrayList de licitações de cada leilão. Esta procura é feita com o username do cliente em questão, sendo este o seu atributo único. O processo é sempre semelhante, criação de uma instância de uma classe que extende pedido, recebe o pedido do cliente e liga-se ao RMI para chamar a função que irá directamente buscar a informação aos ficheiros, e enviar a resposta na thread menu ao cliente.

2.1.7 Editar Leilão

O cliente faz o pedido do que vai ser editado e qual o leilão a ser editado. No menu (TCPServer) é feita a gestão do pedido para obter os parâmetros necessários, é criada uma resposta a partir da classe EditarLeilao, feito o lookup ao RMI e vai buscar a resposta à função correspondente no RMIServer. Nesta função primeiramente é guardado uma cópida do leilão a editar, para posterior visualização. No final são actualizados os ArrayLists e de seguida os ficheiros.

2.1.8 Listar Versões Anteriores de Leilão

Um leilão contem um ArrayList de versões anteriores de si próprio. Sempre que este é alterado é guardada uma cópia neste ArrayList. O cliente a partir do comando e do id de um determinado leilão consegue receber uma lista de todas as versões antigas de determinado leilão.

2.1.9 Escrever Mensagem em Leilao

Um leilão contem um ArrayList de mensagens de Mural. Para escrever uma mensagem num leilão, tem de se usar o comando message e dar um id. Este pedido é tratado com splits quando é recebido no servidor TCP e é criada uma resposta a partir da classe EscreverMural. Esta classe faz o lookup ao RMI e obtem uma resposta obtida a partir da função EscreverMensagem do RMI. Quando é enviada a mensagem, esta é armazenada num ArrayList do tipo Mensagem e caso o dono ou os licitadores desse leilão não estejam online na altura do envio esta é armazenada num ArrayList de mensagens

pendentes. Quando estes utilizadores fazem login, recebem uma notificação de mensagem e esta é eleminada do ArrayList de mensagens pendentes.

2.1.10 Listar os Meus Leilões

Quando é realizado este pedido, é realizado mais uma vez o lookup, e é criada uma lista de Strings que contem os parâmetros necessários a formar o comando de resposta.

2.1.11 Licitar num Leilão

Quando é feito um pedido para licitar um leilão, este só é feito se o valor introduzido for inferior ao valor base introduzido pelo dono do leilão ou ao da licitação melhor. Caso o valor introduzido seja superior, é retornado um comando com o parâmetro "ok"a false.

Caso seja realizada a licitação, é enviada uma notificação tanto ao dono como aos outros licitadores, a informar que determinado utilizador fez uma melhor licitação. Esta notificação fica no ArrayList de mensagens pendentes de cada utilizador e, quando estes utilizadores realizam o login, recebem a notificação de licitação, removendoa mensagem do ArrayList. São actualizados os ficheiros em tempo real.

2.1.12 Cancelar Leilão

Quando um leilão é cancelado, este é removido do ArraList de leilões, sendo que este leilão nunca obtem um vencedor.

2.1.13 Banir Utilizador

Função específica do Administrador; O cliente admin realiza um pedido para banir um utilizador e é criada uma resposta a partir da classe BanirUser. Este utilizador é identificado no arraylist de utilizadores e é removido, assim como todos os seus leilões e todas as suas licitações. É enviada uma resposta ao utilizador admin de sucesso caso tenha encontrado o utilizador que este desejava banir ou de erro, caso este não tenha sido encontrado. No final são actualizados os ficheiros em tempo real.

2.1.14 Listar Estatisticas

Existem três pedidos diferentes de estatísticas diferentes, top 10 utilizadores que já ganharam leilões, top 10 criadores de leilões e os leilões criados nos últimos 10 dias. No top 10 utilizadores que já ganharam, é realizada uma pesquiza nos ArrayLists de Leilões acabados e ganhos em todos os utilizadores presentes no ArrayList de utilizadores. Caso os ficheiros contenham utilizadores com zero leilões ganhos e/ou contenham menos de dez utilizadores, estes são também considerados no top 10.
No top 10 de criadores de leilões, é feita uma pesquisa no ArrayList de leilões geral onde se verificam quais os que criaram mais leilões.
A função de dizer o número de leilões criados nos últimos dez dias, vai verificar a diferença de dias entre a data de criação (guardada aquando da criação do leilão) e a data actual, incrementando um contador, é no final então retornada um comando de resposta ao utilizador com o tipo de pedido e o número de leilões criados.

