Universidade de Coimbra

Faculdade de Ciências e Tecnologia

Licenciatura em Engenharia Informática

*2º Trabalho Prático – Aplicação Distribuída ‘Web-based’*

*Sistemas Distribuídos, 1º Semestre*



**RELATÓRIO**

**Trabalho realizado por:**

* **Ivo Correia nº 2008110814**
* **João Barbosa nº 2008111830**

Coimbra, 10 de Dezembro de 2010

**Índice**

[Introdução 2](#_Toc279851748)

[Principais Estruturas do Sistema 3](#_Toc279851749)

[Aplicação *Web-based* 3](#_Toc279851750)

[Servidor Intermédio 4](#_Toc279851751)

[Cliente 5](#_Toc279851752)

[*Login*, registo e invalidação de sessões 5](#_Toc279851753)

[*Chat* 5](#_Toc279851754)

[Realização de uma aposta e actualização dos jogos 5](#_Toc279851755)

[Actualização da lista de clientes *online* 6](#_Toc279851756)

[REST 7](#_Toc279851757)

[Notícias 7](#_Toc279851758)

[Tratamento de falhas 7](#_Toc279851759)

[Especificação de Testes 8](#_Toc279851760)

[Tabelas de Testes 8](#_Toc279851761)

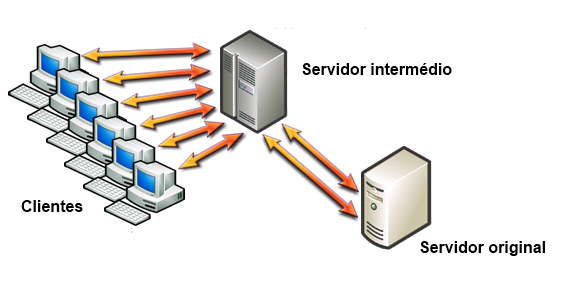
[Conclusão 12](#_Toc279851762)

[Bibliografia 14](#_Toc279851763)

# Introdução

Depois de realizado o primeiro trabalho prático, foi-nos pedido que o estendêssemos para uma aplicação “Web-based”, isto é, criar uma plataforma Web que disponibilizasse as mesmas ferramentas que a aplicação por consola e introduzisse um novo elemento, notícias actualizadas do “The Guardian”, recorrendo à tecnologia REST.

Deste modo, não só tivemos de construir um *site* como tivemos de garantir a interoperabilidade entre os diversos elementos (i.e. *site* e clientes TCP e RMI já existentes para o trabalho prático anterior). Por outro lado, deparámo-nos com um modelo *Three Tier*, exemplificado na imagem em baixo.



*Imagem 1 - Three Tier Model para o nosso projecto.*

Do lado esquerdo, temos os clientes que usam as aplicações *Web-based* para aceder aos serviços BetAndUin. No entanto, ao contrário do cenário do primeiro trabalho prático, estes não comunicam directamente com o servidor principal (na figura, o servidor original), mas sim com o servidor *TomCat*, referenciado como servidor intermédio. São assim feitos os pedidos primeiro para o servidor intermédio e este fica encarregue de comunicar, agora sim directamente com o servidor principal.

Este relatório vai seguir um formato em tudo idêntico ao anterior, onde vamos começar por fazer uma exposição geral do trabalho, indo aos poucos aprofundando e revelando os pormenores mais subtis e relevantes.

É ainda de notar que neste trabalho prático não foram solicitadas soluções para o caso de falhas do servidor, uma vez que tal tarefa encontrava-se englobada no âmbito do primeiro trabalho. Deste modo, corremos o servidor principal do primeiro trabalho (que note-se teve de sofrer leves alterações para se adaptar à nova aplicação) e não pomos em causa a sua fiabilidade.

