Meta 1

eVoting

Voto Eletrónico na UC

FACULDADE DE CIÊNCIAS E TECNOLOGIA

UNIVERSIDADE D COIMBRA

Sistemas Distribuídos

Departamento de Engenharia Informática

2020/2021 - 2º Semestre

Nome	Nº	Email	PL
Filipa Capela	2018297335	uc2018297335@student.uc.pt	PL4
Rodrigo Sobral	2018298209	uc2018298209@student.uc.pt	PL1

Divisão de Tarefas

Durante o processo de divisão de tarefas, decidimos seguir parcialmente a proposta 1 sugerida no enunciado, desta forma as responsabilidades atribuídas foram:

- Rodrigo: Servidores e Clientes RMI e Consola de Administrador
- Filipa: Servidores e Clientes Multicast

Podemos assim dividir o trabalho em duas divisões: a divisão RMI e a divisão MultiCast. Nos próximos tópicos deste relatório serão abordadas as funcionalidades e implementações em cada divisão e, por fim, como se relacionam entre si.

Divisão RMI - Rodrigo

- Arranque dos servidores RMI:

Foi tido em conta que podemos ter os dois servidores RMI em máquinas diferentes (mas na mesma rede), pelo que a primeira coisa que deve ser feita ao arrancar o servidor RMI é indicar o endereço IP da máquina à qual nos queremos ligar, caso seja o servidor secundário a ser arrancado, se for o primário basta clicar *Enter* e ele fará automaticamente um *binding* com o formato *rmi://192.168.1.4:1099/rmiconnection1*, por exemplo. Será, neste caso, este endereço (192.168.1.4), que o servidor secundário (presente noutra máquina) deve inserir para se ligar e consequentemente fazer os *heartbeat pings* (através de *callbacks* RMI).

É proibida a associação de mais de um servidor, mais associações serão recusadas por ambos os servidores (primário e secundário).

- **Dados dos Servidores RMI:**

Não esquecendo que os servidores RMI são responsáveis por armazenar toda a informação do sistema, é aqui que reside toda a responsabilidade de sincronização e integridade dos dados. Em baixo estão apresentados as estruturas utilizadas divididas por tópicos:

Sempre que é feito qualquer tipo de alteração válida nos dados é simultâneamente, se possível, enviada para o servidor secundário (para assegurar a integridade dos dados caso o servidor principal *crashe*) e escrita em ficheiros locais (para quando o servidor principal for reiniciado, poder recuperar toda a informação registada até então). Portanto, sempre que há um cliente/administrador a mais/menos, uma alteração de eleições, ou mesmo o aparecimento dum novo servidor, todos os dados registados são-lhe enviados.

- Dados de Conexão

```
//    Servidor Principal
RMIServer server;
//    Ponte de acesso ao Servidor Secundário
RMIServer_I pinger;
//    Identificadores de Binding dos servidores
String rmiregistry1, rmiregistry2;
//    Porto da Conexão
int port;
//    Flag identificadora do Servidor Principal
boolean main_server;
//    Endereços RMI completos do Servidor Local e Remoto
String my_rmi_ip, remoted_server_ip;
```

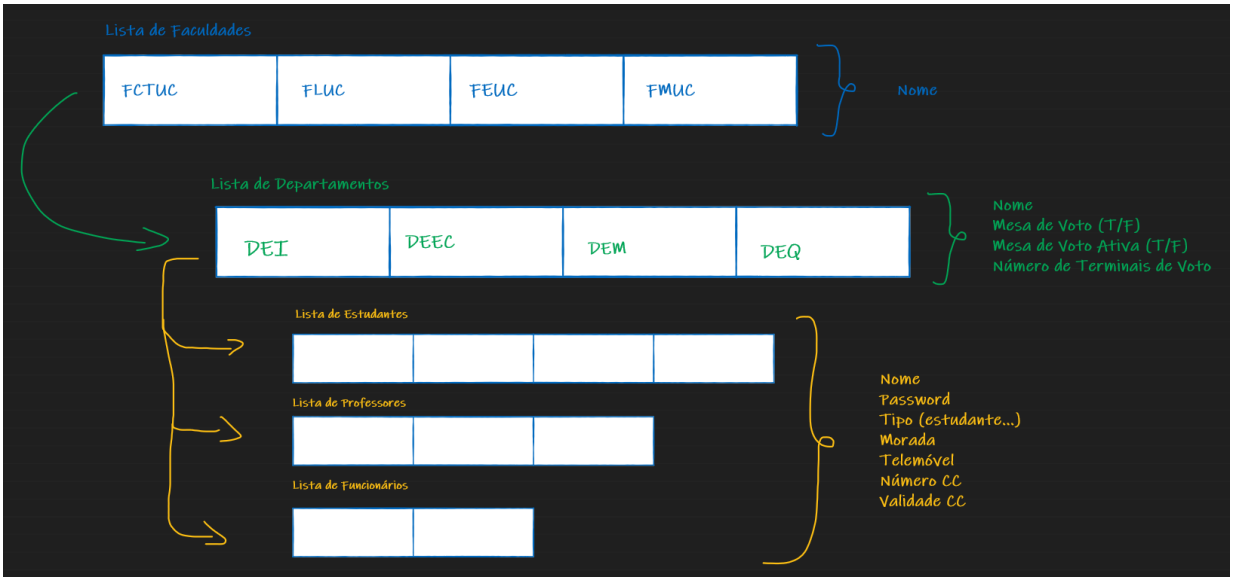
- Dados dos Clientes Conectados:

```
//    Lista das Mesas de Voto e Administradores conectados
ArrayList<RMIClient_I> clients_list, admins_list;
//    Lista dos Departamentos (associados às Mesas de Voto Ativas)
ArrayList<String> associated_deps_list;
```