2.1.15 Servlets

Para cada funcionalidade existe uma servlet com a sua action e a sua bean correspondente. Para além disto, cada acção contém um ou mais jsp's específicos, para os quais o utilizador é redirecionado para a realização de pedidos e páginas com os resultados aos pedidos.

WebSocktes

Para que as funcionalidades dos WebSocktes funcionarem dentro do previsto foi necessário ter as seguintes classes e ficheiros:

- *Notificações.jsp*;
- *scoket.jsp*;
- *WebSocketAnnotation.java*;
- *webSocketConfig*;

2.1.16 API

A componente da API serve para criar uma relação entre a nossa plataforma e plataformas já existentes. No caso específico a este projecto as

aplicações estrangeiras à nossa usadas foram o facebook e o eBay. Isto serve para termos uma base de dados de cliente extra, complementando a nossa - Facebook - e uma base de dados de leilões - eBay. Para tal efeito é usado uma sequência de scripts pré-definidos, que servem para iniciar uma sequência de troca de tokens para a obtenção das informações necessárias, sendo estas depois utilizadas conforme nos é mais útil.

Capítulo 3

Structs

Cada Servlet apresenta a sua própria package. Em cada package de cada Servlet estão presentes uma action e uma bean. No structs.xml é realizada o redirecionamento de páginas através do que a action retorna, se sucess ou error. A partir dos inputs realizados num form de um determinado jsp, o structs chama a função execute() da action correspondente, ou seja, os requests de um cliente são enviados sob a forma de uma action, se o controller recebe um request chama a action que interage com o model correspondente. O model retorna a string a dizer ao controller qual a página de output a enviar ao cliente. A informação é passada entre o model e a view sob a forma de JavaBeans e as tags definidas nos inputs no permitem à view ler e escrever o conteúdo desses benas, da camada de apresentação sem a necessidade de implementar código Java para o fazer.

Posto isto, tal como foi referido anteriormente, para cada função que lidava directamente com os nossos ficheiros, onde era armazenada informação, criamos uma Servlet.

O nosso projecto, relativamente à parte funcional, apresenta as seguintes Servlets:

- Actividade Antiga

  Esta Servlet contém a action ActividadeAntiga, na qual está contido o execute que será chamado através do structs.xml, contém ainda o ActividadeAntigaBean que realiza o lookup ao rmi no porto correspondente. Através das variáveis de da session, acedemos ao nome do utilizador que pretende realizar esta acção de listar a actividadeem Leilões que já acabaram, as suas licitações e os leilões ganhos, com este

nome chamamos a função getVelhos() presente no RMI e obtemos os dados pretendidos, que são apresentados ao utilizador num novo jsp, que acede a esta informação na bean.

- BanirUtilizador

  Para banir um utilizador o utilizador que realiza esta acção necessita ser um admin, sendo que esta acção apenas está presente num menu desse tipo. Quando o utilizador escolhe esta opcção, é redirecionado para uma página (definida no jsp do menu), nesta página é necessário que introduza o nome do utilizador a remover, sendo redirecionado para a action correspondente com um post. O structs irá então, a partir da tag no jsp, buscar o nome do utilizador na bean. Na bean irá realizar lookup ao rmi, e no get (que é chamado no execute da action) irá chamar a função de banir um utilizador com o user inserido. Nesta função do rmi, quando é apagado um user, os seus leilões e as suas licitações são também apagadas. Caso o acto de banir seja bem sucedido é enviada a string correspondente, e a partir deste resultado o structs.xml irá realizar redirecionar o user para a página aí estabelicida. Caso o input tenha sido um utilizador inválido, é devolvida uma string de erro na action e redireccionado o user para a página estabelecida no structs.xml.

- CancelarLeilao

  Para cancelar um leilão o utilizador admin tem de inserir o id deste leilão na página jsp redireccionada pelo menu. É feito um post para a action correspondente e é chamado o método execute desta action através do structs. A action vai chamar o get da bean para verificar que o leilão foi cancelado com sucesso, para determinar a string de resposta e assim redireccionar o user correctamente. Na bean é realizado o lookup ao rmi e através do getId realizado pelo structs para ir buscar o valor introduzido pelo utilizador, é chamada a função do rmi para cancelar um leilão.

