Universidade de Coimbra

Licenciatura Engenharia Informática
2014-2015

Sistemas Distribuídos - Projeto 1

Autores:
Bruno Martins - Nº2007183389

25 de Outubro de 2014

Conteúdo

1 Introdução

O presente relatório destina-se a apresentar a descrição do projecto 1 da cadeira de Sistemas Distribuídos intitulado *Meeto – Collaboration and Social Networking*.

Reuniões são necessárias quando diversas pessoas trabalham em conjunto. Alunos reunem-se para coordenar o seu esforço nos trabalhos práticos; trabalhadores reunem-se para discutir o estado dos projectos e gestores reunem-se para tomar decisões chave relacionadas com as suas organizações e por aí fora. É necessário um sistema que impeça as pessoas de gastarem tempo desnecessário em reuniões. Afinal de contas, tempo é o nosso recurso mais valioso.

Este projecto consiste então numa implementação de um sistema de gestão de reuniões, onde os utilizadores podem criar reuniões, convidar participantes, ter uma sala de chat para cada tópico da reunião e no final adicionar decisões chave a cada reunião. É também possível relacionar pessoas a certas tarefas - uma vez que uma reunião sem atribuição de tarefas concretas aos seus elementos de pouco serve.

2 Internal architecture

O esquema abaixo apresentado procura descrever de forma simples a arquitetura desenvolvida:

Esta aplicação encontra-se subdividida numa série de classes, cada uma responsável por integrar esta aplicação num todo coerente, onde se destacam:

RMIServer:

Dentro da classe **RMIServer** encontram-se os métodos essenciais da aplicação: registo, login, createMeeting, listAllMeetings, acceptMeeting, meetingOverview, addAgendaItem etc. É aqui que é criado o registry e onde é instanciado o objecto para ligação à base de dados MySQL. Dentro do RMIServer são também chamados vários métodos do TCPServer (callback), entre eles o **msgToMany** que trata de enviar uma mensagem a vários destinatários e o **msgsToOne** que envia diversas mensagens para o mesmo utilizador. Relativamente a threads, o RMI faz uma gestão interna automática das mes-

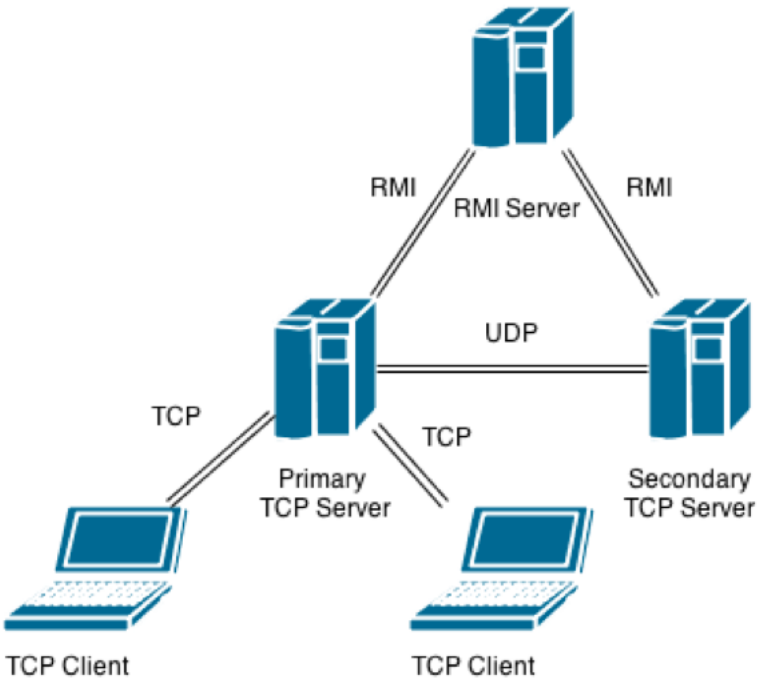

Figura 1: Visão Global da Arquitectura

mas. Aqui encontra-se também um HashSet com os utilizadores actualmente ligados ao sistema.

RmiInterface:

O RmiInterface, tal como o próprio nome indica, é a interface dos métodos existentes dentro da RMIServer.

MySQL:

Classe onde se encontram os métodos para ligação à base de dados MySQL e para correcta execução dos diferentes tipos de queries.