# Principais Estruturas do Sistema

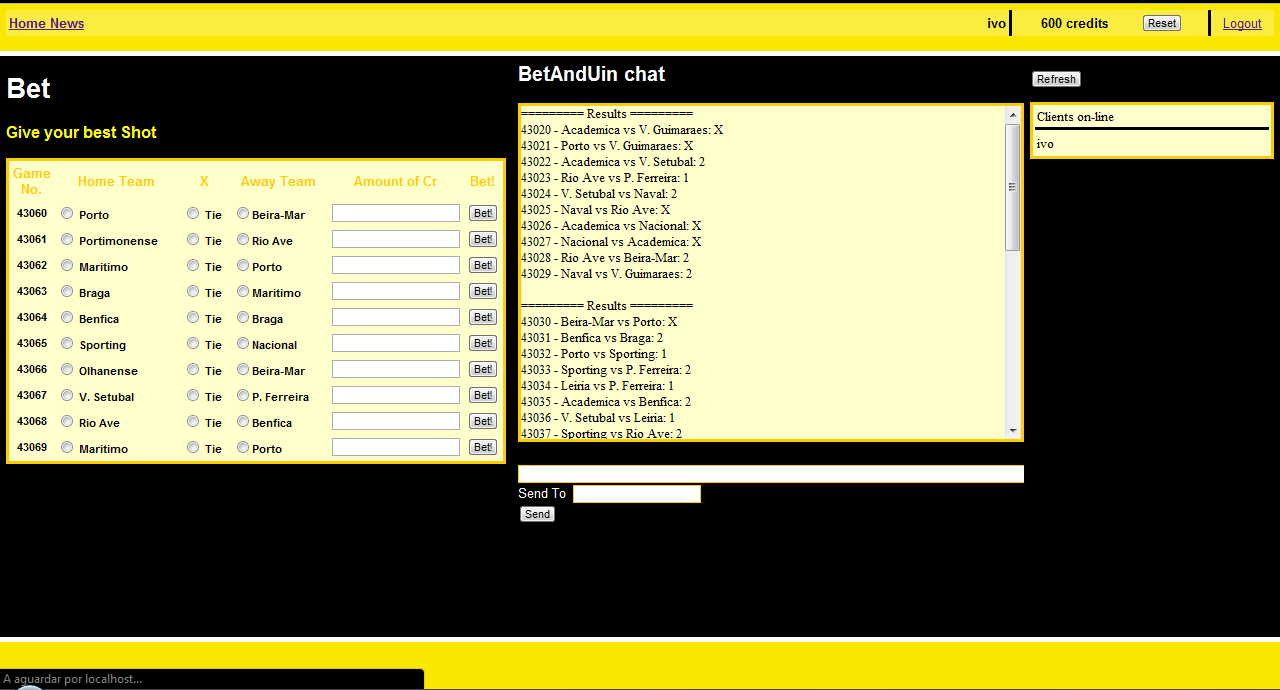
Este trabalho prático envolveu menos elementos do que o anterior, sendo que no entanto é de notar o aumento da complexidade de algumas das partes, especialmente no que se refere à aplicação de interacção com o cliente, já que vimos simplificada a interacção com e entre servidores.

Assim, neste capítulo, iremos iniciar a discussão da aplicação *Web-based* e uma vez observadas todas as suas principais funcionalidades, iremos abordar a arquitectura do servidor intermédio.

## Aplicação *Web-based*

Para aceder à aplicação, precisamos de inicializar todos os servidores de antemão e uma vez concluída esta tarefa, apenas recorremos a um *browser* e acedemos ao seu endereço como se tratasse de um qualquer *site* (estando a correr na nossa máquina, o endereço será *localhost:8080/BetAndUinWeb/Pages/Login.jsp* ).

Uma vez na aplicação, deparámo-nos com a página inicial, onde o utilizador deve inserir um *login* válido para continuar. Caso ainda não tenha um, pode sempre recorrer ao *link* para o registo de novos utilizadores. O texto, na caixa de cor amarela do lado direito, vai dando informações ao utilizador caso haja alguma anomalia com o *login*, como por exemplo nomes de utilizador ou palavras-chave erradas, ou então tentativas de autenticação de utilizadores activos no sistema. Uma vez dentro do sistema, o utilizador é presenteado com uma página que permite realizar todas as actividades de apostas nos servidores BetAndUin.



*Imagem 2 - A página principal do nossa aplicação.*

Do lado esquerdo, temos uma *frame* onde são registados os jogos actuais, assim como campos para a realização de apostas. Ao centro, temos uma janela onde são impressas todas as mensagens do *chat*, sendo que por baixo dessa mesma janela, temos os campos para enviar mensagens.

