· U C ·

UNIVERSIDADE DE COIMBRA

Sistemas Distribuídos

META FINAL

Trabalho feito por:

>>> Francisco Quinaz

> UC 2014204636

>>> Jorge Marques

> UC 2014199355

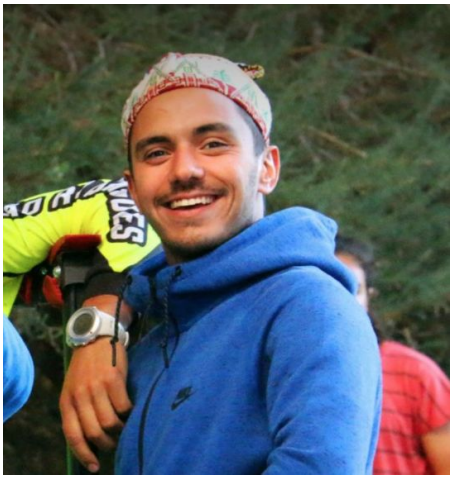
Sistemas Distribuídos

2017/2018

· U C ·

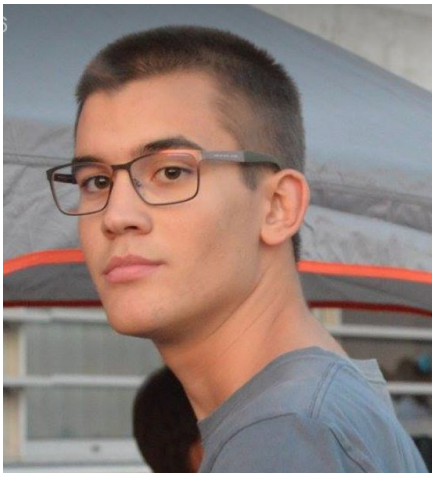
Universidade de Coimbra

A Equipa



Francisco Quinaz Marques

>>> UC 2014204636
>>> TCP Server
>>> Voter Web
>>> REST API



Jorge

>>> UC 2014199355
>>> RMI Server
>>> Admin Web
>>> WebSockets

Sistemas Distribuídos
2017/2018

· U C ·

UNIVERSIDADE DE COIMBRA

>>>1ªMETA<<<

>>> TCP

>>>Protocolo:

Para este projeto utilizámos a recomendação do projeto.

Todas as mensagens transmitidas entre Cliente e Servidor respeitam um conjunto chave-valor:

chave1|valor1;chave2|valor2

Podemos representar assim todas as variações aceites neste protocolo:

Sistemas Distribuídos

2017/2018

```
type:
 |   type|login;id|123123123;password|jorge
 |->login (type|login;)
 |     |
 |     |->id  (id|123123123;)
 |         |
 |         |->password  (password|jorge)
 |
 |   type|logout;
 |->logout (type|logout;)
 |
 |
 |->eleicoes_list
 |     |
 |     |       type|eleicoes_list;request|***;
 |     |->request  (request|***)
 |     |
 |     |     type|eleicoes_list;eleicoes_count|1;eleicao_nome|eleicao1;
 |     |->eleicoes_count  (eleicoes_count|2;)
 |            |
 |            |repetido #(eleicoes_count) vezes
 |            |->eleicao_nome  (eleicao_nome|eleicao1)
 |
 |
 |->listas_list
 |     |
 |     |       type|listas_list;eleicao|eleicao1;
 |     |->eleicao  (eleicao|eleicao1)
 |           |     type|listas_list;eleicao|eleicao1;
 |           |     list_count|1;eleicao_nome|eleicao1;
|          |->list_count  (list_count|2;)
 |                 |
 |                 |repetido #(eleicoes_count) vezes
 |                 |->eleicao_nome (list_nome|lista1)
 |
 |->vote
 |    |
 |    |     type|vote;vote|lista1;
 |    |->vote  (vote|lista1)
 |
 |->msg  type|msg;msg|Bem-vindo!;
 |    |->msg (msg|Bem-vindo!)
 |
 |->status    type|status;logged|on;msg|ja esta ativo;
      |
```

|->logged (logged|on)

|

|->msg (msg|ja esta ativo)