TCPServer:

Interface dos métodos existentes dentro do TCPServerImpl

TCPServerImpl:

Implementa os métodos da interface e é aqui onde os clientes se ligam (abrindo dois sockets e mais duas threads para cada um). Possui 2 threads internas (UDP Sender e UDP Receiver) e guarda a relação entre users e

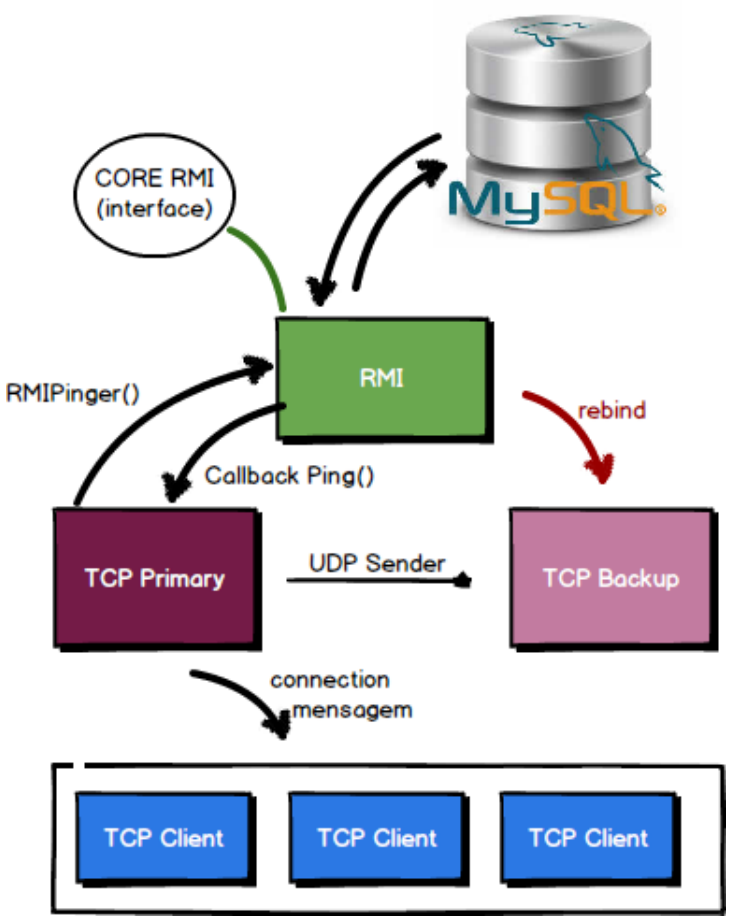

Figura 2: Arquitectura do Sistema

threads numa HashTable. Correndo o ficheiro duas vezes simula a arquitectura desejada (tcp server primário e tcp server de backup).

TCPClient:

Este cliente estabelece a ligação com o servidor através de dois Sockets TCP (uma para pedidos request-reply e outro para eventos em "tempo real") promovendo não só a construção dos menus (menu de login, inicial, etc.) como também a arquitetura da aplicação melhorando, deste modo, a comunicação com o servidor. É responsável por providenciar informação e dados através da criação de mensagens, como também requisita serviços à aplicação, como por exemplo listar meetings existentes, ou comprar aceitar convites. Possui internamente uma thread para ler dados que surjam sem ser necessário uma

interacção do utilizador com o sistema.

Estas classes são responsáveis por estabelecer grande parte da interoperabilidade desta aplicação embora sejam necessárias as restantes que suportam a estrutura por detrás deste sistema.

Relativamente ao armazenamento de dados na Base de Dados, foram criadas as tabelas:

member:

idmember, username, pass, email

A tabela *member* é a responsável por armazenar os dados dos membros.

meeting:

id_ideia, id_person, description, attach

A tabela **Ideias** além de ter o seu próprio id, também guarda o id da pessoa responsável pela mesma. Tem também um campo com uma breve descrição e um campo attach que é um **BLOB**, no qual são guardados os ficheiros de anexo.

Shares:

id_person, id_ideia, quantidade, price

A tabela **Shares** é a responsável pela ligação entre as shares das ideias e da respectiva pessoa que a detém. Tem um campo destinado à quantidade de shares e o seu preço.