No topo, encontramos à esquerda um *link* para a página que contêm as notícias mais recentes relacionadas com o campeonato português de futebol, recorrendo à versão *online* do jornal “The Guardian”, tal como é explicado mais à frente. À direita, é impresso o nome de utilizador, o número de créditos desta conta, um botão para realizar o *reset* dos créditos e um *link* final para realizar o *logout*.

Por fim, à direita, temos a coluna onde são apresentados os clientes que também estão activos, sendo que é possível pressionar o seu nome com o ponteiro do rato o seu nome, que este irá aparecer automaticamente no campo ‘*Send To*’ do *chat*, deste modo poupando o utilizador o trabalho de digitar o nome do destinatário sempre que queira enviar mensagens.

Relativamente à página de notícias, esta contêm diversos *links* que uma vez pressionados, apresentam uma pequena mensagem uma imagem revelando o seu conteúdo.

## Servidor Intermédio

Tal como já foi dito na introdução, o servidor *TomCat* fica responsável pela ponte entre os clientes *Web* e o servidor principal BetAndUin, sendo que sempre que existem mensagens para todos os clientes, o servidor principal apenas necessita de envia uma dessas mensagens para o servidor *Tomcat* que este se encarrega de a espalhar pelos clientes *Web* a si ligados (detalhes a serem discutidos no capítulo TODO x). De uma maneira geral, este servidor vai funcionar como mais um cliente RMI, que faz e envia pedidos para o servidor principal.

Este servidor é constituído por três *servlets*, nomeadamente a *WebServlet*, *Logout* e a *BetServlet*. A primeira fica encarregue de aceitar pedidos de *login* e registo, obtendo os dados necessários para validar a sessão, guardar um registo do cliente em questão e ainda transmitir ao servidor principal a intenção deste novo cliente de entrar no sistema.

A *servlet* *Logout* é apenas usada quando o utilizador pressiona o botão de *logout* ou então fecha o *browser*, sendo que em ambos os casos esta estrutura tem a função de invalidar a sessão agora termina e limpar o registo do cliente das listas de clientes activos do servidor principal.

Por fim, a terceira, *BetServlet*, tal como o nome indica, fica encarregue de actualizar a lista de jogos que é impressa no *browser* de cada utilizador, assim como interagir com o servidor principal sempre que um cliente faz um pedido para realizar uma aposta. Por outro lado, esta *servlet* também é invocada sempre que ocorre uma alteração no número de créditos do utilizador e é preciso actualizar o seu número no topo da página.

# Cliente

A aplicação cliente apenas interage directamente com o servidor *Tomcat* e com os objectos RMI aí criados. Por outro lado, foi posta de lado a interface por linha de comandos presente no primeiro trabalho prático para dar lugar a uma interface *Web* que não só é muito mas apelativa ao utilizador, como se torna mais útil e permite explorar todos os conhecimentos a serem adquiridos através da realização deste trabalho prático.

## *Login*, registo e invalidação de sessões

Tanto o *login* como registo de novos cliente processa-se por duas fases. Primeiro, é enviado um GET com os dados obtidos dos formulários de cada página para a *WebServlet*. Esta, faz uma chamada RMI para o servidor principal e fica à espera da resposta. Uma vez obtida, uma dada mensagem (relacionada com o conteúdo da resposta do servidor principal) é registada na sessão do utilizador e neste momento, abrem-se dois cenários:

* Ou a resposta vinda do servidor principal é favorável e o cliente é redireccionado para a página de apostas;
* Ou por outro lado, a resposta é negativa e o utilizador permanece na mesma página. Contudo, nesta situação, ao fazer o *refresh* da página, é detectado uma sessão já existente e será disposta a mensagem justificando o erro encontrado, tal como conta já existente ou *login* inválido, por exemplo.

As invalidações de sessão ocorrem quando o cliente efectua um *logout*, tanto explicitamente (pressionando o *link* no topo direito da página) ou então fechando a página, sendo que esta última situação é detectada pela *ChatServlet*, uma vez que corre *COMET*, e esta toma as medidas adequadas para proceder a um término limpo e livre de problemas.

## *Chat*

Para a construção do *Chat*, recorremos à tecnologia COMET de modo a lidar com eventos assíncronos (entenda-se mensagens vindas de outros utilizadores ou mesmo do servidor) e evitar os periódicos refrescamentos das páginas.

