

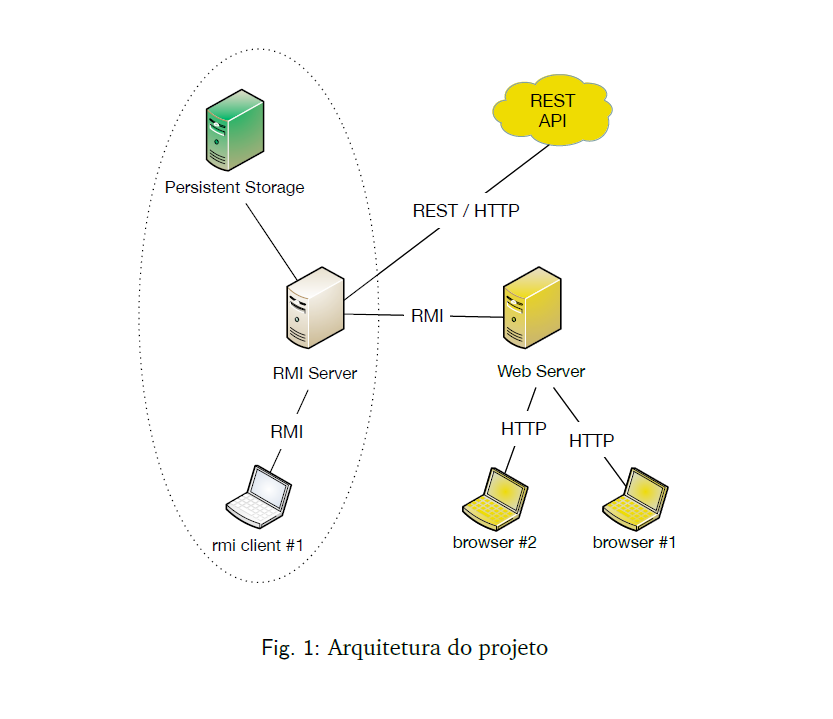
**ucBusca: Motor de pesquisa de páginas Web**

**Projeto de Sistemas Distribuídos – Meta 2**

Carlos Miguel da Luz Lima – 2017266922

Gabriel Nunes Saraiva - 2017256436

* **Arquitetura do projeto**



A arquitetura do nosso projeto consiste em vários clientes que, através do browser, comunicam com um servidor RMI para fazer as ações pretendidas. Este servidor posteriormente, comunica utilizando o protocolo UDP com um servidor multicast que processa o pedido e envia de volta uma mensagem ao RMI Server que, por sua vez, utilizando os models e as views, mostra os resultados ao cliente.

O nosso projeto é composto por vários tipos de ficheiros. O models é constituído pelo ficheiro pesquisaBean.java onde são feitas todas as chamadas ao RMIServer. As views são todos os ficheiros .jsp que é por onde o cliente interage com a aplicação. Através destes ficheiros ele faz ações que tanto enviam dados como recebem, como por exemplo, a ação de Login vai enviar dados para o servidor, ou quando recebe um resultado de uma pesquisa, vai ser mostrado no pesquisa.jsp, pesquisa\_admin.jsp ou pesquisa\_user.jsp, dependendo do tipo de utilizador a realizar a pesquisa.

O controller é constituído pelo ficheiro PesquisaAction.java onde estão todas as actions que são chamadas pelos submits que estão nas views. Estas actions depois vão chamar funções do Bean, devolvendo no fim uma string que depois vai determinar se houve sucesso ou houve um erro.

* **Integração do Struts2 com o servidor RMI**

Vão existir 3 tipos de ficheiros: Os controllers que são os Action, os models que vão ser os Bean e as Views que são os ficheiros .jsp.

O utilizador quando abre o browser e acede à nossa aplicação, é o ficheiro .jsp, ou seja, a View que lhe é mostrado. Dentro desta, ele ao interagir com os botões e a inserir dados, estes, através do submit irão ser enviados para o Action. Neste ficheiro, depois vão ser chamadas funções do Bean com os dados que foram recebidos do utilizador. No Bean, com os dados recebidos pelo Action, irá chamar a função que se encontra no RMI Server. Este, comunicando com o MulticastServer, irá gerar uma resposta que é devolvida para o Bean. Neste Bean, dependendo se for uma verificação de login/registo ou apenas uma apresentação de resultados como numa pesquisa, irá enviar dados para o action que, por sua vez, retorna uma string consoante o que recebeu e, no structs.xml, é decidido que View se vai renderizar de acordo com a tal string recebida do action. Caso seja então um resultado de uma pesquisa, onde apenas é necessário apresentar resultados, a View onde vamos querer apresentar essa informação, irá chamar diretamente o resultado da função Bean e mostrar no ecrã.

* **Integração do serviço REST**

Para ser possível entrar ou registar com a conta do Facebook foi necessário implementar outros ficheiros Action que, através do serviço REST irão adquirir os dados necessários do utilizador. Sendo auxiliados pela ficha 8, implementámos 4 files Action. FacebookAction; CallbackAction; LoginCallbackAction; e LoginFacebookAction.

Quando o utilizador clica no botão “Registar pelo Facebook”, a action CallbackAction é chamada onde se vai buscar o AuthorizationUrl ao service criado, após isto damos setUrl desse Url no nosso model. Para além disso este Action vai dar callback para o outro action (FacebookAction) onde aqui já se vai buscar a informação do utilizador ao perfil dele do Facebook, utilizando o método Parse da biblioteca JSON. Após conseguir obter a informação do utilizador, chama-se a função registo e funciona da mesma maneira, registando um user no nosso file .txt, com Nome = Nome de Perfil, Username = Nome de perfil e Password = “password”.

Para efetuar o login na aplicação, efetua-se exatamente os mesmos passos, contudo desta vez, é chamada a função de Login que vai buscar o Nome de perfil do Facebook do utilizador, e através do model, chamando o RMI verifica todas as condições necessárias e entra na aplicação com o perfil pretendido.

Para além da utilização do REST para a utilização do Facebook, foi utilizado a API do Yandex para traduzir os resultados de uma pesquisa e, para além disso, mostrar a língua original dos resultados. É feito isto enviando o texto pretendido a ser traduzido para uma função em JavaScript e, chamando, então a API do Yandex ele traduz o texto e devolve-o para a View poder mostrar ao utilizador.

* **Integração de WebSockets**

O nosso projeto não possui integração com WebSockets.

* **Descrição de Testes feitos**

|  |  |
| --- | --- |
| Registar novo utilizador | Pass |
| Acesso protegido com password (todas as páginas exceto pesquisas) | Pass |
| Indexar iterativamente ou recursivamente todos os URLs encontrados | Pass |
| Indexar novo URL introduzido por administrador | Pass |
| Pesquisar páginas que contenham um conjunto de palavras | Pass |
| Resultados ordenados por número de ligações para cada página | Fail |
| Consultar lista de páginas com ligações para uma página específica | Pass |
| Consultar lista de pesquisas feitas pelo próprio utilizador | Pass |
| Dar privilégios de administrador a um utilizador | Pass |
| Entrega posterior de notificações (offline users) | Pass |
| Notificação imediata de privilégios de administrador (online users) | Fail |
| Página de administração atualizada em tempo real | Fail |
| Atualização imediata da lista de servidores multicast ativos | Fail |
| Associar conta de utilizador ao Facebook | Fail |
| Partilha da página com o resultado de uma pesquisa no Facebook | Fail |
| Mostrar em cada resultado a língua original da página | Pass |
| Traduzir título e descrição das páginas para Português | Pass |
| Registo com a conta do Facebook (sem conta ucBusca) | Pass |