Transactions:

id_transaction, id_ideia, id_person, quantidade, transaction_price, status

Na **Transactions** é registado as transações de ideias entre as diversas pessoas, bem como o preço a que esta foi transacionada e o seu status (se foi completado ou não).

Relation:

id_ideia1, id_ideia2, value

A tabela **Relation** faz a gestão entre as relações entre diferentes ideias e o valor da sua relação (1 - Positiva, 0 - Neutra, -1 - Negativa)

Ideias_Topic:

id_topic, id_ideia

A tabela **Ideias_Topic** é uma entidade fraca de ligação entre as tabelas *ideas* e *topic*

Topic:

id_topic, description

A tabela **Topic** armazena além do seu respectivo ID, uma breve descrição da mesma.

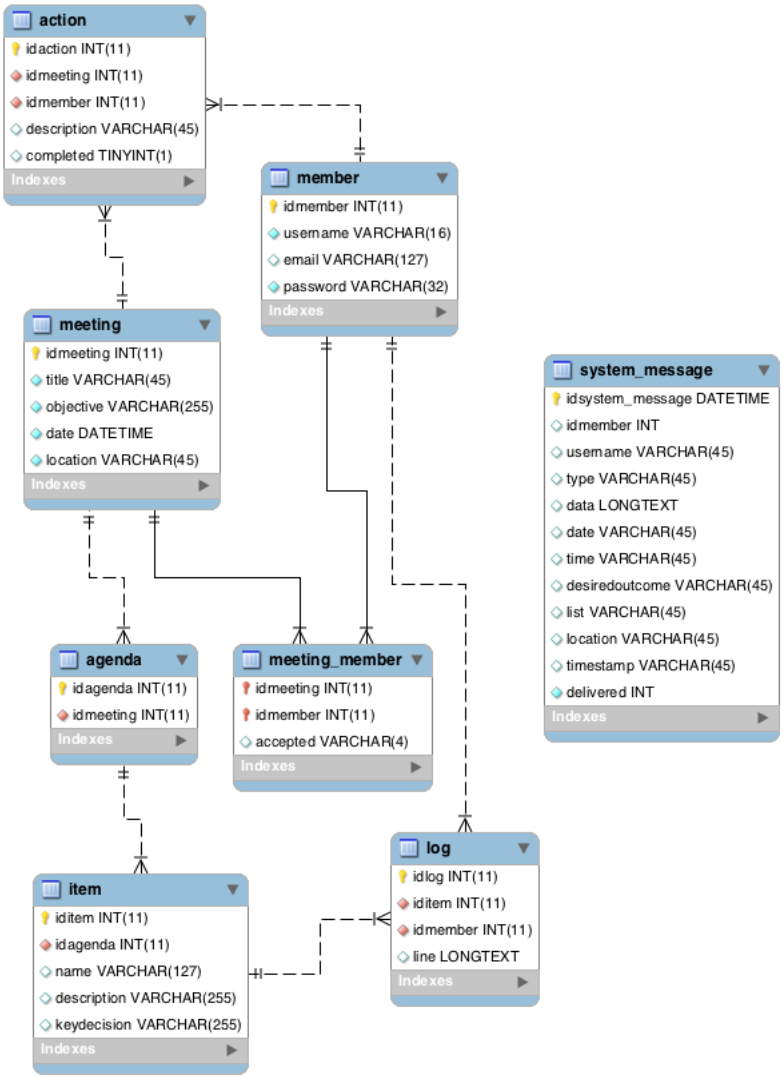

Figura 3: Diagrama Entidade Relacionamento

3 Data model

O seguinte diagrama representa o nosso esquema de Data Model.