De modo a poder comunicar com os clientes, é criado um cliente RMI na *WebServlet* que é referenciado no servidor principal para este poder realizar os *callbacks* sempre que necessita de enviar mensagens. Quando tal acontece, é invocado um método deste cliente, passando como argumento o destinatário. Deste modo, o cliente RMI invoca em primeiro lugar um método da *ChatServlet*, a qual suporta COMET, para este se encarregar de transmitir a mensagem final para o ecrã do utilizador.

## Realização de uma aposta e actualização dos jogos

Em segundo, a mesma mensagem é analisada e se se concluir que é do tipo de actualização dos resultados dos jogos, então realiza-se uma nova chamada de um método, desta vez da *BetServlet*. Esta dá ordem de recarregamento da lista de jogos, assim como o número de créditos do utilizador, visível no topo da página.

Para realizar apostas, também recorremos ao COMET, de modo a fazer com que a *BetServlet* seja totalmente responsável pela secção dos jogos. Assim, sempre que é realizada uma aposta, é enviado um GET para a *servlet*, com a informação necessária para realizar tal operação. Por seu lado, este evento é detectado pelo COMET (como um evento do tipo BEGIN) e se o tipo corresponder a uma ‘*bet*’ (aposta), é invocado o método remoto de apostas do servidor principal.

## Actualização da lista de clientes *online*

Para terminar este capítulo, analisemos como é feita a actualização automática das lista de clientes.

Sempre que um cliente entra ou sai da sala de *chat*, o servidor informa todos os apostadores com uma sessão activa de tal evento, sendo que a notificação é feita através de uma mensagem especial iniciada por ‘*BetAndUinChat:* ‘. Deste modo, sempre que é enviada uma mensagem para o *chat*, esta é analisada recorrendo à função de *callback* do COMET.

Como é que é detectada esta especificamente esta mensagem? Simplesmente temos de guardar o número de caracteres já lidos até à recepção da mensagem (uma vez que na variável *response*, devolvida pelo COMET, vem todo o histórico da conversação) e encontrar a *substring* que vai desse ponto até ao final de toda a resposta.

Obtida a última parte da mensagem, apenas verificamos se esta é iniciada pela tal sequência acima referida. Se houver uma correspondência, então apenas temos de fazer um recarregamento da lista de clientes.

# REST

O nosso *site* também contêm uma secção de notícias que obtêm imagens e títulos mais relevantes a partir do jornal “The Guardian”, sendo que são feitas três pesquisas, cujos conteúdos estejam relacionadas com as palavras-chave Benfica, Sporting e Porto.

## Notícias

O modo de obtenção das notícias foi fortemente baseado nos conteúdos das fichas práticas, disponibilizados pelos docentes da cadeira. A única alteração de maior que teve de ser feita prende-se com o facto das funções disponibilizadas apenas permitirem a devolução de um *array* com o resultado de uma dada pesquisa.

Deste modo, tivemos de criar um método que aceitasse um *array* de *strings* com todas as palavras-chave a procurar e que iterasse por essas mesmas palavras. Uma vez feita uma pesquisa, o resultado é posto num *array list* e passa-se para segunda pesquisa. Concluído o processo, o resultado é colocado no posto no mesmo *array list* sequencialmente, sendo que no fim iremos ter os três *arrays* de notícias todos juntos num só.

Assim, quando o ficheiro JSP apenas irá sempre ver os resultados como se tratasse de uma única pesquisa, o que facilita a colocação (e posterior substituição) das imagens e textos disponibilizados pelo “The Guardian”.

## Tratamento de falhas

Por outro lado, tal como foi requerido no enunciado do projecto, também tivemos de ter atenção à possibilidade de ocorrerem falhas na ligação ou de temporária indisponibilidade do serviço por parte dos servidores do “The Guardian”.

Deste modo, sempre que fazemos um pedido, em caso de falha, o *array* devolvido pelo método *mergedHeadlines()* irá ser *null* e consequentemente, antes de dispor na página todos os conteúdos, apenas temos de verificar a validade deste mesmo *array* e se não for válido, então simplesmente imprimimos uma página informando o utilizador que foi encontrado um problema na ligação e para voltar a tentar mais tarde.



*Imagem 3 - Imagem apresentada em caso de falha.*

# Especificação de Testes

De modo a confirmar que toda a arquitectura funciona em pleno e responde às situações de falha mais relevantes, realizámos diversos testes, entre os quais incluímos o cenário de duplo servidor primário (iSTONITH).