>>>Arquitetura:

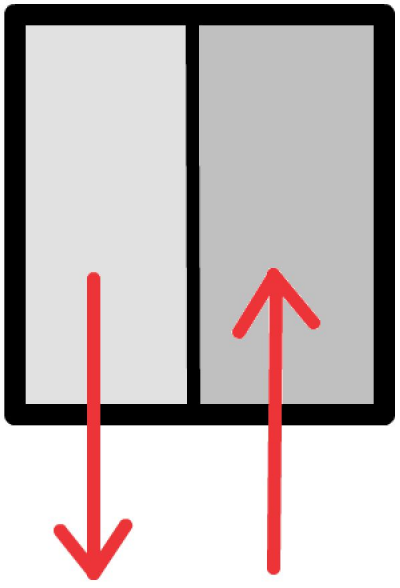
>>>Client

Quanto à arquitetura, o Cliente funciona com duas Threads:

-Uma apenas recebe a informação vinda do Servidor e decodifica-a respeitando o protocolo descrito em cima para apresentar um género de UI (facilitando a visualização da informação)

-A outra serve apenas para mandar informação ao Servidor, tendo esta que respeitar o protocolo em cima descrito.

TCPClient

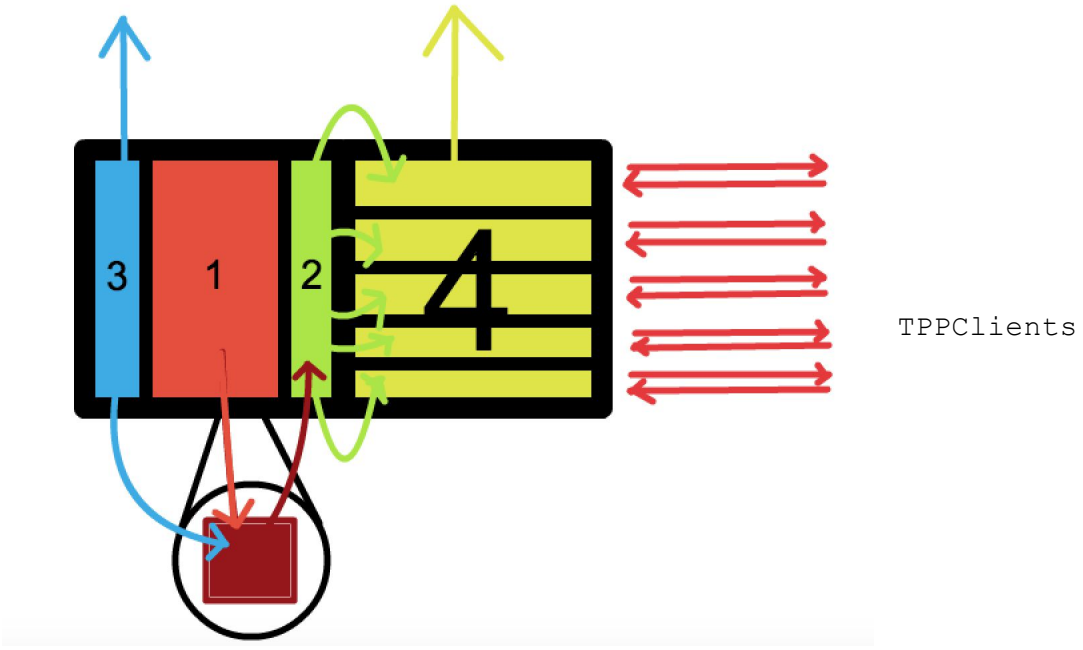


TCPServer

>>>Client

O funcionamento do Servidor TCP traduz-se na seguinte imagem:

rmiregistry voterRMIInterface



Cada número representa um tipo de Threads:

1- main Thread - esta Thread é responsável por ler o ficheiro de configuração, criar um objecto "Utilizadores" que vai guardar toda a informação sobre o sistema (threads 4 e toda a sua informação (socket,autenticação, bloqueio e servidor RMI))onde todas as outras threads podem aceder e aceitar conexões de TCP Clients. A sua aceitação consiste em ficar à escuta de um ServerSocket, onde, depois da sua

aceitação (Cliente) esta vai adicionar ao objeto "Utilizadores" a informação do cliente e a Thread 4 (Connection) criada para responder aos pedidos do cliente.