A comunicação entre client-server pressupõe uma eficiente troca de dados que podem ser 'transmitidos' diretamente através de Sockets (TCP). Estruturámos a informação de forma eficiente e intuitiva construindo diversas classes capazes de representar este modelo de dados. Devido a esta separação em objetos das mensagens a transmitir entre as entidades executantes, optámos por utilizar ObjectStreams

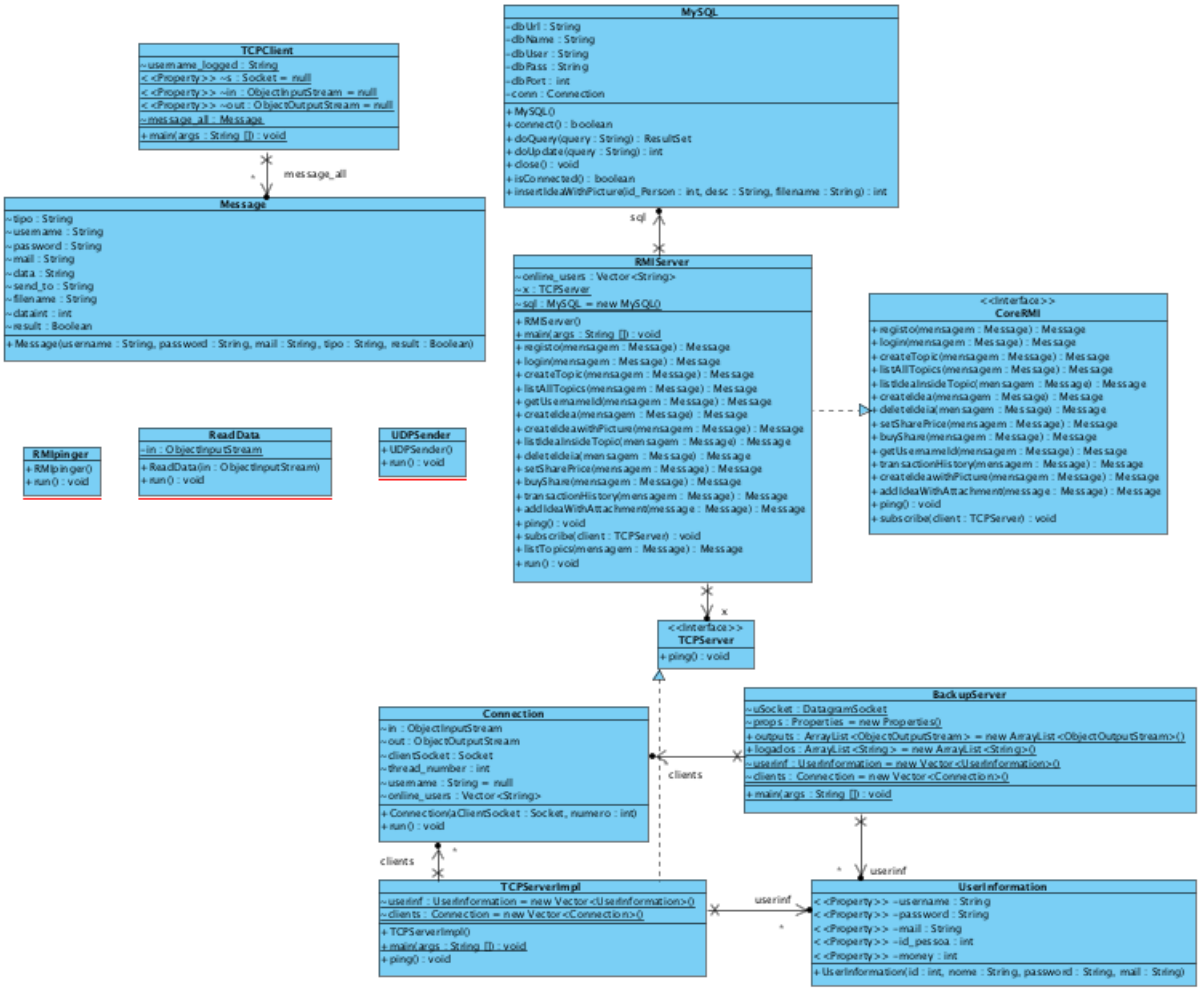

Figura 4: Data Diagram

(para Sockets TCP) salvaguardando toda essa informação em ficheiros de objetos garantindo, assim, a fiabilidade e integridade dos dados em trânsito (Message).

4 Protocol Specification (TCP, UDP, RMI)

4.1 TCP

O servidor cria um ServerSocket, bloqueia no método accept e espera novos clientes. Quando um cliente se liga ao servidor, este estabelece um socket de comunicação com o mesmo e cria uma nova thread para "tratar" desse cliente.