## Tabelas de Testes

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 1: Efectuar o *login* / registo e correr a aplicação | | |
| **DESCRIÇÃO:** | | O utilizador inicia a aplicação, tentando efectuar o *login* caso já tenha uma conta no sistema ou então proceder ao registo caso contrário.  Uma vez dentro do sistema, o utilizador enviará qualquer um dos comandos disponíveis. | | |
| **PRÉ-REQUISITOS:** | | O utilizador introduz toda a informação correctamente. Caso contrário, a aplicação rejeitará os comandos. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1 a). | O utilizador insere um novo registo. | | Se o nome de utilizador já se encontrar em uso ou então for a *keyword* ‘*all’*, o registo não deve ser efectuado.  Caso ainda esse nome ainda estiver disponível, o registo é realizado com sucesso e a aplicação automaticamente faz o *login* no sistema. Caso contrário é apresentada uma mensagem de erro. | Todos os resultados observados para este caso de teste estão de acordo com o resultados esperados. |
| 1 b). | O utilizador envia um comando de *login*. | | Se o utilizador não estiver registado no sistema, a palavra-chave não estiver correcta ou outro utilizador já está no sistema usando o mesmo nome de utilizador, o cliente é notificado no quadro lateral direito e o *login* não é concluído.  Caso contrário, o utilizador pode começar a utilizar as funcionalidades do programa sendo redireccionado para a página principal. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N**OME DO TESTE:** | | Caso de Teste 2: Logout | | |
| **DESCRIÇÃO:** | | Teste à saída do utilizador. Pode recorrer ao comando disponibilizado Logout, ou fechando simplesmente o browser. | | |
| **PRÉ-REQUISITOS:** | | Utilizar Mozilla Firefox ou Google Chrome e estar autenticado no sistema. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | Usa-se o *link logout* disponibilizado no topo da página. | | O cliente após clicar na ligação é redireccionado para a página inicial de login e o logout no servidor é realizado com sucesso. | Para o passo 1 e 2a) os resultados verificados vão de encontro aos esperados. No entanto no passo 2b) se estivermos a usar Firefox, o *logout*  é feito correctamente no servidor, se for Google Chrome a sessão permanece aberta. |
| 2 a). | O cliente simplesmente fecha a *tab* do *browser*. | | Se o cliente fechar simplesmente a *tab* do *browser* onde está a página principal da sua conta, será feito o *logout* no servidor automaticamente evitando assim que num sítio público a conta fique aberta acessível por qualquer um. |
| 2 b). | O cliente simplesmente fecha o *browser*. | | Se o cliente fechar simplesmente o *browse*, será feito o *logout* no servidor automaticamente evitando assim que num sítio público a conta fique aberta acessível por qualquer um. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 3: Entrada e saída de clientes do sistema | | |
| **DESCRIÇÃO:** | | Testamos o caso em que um cliente entra e sai do sistema, verificando se nos é dada alguma informação sobre estes eventos e se a lista de utilizadores *on-line* é actualizada. | | |
| **PRÉ-REQUISITOS:** | | Estar autenticadono sistema pelo menos um utilizador que analisa os eventos gerados pelo *log in*  *log out*  de outros utilizadores. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | Um cliente faz *log in* | | Quando este evento é gerado, aparece uma mensagem na janela de chat, indentificando o utilizador que acabou de entrar e inserindo-o na lista de utilizadores *on-line* | Todos os resultados observados para este caso de teste estão de acordo com o resultados esperados. |
| 2 | Um cliente faz *log out* | | Quando este evento é gerado, aparece uma mensagem na janela de chat, indentificando o utilizador que acabou de sair removendo-o da lista de utilizadores *on-line.* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 4: Realização de apostas | | |
| **DESCRIÇÃO:** | | Tal como indica, neste caso de teste, o cliente insere os créditos escolhendo uma equipa de um determinado jogo. | | |