- CreateLeilao

  Para criar um leilão o utilizador necessita introduzir no jsp correspondente o seu título, descrição, código do artigo, data de término e a licitação máxima. Através da session, no structs quando quando vai

buscar estes parâmetros, é definido como o criador o nome do utilizador dessa mesma sessão. Na bean é realizado o lookup e chamada a função do rmi, sendo retornado um boolean, o qual servirá para a action retornar a string de sucesso ou de erro, que irão determinar para qual página é que o utilizador é redirecionado. Caso o leilão tenha sido criado irá aparecer na lista dos seus leilões.

- EditarLeilao

  O utilizador pode editar todos os parâmetros excepto o id do leilão. No jsp correspondente o utilizador adiciona os novos parâmetros do leilõe e carrega em submit. Para este leilão ser alterado, é chamada a action correspondente através de um post e do structs. Nesta acção é chamado o método execute, neste método, é chamado o get da bean, onde é devolvido true em caso de sucesso, ou false em caso de insucesso. Na bean é relizado o lookup ao rmi no porto correspondente e chamada a função editarleilao do rmi com os input que o structs foi buscar ao jsp. Caso este leilão exista e seja alterado é retornado true, e caso não exista e não seja alterado é retornado false. A partir deste output, a action retorna uma string de sucesso ou de erro e a página para a qual o utilizador é redirecionado neste dois casos, é definido no structs.xml.

- LeiloesDezDias

  Esta servlet retorna ao utilizador admin a informação de quantos leilões é que foram criados nos últimos dez dias. O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- Licitacoes

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- Licitacoes

  Para um utilizador listar as suas licitações, vamos buscar o seu nome a partir da session para no bean, depois de ser realizado o lookup ao RMI, chamar-mos a função MyLici presente no RMI. É enviado um bolean para a action, de maneira a determinar qual é a página para qual o utilizador deve ser redirecionado. Na bean, anteriormente foi definido um comando de resposta com a informação a listar. Caso o

pedido seja bem sucedido, o utilizador é redirecionado para uma página onde acedemos a essa variável comandoderesposta e esta é apresentada ao utilizador. Caso o utilizador não apresente nenhuma licitação, é redireccionado O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- ListarMyLici

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- ListarUsersOnline

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- ListarVersoesAntigas

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- Login

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- LoginAdmin

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- Logout

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- mensagemLeiloes

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- Register

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- RegisterAdmin

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- SearchLeilao

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- SearchLeilaoDetalhe

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- SearchLeilaoUser

  Nesta servlet, para além de ser necessário procurar os meus leilões, era também necessário fazer esta lista com um link para cada um desses leilões, de forma a poder escolher um e ser redirecionado para a sua página de detalhes. O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores, mas no jsp de listar os meus leilões temos um tipo de input escondido com o id desse leilão, sendo que este jsp irá redirecionar para a action de listar os detalhes de um leilão com o id desse mesmo leilão escolhido, sem que o utilizador seja obrigado a inserir esse input ele mesmo.

- Top10LeilaosCriados

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

- Top10LeilaosGanhos

  O processo para retirar esta informação do rmi e o uso do structs é idêntico ao processo utilizado nas funções anteriores.

Capítulo 4

Funcionamento do Servidor RMI

No propósito do trabalho é-nos pedido a implementação de um servidor *Remote Method Invocation* (RMI), de modo a criar uma camada de persistência de dados. Um RMI é um método de programação orientada a objectos onde os dados objectos conseguem interagir entre diferentes computadores numa determinada rede distribuida.

4.1 Detalhe das Classes

4.1.1 RMI/RMIServer

A implementação do RMI passa pela criação de uma classe *RMIServer.java* - com o respectivo *header* de interface *RMI.java* - onde são implementadas todas as funções de ligação directa dos pedidos efectuados pelo utilizador, com a informação - Users, Leilões, Licitações - necessária à resposta dos seu pedido, informação essa guardada temporariamente em ArrayLists. Dentro desta classe encontram-se então as funções de Login/Logout e listagem de clientes e administradores, acesso e escrita de ficheiros, criação, edição, procura, cancelamento e término de leilões, escrita de mensagens no mural do leilão, licitação sobre os leilões e ainda funções de administrador, como recolha de estatísticas.