Desta forma, temos sempre o main bloqueado no listenSocket.accept à espera de novos clientes e preparado para criar uma thread por cliente (cada um tem a sua, Servidor Multithread).
Os métodos de que o servidor precisa para tratar dos pedidos de um cliente TCP estão criados no RMI Server. Quando termina de efectuar a acção requisitada pelo cliente, ela envia o resultado para o mesmo através de DataStream S.

4.2 UDP

A conexão UDP é estabelecida no início do funcionamento da aplicação de modo a assegurar o permanente controlo do estado de cada das entidades que supervisionam a gestão deste serviço. A diferença básica entre o UDP e o TCP é o fato de que o TCP ser um protocolo orientado à conexão e, portanto, inclui vários mecanismos para iniciar, manter e encerrar a comunicação, negociar tamanhos de pacotes, detectar e corrigir erros, evitar congestionamento do fluxo e permitir a retransmissão de pacotes corrompidos, independente da qualidade do meio físicos. No UDP não existem verificações, nem confirmações. É portanto uma escolha adequada para fluxos de dados em tempo real, especialmente aqueles que admitem perda ou corrompimento de parte de seu conteúdo.

4.3 RMI

O Java RMI(Remote Method Invocation) usa uma metodologia diferente da do TCP. Ao contrário deste não é estabelecido directamente um canal de comunicação como no TCP em que o servidor está a escuta num determinado porto e para cada cliente cria um socket. Especificamente, neste projecto o RMI é o nosso main-server, sendo este responsável pelos métodos existentes em toda a sua envolvente.

5 Architecture of File Transmission

O envio de ficheiros no nosso sistema foi feito recorrendo ao *blob* na base de dados MySql. A base de dados, na tabela *Ideias* tem um campo *attach*, no qual está sendo armazenado os ficheiros pretendidos pelo utilizador. O utilizador ao introduzir uma ideia, é-lhe perguntado se este pretende associar um ficheiro a esta

e de seguida coloca o path do ficheiro. Para fazer download de um attachment, o utilizador escolha mediante a apresentação da listagem de *ideias*, aquele que pretende guardar. Este será guardado na raiz da aplicação de onde o programa se encontra a ser executado.

6 Exception Handling

No que toca a tratamentos de excepção, foi assegurado que um Cliente reconecta ao mesmo servidor em caso de falha temporária do TCPServer (através de um tempo de espera), sendo a falha transitória e transparente para o usuário final. Através do uso de base de dados, o TCPClient não perde a ideia que está sendo escrita. O TCPServer reconecta em caso de perda de conectividade com o RMIServer e a falha transitória da RMIServer é transparente para o usuário final. Também houve o cuidado da verificação e foi assegurado que não existem ideias duplicadas, mesmo em cenários de entrega.

6.1 A nível de Base de Dados

O tratamento de erros do nosso MySQL é baseado em condições, que por sua vez, são nomeadas e preparadas para serem executadas mediante ao atendimento de tais condições. Além de tratarmos a violação de chaves primárias, foi tido em conta que os ids não assumam valores nulos (NULL), assim como houve um cuidado para que estes fossem auto-incrementáveis. Ao definir corretamente todas as relações e entidades fracas, foi possível haver um relacionamento directo entre entidades de diferentes campos (nomeadamente *id_person* na tabela *person* com o *id_person* da tabela *ideias*). Foi dado **cascade** em operações de *update* e *delete*, para que as ligações entre tabelas não fosse quebrada. Assim conseguimos garantir que ao apagar um *ideia*, a tabela de relação com uma share é também destruída.

6.2 TCP

De modo a tornar esta aplicação ao mesmo tempo transparente a erros e eficiente no seu tratamento procurámos controlar as exceções provenientes dos mecanismos de conexão (Sockets, RMI registry, etc.). Estes problemas advêm de falhas

de ligação entre o servidor e o cliente, de uma ruptura do socket TCP, etc. Implementámos diversas rotinas para gerir estas exceções (try/catch (IO Exception, Remote Exception)) em ambas as 'extremidades' da aplicação. As threads responsáveis por estas funções (ConnectBackupServer, UDP) permitem não só isolar o tratamento destas situações, podendo o cliente continuar a usar a aplicação (com algumas restrições), como também restabelecer as ligações entre as entidades executantes ou, em caso de insucesso, ativar um backup. Quando o servidor deixa de funcionar correctamente, vai ser detectada uma falha. O cliente irá então tentar voltar a conectar-se, sendo a função udpsender() a responsável pelas tentativas. Na tentativa do Cliente se conectar ao Servidor, este efetua uma espera de um intervalos de 3 segundos entre cada tentativa de conexão. Entretanto, caso o cliente tente realizar algum pedido, a thread destinada à escrita de teclado fica bloqueada até este realizar re-connect.