2- ConnectionManager - Para garantir o bloqueio dos terminais dos Clientes foi criada esta Thread.

Esta Thread corre num ciclo onde vai perguntar à pessoa responsável pela mesa de voto qual o terminal (TCPClient) a desbloquear e qual o cliente em causa para que este possa votar. Porém este desbloqueio necessita de uma identificação.

Essa identificação é executada na tentativa de desbloqueio do terminal. Esta Thread comunicará com o voterRMIInterface para ver se este cliente existe e se não está ativo num outro terminal. Caso o voterRMIInterface garanta estas condições, o terminal é desbloqueado **apenas** para esse eleitor.

A thread tem acesso ao objeto "Utilizadores" de modo a conhecer todas as conexões e o servido RMI a utilizar.

3- findRMI - Thread responsável pela atualização do servidor RMI em caso de um falhar.

Em intrevalos de 5 segundos, o findRMI vai fazer lookUp, ou seja vai procurar no rmiregistry o nome do servidor RMI, devolvendo uma referência para este. Esta referência é atualizada no objeto "Utilizadores" para que qualquer chamada da voterRMIInterface seja efectuada com sucesso.

4- Connection - Existem tantas Threads Connection quantas as conexões a terminais (TCPClient).

Estas Threads funcionam no modo pedido-resposta.

Enquanto o terminal (TCPClient) está bloqueado(quer seja por ausência de bloqueio inicial ou timeout de 2 minutos sem atividade) a Connection não irá responder a nenhum pedido do cliente.

Após o seu desbloqueio o eleitor terá que fazer login no terminal respeitando o id de desbloqueio (efetuado pelo ConnectionManager).A Connection irá então aceder ao servidor RMI actualizado (pela findRMI) para garantir o login:

```
users.getServer().authenticateVoter(no.getId(), message2[1])
```

Em caso de sucesso o eleitor pode proceder a todas as outras outras opções.

TODAS as chamadas ao servidor são confirmadas, caso não seja possível executar a chamada este vai tentar até conseguir ou até chegar ao timeout definido no ficheiro de configuração

Estas tentativas são executadas pelos métodos handleRemoteException****()**

Neste caso

```
try {
   if ((resp = users.getServer().authenticateVoter(no.getId(), message2[1])) !=
null) {

       //RMI PARA VALIDAR
       System.out.println(resp);
       ///
   }
} catch (RemoteException e1) {
   resp = handleRemoteExceptionAutenticateVoter(no);

}
```

Todas estas funções "**handleRemoteException******()** " têm um funcionameto semelhante.

Depois de autenticado, o eleitor pode então pedir ao servidor a listagem de eleições e a listagem de listas candidatas a cada eleição. processadas nas funções:

```
try {
   rs = users.getServer().listElections(no.getId());
} catch (RemoteException e1) {
   rs = handleRemoteExceptionEleicoesList(no);

}
```

e

```
try {
   rs = users.getServer().listLists(message1[1], no.getId());
} catch (RemoteException e1) {
   rs = handleRemoteExceptionListasList(message1[1], no);

}
```

Na votação, o Servior corre a função:

```
try {
    if ((resp = users.getServer().vote(no.getId(), message1[1], message2[1], 3)) !=
null) {

        //RMI PARA VALIDAR
        System.out.println(resp);
        ///
    }
} catch (RemoteException e1) {
    resp = handleRemoteExceptionVote(no, message1[1], message2[1], 3);
```

sendo que depois irá fazer automáticamente o logout.