Na função main desta classe encontra-se um *try/catch* para criar a ligação

a um *RMIServer* (tenta ligar-se a um e, caso não consiga, tenta a outro) - e a leitura dos ficheiros para carregar a informação necessária para o decorrer do projecto.

Capítulo 5

Web Sockets

Neste projeto é pedido para usarmos WebSockets para garantir comunicação entre clientes, recebendo notificações de leilões, de mensagens e de licitações. Para isso são usados os seguintes ficheiros:

- *Notificações.jsp*;
- *scoket.jsp*;
- *WebSocketAnnotation.java*;
- *webSocketConfig*;

5.1 Funcionamento

Quando a WebSocket é iniciada, ou seja, quando alguém faz login (normal ou admin), essa mesma WebSocket vai estar sempre à espera de receber mensagem para imprimir na pagina html. Para além disso a WebSocket envia uma mensagem da lista de utilizadores online autilizada, que é actualizada sempre que alguém faz lougout.
Há ainda um contador de todos os utilizadores online, também sempre actualizado.

5.2 Notificaçoes de licitações

Quando é executada uma licitação é devolvido pelo RMI o dono do leilão, uma lista de todos os users que já tenham licitado nesse leilão. Após isso veri-

ficamos se o dono do leilão está online e, se estiver, enviamos uma mensagem mensagem para o mesmo através de webSocket. Caso isso não aconteça é chamada a função WriteMensagem do RMIServer, que recebe como parâmetros o id(int) do leilão a notificacão(String) e o nome do cliente(String) a que pertendemos enviar a notificação, essa função vai colocar essas notificacão no array de notificações pendentes.

5.3 Notificações de mensagens

Tal como nas notificações de uma licitação, as notificações de mensagens vão funcionar de maneira idêntica. Quando é escrita uma mensagem num leilão vai-se verificar se o user desse leilão está online iniciando-se o mesmo processo acima descrito.

5.4 Notificações Login

Quando um user faz login é automaticamente chamada a função login que verifica se e o user existe. Se existir, vai retornar um array de notificações pendentes, caso as tenha, e vai colocar esse array na session desse utilizador.

Ao fazer login a pagina envia uma mensagem ao WebSocket com o comando login. Esse comando vai buscar a session do utilizador e envia as notificações pendentes. Essas notificações pendentes vão passar a estar lidas e vão ser eliminadas do array das pendentes. Se o utilizador fizer lougot e login de seguida essas notificações já não vão aparecer.

Capítulo 6

API

No propósito do trabalho é-nos pedido para criarmos uma ligação entre a nossa aplicação e duas plataformas distintas: o Facebook e o eBay.

Este tipo de relação é feita através da ligação das Application Programming Interfaces (API's) correspondentes, ao programa, de modo a conseguir criar um canal de troca de *Tokens* para responder aos pedidos necessários de desenvolvimento da aplicação.

Dada a estrutura do trabalho que nos foi pedido, o propósito do acesso a estas plataformas significa a criação de uma maior facilidade no processo de register/login, através do Facebook, e um melhor acesso a leilões já existentes, eBay, entre outras funcionalidades.

6.1 Facebook

O facebook é, hoje em dia, uma das plataforma mais usadas do Mundo, sendo assim uma das bases de dados mais completas a nível de informações pessoais, havendo a possibilidade de obter o nome, idade, género, foto de perfil e outras informações requeridas mediante a manipulação que nós como, programadores apliquemos à interface.

Apesar disto tudo, para o desenvolvimento da nossa aplciação, e dadas também as limitações de requerimentos aos quais temos acesso por parte do Facebook, é-nos únicamente necessário o ID de conta referente ao utilizador que faça o registo/login com o seu perfil pessoal. Com essa informação conseguimos associar esse ID ao perfil previamente criado, dando uma possi-

bilidade de unificar ambas as contas, facilitando futuros acessos à aplicação.

É também requerido um acesso à página principal do utilizador para que este, ao criar um leilão, fique com dado leilão publicado no seu mural, a título informativo da sua acção. Estas acções são todas desenvolvidas de modo automático sem que o utilizador tenha de dar permissões nem tenha que publicar nada manualmente, mas sendo este um trabalho meramente simulativo de uma plataforma de leilões não é necessário ter essas publicações como públicas.