6.3 RMI

No RMI foram dados os mesmos tratamentos de exceção utilizados no TCP.

7 FailOver

O fail-over pressupõe a existência de dois servidores (primary e backup -> tipo 1 e 2 respetivamente) que comunicam entre si através de um canal UDP. Só assim é possível controlar o estado (ativo/inativo) de cada uma das entidades e efetuar a troca em caso de necessidade. Para tal, operámos um mecanismo de ping (3s) iniciado pelo servidor secundário que providencia informação essencial na gestão de erros. Implementámos uma thread com uma duração de 5 segundos que corresponde ao tempo máximo de delay aceite pela aplicação. Se durante esse período a 'entidade gestora' em utilização não comunicar, os clientes serão redirecionados para o servidor secundário que assumirá o papel de principal.

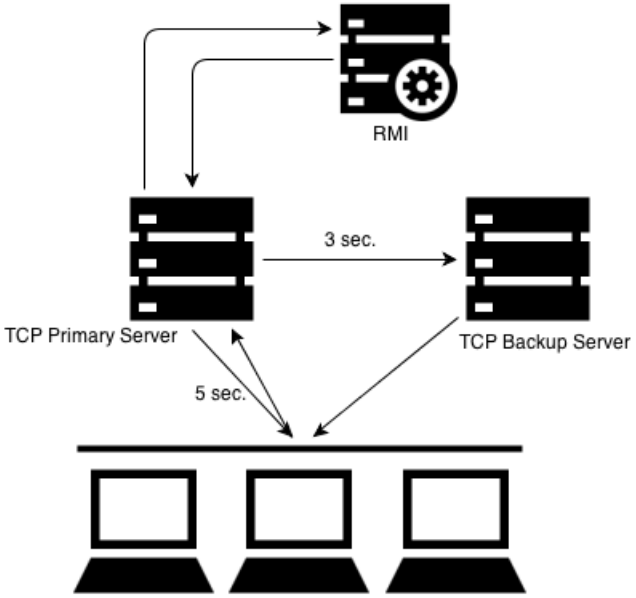

Figura 5: Esquema de FailOver

8 Causal Order Implementation

Para implementar o nosso projecto, começamos por criar as respectivas ligações e a criação dos clientes. Inicialmente criamos somente o TCP Server, no qual colocamos todas as operações necessárias e mais tarde passamos estas mesmas para o servidor RMI. Foi também criado uma interface *CORERmi*, na qual temos a declaração das funções necessárias ao longo da aplicação. Relativamente ao armazenamento de dados, optamos numa primeira instância por recorrer a ficheiros (tipo vector), mas logo percebemos que para existir uma maior segurança e fiabilidade de armazenamento de dados a melhor operação era recorrer a uma base de dados, nomeadamente o *MySQL*. Relativamente à base de dados, foi feito uma análise cuidada das tabelas, campos e ligações a usar de acordo com as nossas necessidades.

Por fim, foi criado o servidor backup e a sua respectiva ligação UDP. Somente no fim da implementação dos mecanismos de **FailOver** e dos mecanismos de **tratamento de excepções**, é que nos concentramos na parte dos objectos / requisitos funcionais.

9 Description of tests (Table with Pass/Fail)

Foram feitos diversos testes nas diversas funcionalidades da aplicação, de modo a assegurar a robustez da aplicação garantindo, assim, um sistema seguro e eficiente, procurámos testá-lo minuciosamente.