| **PRÉ-REQUISITOS:** | | O cliente necessita de ter a sua sessão iniciada. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | O cliente não insere os créditos ou não selecciona nenhuma equipa ou empate. | | Surgirá um *pop-up* indicando que o utilizador tem de preencher o campo dos créditos e que tem de escolher uma equipa ou um empate. | Todos os resultados observados para este caso de teste estão de acordo com o resultados esperados. |
| 2. | O cliente possui zero créditos ou um montante inferior ao postado e tenta apostar. | | Surge uma mensagem na janela de *chat* indicando que o utilizador não possui créditos suficientes. |
| 3. | O cliente realiza uma aposta com sucesso. | | Surge uma mensagem na janela de *chat* indicando que a aposta foi realizada com sucesso, retirando de imediato o montante de créditos apostados ao indicado na barra superior da janela justaposto ao *username*. |
| 4. | O cliente ganha ou perde uma aposta. | | Perdendo ou ganhando uma aposta surge sempre uma mensagem na janela de *chat* indicando que tal aconteceu no fim de cada ronda –quando se sabe o resultado. Caso o utilizador ganhe a aposta o montante deste é actualizado somando o triplo do valor que apostou. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 5: Actualização dos jogos | | |
| **DESCRIÇÃO:** | | A lista de jogos em que é possível apostar é actualizada automaticamente assim como a lista de resultados. | | |
| **PRÉ-REQUISITOS:** | | O cliente necessita de ter a sua sessão iniciada. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | Iniciar/terminar de uma nova ronda. | | A lista de jogos para o utilizador apostar é actualizada automaticamente, mostrando no fim de cada ronda os resultados dos respectivos jogos. | Os resultados observados para este caso de teste estão de acordo com o resultados esperados. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 6: Aceder directamente aos frames/páginas pelo url | | |
| **DESCRIÇÃO:** | | Neste caso de teste, o utilizador tenta aceder directamente aos frames ou mesmo a uma página inteira directamente pelo url sem estar autenticadono sistema. | | |
| **PRÉ-REQUISITOS:** | | O utilizador não está autenticado | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | O utilizador tenta aceder directamente aos frames ou mesmo a uma qualquer página directamente pelo url. | | O cliente é redireccionado para a página de *logIn*. | Os resultados observados para este caso de teste estão de acordo com o resultados esperados. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 7: Envio de mensagens | | |
| **DESCRIÇÃO:** | | O cliente envia um mensagem para um grupo de utilizadores. | | |
| **PRÉ-REQUISITOS:** | | O cliente já tem de estar autenticado no sistema. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | O cliente envia um mensagem para todos os utilizadores. | | Todos os utilizadores que estão *online* (exceptuando o emissor) recebem a mensagem, sendo que o emissor é notificado do sucesso do envio. | Todos os resultados observados para este caso de teste estão de acordo com o resultados esperados. |
| 2 | O cliente envia uma mensagem para um utilizador específico. | | Se o destinatário está *offline* ou não estiver registado, o cliente é notificado de tal situação e a mensagem é descartada.  Se o destinatário estiver *online*, a mensagem é entregue e o emissor é notificado. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NOME DO TESTE:** | | Caso de Teste 8: Consultar as notícias | | |
| **DESCRIÇÃO:** | | O cliente visualiza um conjunto de 30 notícias sobre os três grandes do Futebol Português. | | |
| **PRÉ-REQUISITOS:** | | O cliente já tem de estar autenticado no sistema. | | |
|  | | | | |
| **PASSO** | **EVENTO** | | **RESULTADOS ESPERADOS** | **RESULTADOS OBSERVADOS** |
| 1. | O cliente clica no link para as notícias e a ligação é estabelecida com sucesso. | | É apresentada a lista de notícias em que cada é clicável para obter uma imagem e uma pequena descrição da notícia. | Todos os resultados observados para este caso de teste estão de acordo com o resultados esperados. |
| 2 | O cliente envia uma mensagem para um utilizador específico. | | É apresentada uma mensagem de erro acompanhada de um *cartoon*. |