O logout é chamado pelo eleitor, pelo voto(em caso de sucesso) ou pelo timeout. A função é:

```
if ((log = users.getServer().lockVoter(no.getId())) != null) {

        //RMI PARA VALIDAR
        System.out.println(log);
        ///
    }
} catch (RemoteException e1) {
    log = handleRemoteExceptionLockVoter(no);

}
```

>>> RMI

O servidor RMI começa por pedir o nome do ficheiro de configuração. Caso o ficheiro não esteja escrito corretamente o servidor vai criar um novo com valores default.
A thread principal começa por criar um objeto chamado UDPConnection. Este objeto serve apenas para trocar pings com o outro servidor RMI, de modo a assegurar que pelo menos um servidor RMI está sempre a funcionar.
Após criar um objeto UDPConnection, o servidor vai verificar na registry à qual se pretende ligar se já se encontra um servidor ligado no nome que este pretende assumir. A verificação é feita através de um lookup e uma chamada de uma função da API AdminVoterInterface chamada sayHello. Caso não seja atirada uma excepção significa que nenhum servidor está ligado naquele domínio. O servidor RMI assume assim as funções de servidor primário. Caso contrário assumirá as funções de servidor secundário.
O servidor secundário limita-se a receber pings do servidor primário através de uma socket UDP. Se não receber pings durante o timeout definido, este servidor vai tentar substituir o servidor primário.
O servidor primário começa por estabelecer uma ligação com a base de dados. O nome de utilizador e password para acesso à base de dados são fornecidos no ficheiro de configuração. Depois, cria uma thread que atualiza o estado das eleições (ativa/inativa) conforme a data atual. Depois entra num ciclo que envia pings em intervalos de 5 segundos ao servidor secundário através de DatagramPackets (UDP). Os ports das sockets UDP são definidos no ficheiro configuração.
As funções do RMI que serão invocadas pela consola Admin estão numa Interface chamada AdminRMIInterface, e as funções a serem chamadas pelos servidores TCP estão numa interface chamada VoterRMIInterface.
Todas as funções estão discriminadas no javadoc que segue com este relatório.

>>> Admin Console

O admin console é um cliente RMI que começa por ler as configurações de um ficheiro de configurações, caso o ficheiro esteja mal escrito, cria um novo com valores default. Após ler as configurações dá lookup a um domínio numa certa registry, ambos especificados no ficheiro de configurações. Depois cria um objeto que contém métodos que criam todos os menus necessários para as operações de admin. O método showMenu chamado é chamado em loop e trata do failover e das alterações feitas pelo admin.

>>> Failover

O failover do lado do servidor RMI é feito através de pings UDP.
O servidor principal envia pings ao servidor secundário em intervalos de 5 segundos. Quando o servidor secundário parar de receber pings durante mais de x segundos (x definido no ficheiro de configuração do RMI), o servidor secundário vai tentar substituir o servidor principal.
No lado da consola do admin ao invocar um método do servidor, caso este tenha ido abaixo vai ser atirada uma remoteException que ao ser apanhada vai fazer com que a consola espere 5 segundos, volte a dar lookup ao servidor RMI e volte a tentar invocar o método previamente chamado. Repete este processo até conseguir invocar o método com sucesso ou atingir o timeout. Verifica se atingiu o timeout sempre que apanha uma remoteException. Quando a invocação do método for bem sucedida a consola resume o funcionamento. Caso atinja o timeout a consola fecha e o utilizador é notificado.