Em última instância é-nos pedido para podermos desassociar a conta de facebook da conta do utilizador. Por isto entende-se que todas as permissões previamente dadas pelo utilizador (acesso a nome, ID, publicações automáticas), são negadas e deixa de haver relações entre o cliente da aplicação iBei e o cliente da plataforma Facebook.

6.1.1 Explicação de código

Tendo o facebook uma dimensão e impacto tão grande, nos dias de hoje, a maioria do acesso às suas informações, ou seja, a manipulação da sua API, está compposta de forma bastante facilitada, havendo um conjunto enorme de scripts já fornecidos pela própria plataforma (tendo sido disponibilizada pelos Professores da disciplina sob a forma da ficha prática nº8).

A obtenção das informações consiste em gerar um Standard Open Authorization Request Service (OAauthService) através de um builder (no nosso caso é o Facebook, mas também aceitaria google, por exemplo); de uma apiKey e apiSecret, ambos fornecidos aquando do registo da aplicação na subplataforma de facebook, facebook developers; de um callback de recepção dos tokens.

Este service comunica com a plataforma, enviando um empty token, para que seja retornado um Url de autorização que redirecionará para um outro Url que conterá o código principal que servirá como "moeda de troca"pelo token de acesso às informações. Este token é assim passado pela função que irá requerer o token a ser usado, seja ele um token de recepção ou publicação de informação. Esta mudança depende do verbo usado (GET ou POST respectivamente) ao qual corresponde a acção.

Por último ou a informação a publicar é publicada no perfil do user, ou a plataforma devolve a informação pedida sob a forma de um objecto précriado (Response) nos quais uma das suas variáveis é um código de código

de permissão, que tem o valor de 200 caso haja validade de transmissão de informação, ou 401 caso não haja e, no primeiro caso, um parametro body que contem toda a informação requirida.

6.1.2 Associação de contas

Começa assim a parte que é da nossa competência desenvolver pois isto tudo descrito até agora consiste em material que já nos é fornecido sem precisarmos de implementar.

Uma das primeiras acções na qual utilizamos esta técnica de obtenção de informação é aquando da associação de contas (uma de iBei e uma de facebook). Para tal, após o login com a conta iBei, é dada a opção ao utilizador de fazer o login com a conta facebook. Esta opção direciona para um script para dar início à acção correspondente, sendo esta declarada na struct. Esta acção começa por iniciar o script java de login por facebook que é composto por 3 partes. A primeira parte, iniciada na função execute, separada das outras partes por uma cadeia *if/else*, começa por admitir que ainda não existe o código que inicia a troca de tokens. Como tal irá iniciar o service para receber o *authorizationUrl* da plataforma. Após isto envia uma resposta à acção que, em função a tal resposta irá iniciar um novo script do tipo *.jsp*. Este novo script irá obter o novo Url para o qual o original é redirecionado que lhe permitirá obter a variável *code* que, tal como dito antes, é a variável que permite despelotar a sequência de troca de tokens.

Após este processo a acção que está activa previamente responde a um novo resultado, voltando a direciona-la para o mesmo script java. No entanto, desta vez a variável code já tem um valor efectivo, não sendo *null*, fazendo assim com que seja iniciada a segunda parte da função *execute*. Esta segunda parte é a parte em que existe a troca de tokens para a obtenção da informação do user que fez o login. No fim deste processo ficamos então com o ID inerente à conta do user e procedemos à terceira parte que será correr todos os nossos users e atribuir à sua variável *FID* o valor do FacebookID.

No fim deste processo todo, o utilizador é enviado para uma página de confirmação da acção e redirecionado para o seu menu correspondente (menu de user normal, ou administrador).

6.1.3 Login directo

Após a acção ditada anteriormente fica associado ao cliente o seu FacebookID, dando assim a oportunidade de fazer-se um login directo com a conta de facebook, em vez de ser com a conta da aplicação. O processo é o mesmo que o anterior, excepto no último passo.