# Conclusão

O sistemas distribuídos indiscutivelmente avolumam exponencialmente o número de problemas que encontramos para um sistema fechado e isolado. Existe o problema da sincronização entre todos os intervenientes; ao contrário de um sistema fechado, parte da rede pode falhar sem que a outra saiba do sucedido; a própria ligação entre os intervenientes está sujeita a interferências, o que, no caso de um sistema restringido ao seu espaço de endereçamento, raras vezes acontece sem ser por erros de programação.

Assim, o programador fica impossibilitado de evitar grande parte dos problemas existentes, pois um considerável quantidade deles depende de factores externos e estocásticos. Cabe então aos responsáveis prever as dificuldades com as quais as aplicações podem eventualmente confrontar-se e implementar mecanismos que lidem com as adversidades. Foi isso que procurámos fazer, com o máximo de pormenor permitido pelos nossos conhecimentos e experiência na área.

Apesar das soluções que apresentámos ao longo do relatório serem adequadas a um ambiente de desenvolvimento académico, claramente estão longe de cobrir todas as possibilidades de falha, especialmente se transportarmos o nosso sistema para o mundo global e, hipoteticamente, permitíssemos milhares de clientes ligados simultaneamente.

Em primeiro lugar, talvez precisássemos de um maior número de servidores secundários, ou mesmo vários servidores que trabalhassem em conjunto como um único servido primário. O protocolo de troca de mensagens e eleição do servidor primário teria de ser repensado, apesar de certamente partilhar muitas das características do protocolo usado na nossa aplicação.

Em segundo lugar, a forma de atender clientes TCP iria certamente explodir com os recursos do sistema. Tendo uma *thread* responsável por cada cliente que se liga, aumentando o número de clientes médios para as casas dos milhares e não das unidades, como temos agora, o número de *threads* que teriam de estar activas no servidor seria incomportável. Desse modo, uma solução possível seria criar uma *pool* de *threads* que permitisse gerir o número máximo de entidades activas no sistema.

Também não podemos esquecer que os nossos ficheiros e mecanismos para implementar a base de dados dos clientes são bastante rudimentares. Tal como é dito no enunciado, partimos do princípio que não existe falha na ligação entre servidor e sistema de suporte à persistência dos dados. Na realidade, como é natural, tal garantia não existe.

Ainda no campo da salvaguarda dos dados, o nosso sistema recorre aos ficheiros sempre que existe qualquer alteração, por mais pequena que seja. Enquanto no nosso pequeno sistema tal *overhead* não será demasiado significativo, no mundo real, estaríamos a escrever e ler de ficheiros um elevado número de vezes num curto espaço de tempo. Para além disso, tudo é guardado em memória física, o que, devido à lentidão de acesso a disco, ainda atrasa mais o sistema.

Por seu lado, a assumpção de que não há falhas na ligação e processamento de informação pela base de dados também nos aliviou de um outro problema que poderá ser bastante grave. Imaginemos que um cliente realiza uma aposta, e esta é recebida pelo servidor. No entanto, antes de a conseguir processar, o servidor falha e a mensagem perde-se. Aqui, o cliente fica num dilema. Sabe que o servidor recebeu a mensagem, mas, não obtendo resposta, não sabe ao certo se a aposta foi processada ou não.

Como soluções, poderíamos permitir que um cliente apenas pudesse fazer uma aposta para um dado jogo. Assim, em caso de dúvida, voltaria a reenviar a mensagem e caso ela já tivesse sido processada, era apenas substituída. O problema seria, pelo menos no nosso caso, ao usar um vector, que teríamos sempre de percorrer toda a lista de modo a verificar se o cliente já tinha feito uma aposta para o dado jogo. Outras estruturas de dados e funcionamentos teriam de ser analisados. Como exemplo, poderíamos separar as apostas por cliente e dentro dessa separação, ter um campo destinado para cada jogo da ronda, onde era dito se havia ou não apostas já realizadas. Assim, teríamos de fazer duas procuras directas, dado o nome do utilizador e número do jogo em questão.

Contudo, apesar de todos estes pontos negativos que podem impossibilitar a nossa aplicação de crescer para um escala maior no imediato, consideramos o trabalho em muito satisfatório para os cenários propostos e requeridos, principalmente no que diz respeito à aquisição de conhecimentos e apresentação de problemas que nos ensinem a pensar de uma forma distribuída.

# Bibliografia

**IDE**

Java Eclipse IDE

**Sites Internet**

- Javaspecialists.eu java training: Resetting ObjectOutputStream by Dr. Heinz M. Kabutz. Disponível em: [http://www.javaspecialists.co.za](http://www.javaspecialists.co.za/) Acesso em 23/10/10

-

**Imagens  
 Capa:** BWIN Disponível em: <http://www.bwin.com/> Acesso em 20/10/10

**Diagrama –** Microsof Visio 2010