CheckList

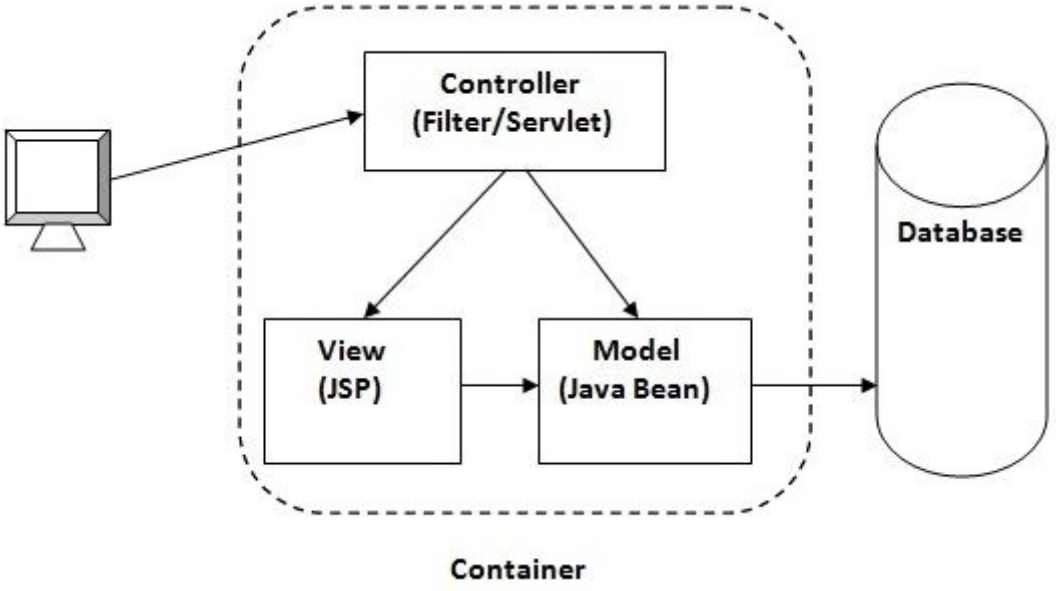
Requisitos Funcionais	
Registar pessoas (estudantes, docentes, ou funcionários)	√
Criar departamentos e faculdades	√
Criar eleição	√
Criar listas de candidatos a uma eleição	√
Adicionar mesas de voto a uma eleição	√
Identificar eleitor na mesa de voto e desbloquear terminal de voto	√
Autenticação de eleitor no terminal de voto	√
Votar (escolher, uma só vez, uma lista no terminal de voto)	√
Alterar propriedades de uma eleição	√
Saber em que local votou cada eleitor	√
Consolas de administração mostram mesas de voto on/off	√
Consolas de administração atualizadas em tempo real nas eleições	X
Eleição termina corretamente na data, hora e minuto marcados	√
Consultar resultados detalhados de eleições passadas	√
Tratamento de Exceções	
Avaria de um servidor RMI não tem qualquer efeito nos clientes	√
Não se perde/duplica votos se os servidores RMI falharem	√
Avarias temporárias (<30s) dos 2 RMIs são invisíveis para clientes	√
Terminal de voto bloqueado automaticamente após 120s sem uso	√
Crash de terminal de voto é recuperado	√
Grupos de 3: Crash de um servidor TCP é recuperado (-4)	√
Failover	
Heartbeats via UDP entre o servidor primário e o secundário	√
Em caso de avaria longa os servidores TCP ligam ao secundário	√
Servidor RMI secundário substitui o primário em caso de avaria long	√
Os dados são os mesmos em ambos os servidores RMI	√
O failover é invisível para utilizadores (não perdem a sessão)	√
O servidor original, quando recupera, torna-se secundário	√
Relatório	
Arquitetura de software detalhadamente descrita	√
Detalhes do funcionamento do servidor TCP (protocolo, etc.)	√
Detalhes do funcionamento do servidor RMI (API javadocs, etc.)	√
Distribuição de tarefas pelos elementos do grupo	√
Testes de software (tabela com descrição + pass/fail de cada teste)	√

· U C ·

UNIVERSIDADE DE COIMBRA

>>>2ªMETA<<<

>Arquitetura Struts2



Para este projeto foi usado um modelo MVC. Para isso foi usado a framework struts2 onde o model se chama bean, o controler action e a view jsp.
Existe um bean para cada modelo de informação, ou seja, os users têm um bean, as faculdades outro e por ai em diante. Todos os métodos dos beans, exceto os setters e os getters das suas variáveis, dizem

respeito a operações com o RMI. Estes métodos retornam o resultado da operação.
As actions têm métodos para ir buscar os beans, que são guardados na session, ou para criar um caso ainda não exista nenhum na session. Ao guardar os beans na session evitamos estar a criar vários beans repetidos para o mesmo utilizador.