9.1 Teste de Registo

Test Name: Registar				
Description: Registo do cliente na BD da aplicação				
Prerequisites: O cliente tem de entrar na aplicação e escolhe a opção REGISTAR				
Setup: O cliente cria uma nova conta				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**Pass / Fail**
1	Escolhe a opção Registar	Sistema abre os campos de registo	São os esperados	PASS
2	Inserir username, password e email	Sistema aceita dados e guarda na BD ou rejeita caso o username escolhido já esteja a ser utilizado	São os esperados	PASS

9.2 Teste de Login

Test Name: Login				
Description: Identificação do cliente				
Prerequisites: Cliente tem de ter uma conta criada na aplicação				
Setup: O cliente insere a sua pass e username para usar a sua conta				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**Pass / Fail**
1	Inserir username, password	Sistema aceita dados se essa conta existir	São os esperados	PASS

9.3 Teste de Submissão de Ideia

Test Name: Teste de Submissão de Ideia				
Description: É testada a funcionalidade de introduzir ideia				
Prerequisites: O utilizador tem que já estar registado e já devem existir ideias				
Setup: O servidor irá fazer handle das ideias				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**P / F**
1	logar User	Login efectuado com sucesso!	Nada apontar	PASS
2	User acede ao menu e escolhe a opção 3	aparece o passo seguinte	antes de escolher a opção, aparece uma listagem de opções	PASS
3	Introduz a conteúdo da ideia	Pergunta se deseja introduzir anexo	nada apontar	PASS
4	Escolhe a opção de sim relativamente ao anexo	Passa ao passo seguinte	é introduzido a path do anexo	PASS
5	—	Passa À seguinte pergunta relativamente a adicionar a ideia a um tópico	pergunta se deseja introduzir um novo tópico ou usar os existentes	PASS
6	Escolhe a opção de usar tópicos existentes	Lista todos os tópicos existentes	nada apontar	PASS
7	Vai introduzir os tópicos 1 e 2 e por fim clica em 0 para finalizar	Ir pedindo mais tópicos enquanto não se clica na tecla 0	É introduzida a associação da ideia com o tópico, com sucesso	PASS

9.4 Teste de Ligação ao Servidor Backup

Test Name: Ligação ao Servidor Backup				
Description: Teste de Ligação ao Backup				
Prerequisites: Cliente tem de ter uma conta criada na aplicação e devem estar os dois servidores a correr (Principal e Backup). Será feita uma ligação normal entre cliente TCP e Servidor TCP. De seguida será quebrada a ligação do servidor principal TCP, esperando assim que o cliente se reconecte ao servidor backup				
Setup: Acesso normal À sua conta				
Step	**Operator Action**	**Expected Results**	**Observed Results**	**Pass / Fail**
1	Inserir username, password	Sistema aceita dados se a conta existir	São os esperados	PASS
2	O servidor principal é desligado	Existe um período de espera da parte do cliente até este se ligar ao servidor backup, podendo ter existido uma pequena baixa da parte do principal	O servidor principal não vem à vida dentro o tempo de espera	PASS
3	—-	Ligação feita ao servidor backup	São os esperados	PASS

10 Manual de Instalação

Para correr a aplicação irá necessitar dos seguintes requisitos:

- Java 8
- Servidor com MySQL com a respectiva base de dados (o script de criação da base de dados encontra-se na raiz deste projecto dentro do ficheiro *sd1_2013.sql*)

Após reunidos todos requisitos necessários, é necessário que o MySQL esteja a correr dentro da própria (localhost), ou no caso que esteja se encontre noutra máquina, que seja definido o respectivo IP e porta. Como foi mencionado anteriormente, deve-se correr o script *sd1_2013.sql* que contem todas as tabelas criadas bem como alguns dados. Por fim, deve-se executar-se por esta ordem a execução de Servidores da máquina:

- 1 - RMIServer.java
- 2 - TCPServer.java
- 3 - BackupServer.java
- 4 - TCPClient.java

Relativamente ao IP de máquina onde os servidores e o cliente estão a correr, existe um ficheiro intitulado **Property** onde se pode alterar os endereços e as portas onde estes vão estar a correr.

```
tcpip1=127.0.0.1
tcpip2=127.0.0.1
tcpserverPort=7000
tcpserverPortAux=7002
tcpbserverPort=7001
udpPort=5000
rmiServerip=127.0.0.1
rmiServerPort1=1099
rmiServerPort2=6001
```

E... pronto! É só escolher as operações pretendidas!