Na útlima componente do processo, em vez de se procurar, no conjunto dos utilizadores, pela correspondência para atribuir o valor de ID, é feito o processo inverso: é procurado o ID, comparando-o do que foi obtido pelo token de facebook para que, havendo correspondência, se faça o login directo usando as credenciais previamente definidas. É ainda verificado se o utilizador é uma utilizador normal ou um administrador de modo a envia-lo para o seu menu correspondente, nuna encorrendo no risco de tentar fazer login normal com um admin, ou vice-versa. Garante-se ainda que, não existindo esse dado facebook associado a um user, a aplicação redireciona-se para a sua própria página principal, não dando acesso à aplicação sem conta.

6.1.4 Publicação no mural

Mais uma vez este processo altera-se só na terceira fase do contacto com a aplicação, alterando-se parte do código previamente fornecido.

Nesta situação muda-se o verbo no Token Request de GET para POST. Isso faz com que o que seja enviado para o developers facebook seja o que se pretende publicar no mural. Na altura de primeiro contacto com este tipo de acção é feito logo o pedido de permissão para publicação da informação, não pedindo numa futura situação, dado que esse tipo de informação é guardada pelo facebook em sí, não sendo necessário guardar isso da nossa parte.

Capítulo 7

Distribuição de Tarefas

Requisitos Funcionais	
Registar novo utilizador	Inês Fidalgo
Login protegido com password	Inês Fidalgo
Criar um leilão (incl. Integração com a meta 1)	Inês Fidalgo
Pesquisar leilões por código EAN/ISBN	Inês Fidalgo
Ver todos os detalhes de um leilão (incl. histórico de licitações)	Inês Fidalgo
Listar todos os leilões em que o utilizador está/esteve ativo, actuais ou antigos	Inês Fidalgo
Licitar um valor num leilão (incl. integração com a meta 1)	Alexandre Costa e Inês Fidalgo
Editar propriedades de um leilão, guardando versões anteriores	Alexandre Costa e Inês Fidalgo
Escrever mensagem no mural de um leilão (incl. integração com a meta 1)	Alexandre Costa e Inês Fidalgo
Leilão termina corretamente na data, hora e minuto marcados	Inês Fidalgo
cancelar um leilão	Inês Fidalgo
banir utilizador, cancelando leilões e licitações	Inês Fidalgo
mostrar estatísticas de atividade	Inês Fidalgo

WebSocket	
Notificação imediata de licitação melhor de outro utilizador	Alexandre Costa
Utilizadores online leem novas mensagens imediatamente	Alexandre Costa
Utilizadores offline leem novas mensagens assim que se ligam	Alexandre Costa e Inês Fidalgo
Listar utilizadores online	Alexandre Costa
Rest	
Associar conta ao Facebook	Pedro Coelho
Login com o Facebook	Pedro Coelho

Capítulo 8

Testes

Figura 8.1: Login com sucesso.

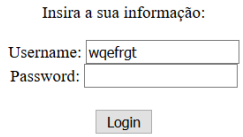

Figura 8.2: Login falhado

Figura 8.3: Lista dos leilões dos últimos 10 dias

Figura 8.4: Listagem de notificações imediatas

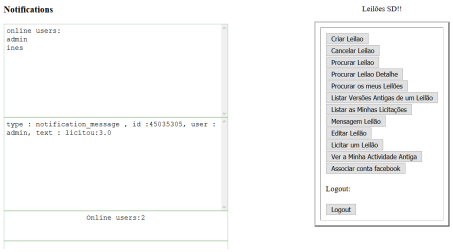

Figura 8.5: Recepção de notificações pendentes

Figura 8.6: Procura de Leilão por ID

Figura 8.7: Menu de procurar leilão por código

Figura 8.8: Resultado de procurar leilão por ID

Figura 8.9: Apresentações dos leilões referentes ao user

Figura 8.10: Procurar leilão por código

Figura 8.11: Apresentação de leilões não existentes

Figura 8.12: Ecrã de recepção de mensagens num leilão

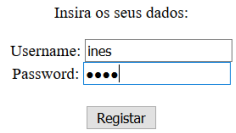

Figura 8.13: Ecrã de registo

Figura 8.14: Ecrã de apresentação de top10 criadores de leilões.

Figura 8.15: Ecrã de apresentação de top10 vencedores de leilões.

Figura 8.16: Ecrã de apresentação de detalhes do leilão.

Figura 8.17: Ecrã de apresentação de versões antigas de um leilão.

Figura 8.18: Ecrã de pedido para ver versões antigas.

Figura 8.19: Ecrã de apresentação de versões antigas não existentes.