Mas como é que a informação chega aos beans?

Toda a informação é submetida pelo utilizador através de forms que estão nas views. O form tem vários campos onde o utilizador escreverá o que é pedido. Quando o utilizador preencher todos os campos submete então o form.
É chamada então uma action que vai buscar o bean correto para o tipo de dados que foram submetidos à session. Caso não exista nenhum bean para este tipo de dados será criado um novo e posto na session.
Depois utilizando os setters do bean, a action, vai preencher as variáveis deste com a informação submetido pelo utilizador.
O utilizador é depois levado para uma página que vai buscar o bean, onde se encontram os dados submetidos à session e invoca o método do bean que retorna o resultado pedido pelo utilizador.
Este resultado é depois imprimido no ecrã.

>Proteções

Ao fazer login a informação do utilizador é guardada num bean que é guardado session, é também guardado na session um booleano para saber que este está logado e outro para saber se é ou não um admin.
Toda a navegação é feita através de actions e todas as actions antes de executarem verificam se os dados na session relativos ao login estão corretos.
Deste modo evitamos que a autenticação seja evitada através da navegação pelo URL.

· U C ·

Universidade de Coimbra

>Integração Struts2 com RMI

A ligação ao servidor RMI é feita durante a inicialização de um Bean, ou seja a ligação é estabelecida no construtor dos beans.
Após a ligação ser estabelecida o bean guarda a referência numa variável que servirá para chamar os métodos do servidor RMI.
Estes métodos são chamados nos métodos do bean que usam as variáveis deste, que terão os valores que lhe foram passados pelo user.
O resultado da chamada é depois devolvido pelo método do bean que fez a operação.
Existe também um handler caso a chamada do RMI falhe. Este handler tentará restabelecer a ligação com o servidor RMI de 5 em 5 segundas durante 30 segundos, onde ao fim destes simplesmente perderá a conexão e notificará o user.

Sistemas Distribuídos
2017/2018

>REST API

Na produção deste projeto utilizamos a REST API da plataforma Facebook.

De modo a garantir o bom funcionamento da mesma, e de forma segura, nunca o Access Token do Utilizador foi guardado em memória,assim, qualquer ataque feito à base de dados não irá comprometer a segurança dos nosso utilizadores.

Sempre que se necessita de alguma informação desta API, quer seja para fazer login, associar conta ou partilhar um voto, a 1ª action responsável pelo processamento do pedido irá utilizar a apiKey e o apiSecret obtido a partir de developers.facebook.com/apps em primeiro lugar para obter o URL de autorização de pedido:

```
String apiKey = "154164651894827";
        String apiSecret = "84b8ecde7405e2286b29c09cf0ee8de6";

        OAuthService service = new ServiceBuilder()
                                .provider(FacebookApi2.class)
                                .apiKey(apiKey)
                                .apiSecret(apiSecret)
.callback("http://localhost:8080/ProjetoSD_Meta2/voterfacelogin2") // Do not change this.
                                .scope("publish_actions")
                                .build();

        System.out.println("Fetching the Authorization URL...");

            this.authorizationUrl = service.getAuthorizationUrl(EMPTY_TOKEN);
        System.out.println("Got the Authorization URL!");

        //session.put("facebookurl", authorizationUrl);
        this.FacebookUrl=authorizationUrl;
```

Após esta etapa irá haver um redirecionamento para o URL anteriormente obtido com a finalidade de obter o código de autorização. Este redirecionamento irá chamar uma outra ação assim que concluído que irá obter o access token do cliente em causa:

```
Verifier verifier = new Verifier(getCode());

    // Trade the Request Token and Verfier for the Access Token
    System.out.println("Trading the Request Token for an Access Token...");

    Token accessToken = service.getAccessToken(EMPTY_TOKEN, verifier);
            System.out.println("Got the Access Token!");
```

Depois destas operações irá haver então a operação em causa a realizar com a REST API:
(Sempre seguindo o modelo MVC)

-Para a Associação com Facebook:

A partir do Access Token iremos obter o id de utilizador e em seguida armazená-lo na nossa base de dados com recurso ao servidor RMI.

-Para Login com Facebook:

A partir do Access Token iremos obter o id de utilizador da sessão atual e em seguida procurar um utilizador na nossa base de dados com o id correspondente.

-Para partilhar um voto:

A partir do Access Token criamos um pedido (POST), assinado pelo Access token do cliente. (Atenção substituir ' ' por '%20')
RELEMBRAMOS: para segurança do cliente, o seu token não é armazenado.

Universidade de Coimbra

Sistemas Distribuídos
2017/2018

>WEBSOCKETS

Para conseguirmos cumprir os requisitos foi necessário tornar o server endpoint num servidor rmi.
Para isso a class que server de server endpoint foi tornada num RemoteUnicastObject.
O servidor rmi contém assim um set com todas as websockets ativas. Sempre que um tipo de operação específico é realizado pelo rmi é invocado um método específico do server endpoint. Por exemplo quando é chamado uma função de voto no rmi, este informa as websockets que precisam de atualizar os clientes sobre os votos. É então invocado um método do server endpoint que invoca um método do rmi para atualizar os clientes com os novos dados.
Deste modo mantemos os clientes atualizados em tempo real.
Como existem 3 tipos de operações diferentes a websocket tem de saber qual a operação que lhe pertence. Para isso, ao ser feito o handshake entre o cliente e o server, o cliente envia uma string ao server de modo a este saber a sua função (ex. tipo:infoadicional).
Cada websocket apenas executa 1 função.

55	Requisitos Funcionais	0
5	Registar pessoas (estudantes, docentes, ou funcionários)	✓
5	Login protegido com password (acesso a ações e a páginas)	✓
5	Criar eleição (incl. integração com a meta 1)	✓
5	Criar listas de candidatos a uma eleição	✓
5	Listar eleições e consultar detalhes de cada uma delas	✓
5	Adicionar mesas de voto a uma eleição (incl. integração com a meta 1)	✓
5	Alterar propriedades de uma eleição	✓
5	Votar (incl. integração com a meta 1)	✓
5	Saber em que local votou cada eleitor	✓
5	Eleição termina corretamente na data, hora e minuto marcados	✓
5	Consultar resultados detalhados de eleições passadas	✓
0	Grupos de 3: Alterar dados pessoais (-5)	✓
0	Grupos de 3: Gerir membros de cada mesa de voto (-5)	✓
0	Grupos de 3: Considerar eleições de departamento e faculdade (-5)	x
15	WebSockets	0
5	Página de uma eleição mostra eleitores em tempo real	✓
5	Páginas de administração mostram o estado das mesas de voto (da meta 1)	✓
5	Listar utilizadores online	✓
20	REST	0
5	Associar conta existente ao Facebook	✓
5	Login com o Facebook	✓
5	Partilha da página de uma eleição no Facebook sendo o post atualizado no fim	x
5	Post no Facebook de um eleitor assim que vote numa eleição	✓
0	Grupos de 3: deve ser possível desassociar um utilizador do facebook (-5)	0
10	Relatório	0
2	Arquitetura do projecto Web detalhadamente descrita	✓
2	Integração do Struts com o servidor RMI	✓
2	Integração de WebSockets com Struts e RMI	✓
2	Integração de APIs REST no projecto	✓
2	Testes de software (tabela: descrição e pass/fail de cada teste)	✓
	Extra (até 5 pontos)	0
	Utilização de HTTPS (4 pts)	
	Utilização em smartphone ou tablet (2pts)	
	Outros (a propor pelos alunos)	
	Pontos Obrigatórios	0
	Pontualidade (-10)	
	O projeto corre distribuído por várias máquinas (-5)	
	Código HTML e Java estão separados (-5)	
	A aplicação não apresenta erros/exceções/avarias (-5)	
	Código legível e bem comentado (-5)	