- Dados Manipulados pelo Administrador:

```
//    Lista de Faculdades->Departamentos->Usuários
ArrayList<College> colleges;
//    Lista de eleições não começadas, começadas e acabadas,
respetivamente
ArrayList<Election> unstarted_elections, running_elections,
finished_elections;
```

- Estrutura dos Dados dos Usuários:



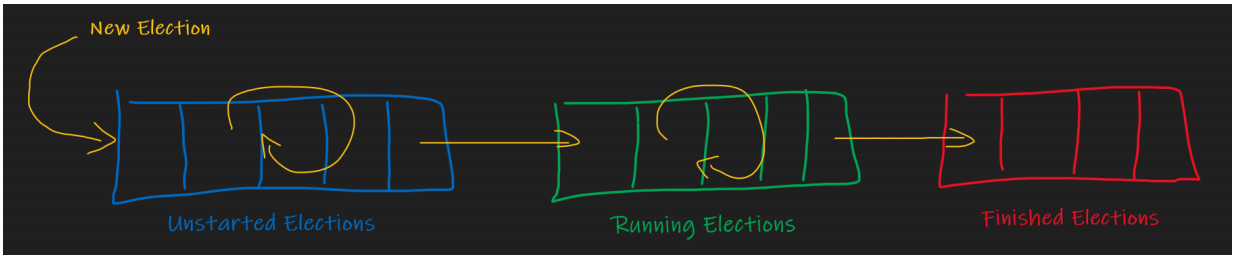
- **Threads dos Servidores RMI:**

Durante o seu funcionamento, os servidores executam paralelamente 3 threads que servem para coisas distintas, passemos a listá-las:

- ServersState **(FAILOVER)**: Caso haja um servidor associado, ambos criam uma relação simbiótica, na qual enviam pings mutuamente de forma a detetarem a ausência um do outro. O principal, depois de 5 pings falhados ao secundário, desassocia-o. Se for o secundário a detetar a ausência permanente do principal, ele torna-se o principal e remove a sua associação ao anterior servidor principal.

- ClientsState: Percorre constantemente todos os clientes RMI associados a ele (administradores e mesas de voto) e, assim como aos servidores associados, faz-lhes um *ping* para saber se ainda se mantêm conectados. Desta forma, quando um cliente encerra o seu processo inesperadamente, o servidor RMI tem esse conhecimento e remove-o.

- ElectionsState: Percorre as eleições começadas e não começadas e, tendo em conta a hora atual, verifica e gerencia as eleições assim que uma eleição acaba e começa, respetivamente. Portanto, quando, por exemplo, uma eleição acaba, ela é removida da lista *running_elections* e passa a fazer parte da lista *finished_elections*. Desta forma conseguimos ter todas as eleições organizadas e não há espaço para conflitos quando elas são requisitadas pelos administradores, uma vez que são dados independentes.



- **Arranque dos Clientes RMI**

Os clientes RMI (por *default* serão as mesas de voto), passam sensivelmente pelo mesmo processo que aos servidores, no que toca ao seu arranque. É primeiramente pedido o endereço do Servidor Principal e a conexão é feita, porém, o cliente não está subscrito ao servidor, o que significa que, apesar de o cliente ter

acesso aos métodos do servidor, o servidor ainda não o reconhece como cliente. Quando a subscrição é feita o cliente passa não só a fazer parte do servidor o principal como do secundário (se existir) para que, caso o principal deixe de funcionar, o cliente possa encaminhar a informação para o secundário.

- **Arranque da Consola de Administrador**

A Consola de Administrador nada mais é que um cliente, por isso trata-se duma classe filha dos clientes RMI, porém, com funcionalidades adicionais que estão, quando solicitado, a obter e adicionar dos/nos servidores dados das eleições, departamentos, entre outros. É aqui que a maioria dos métodos remotos de *callback* são utilizados, uma vez que o administrador faz inserções de dados nos servidores e precisa de receber dele uma confirmação de como os dados foram devidamente registados, além disso, faz também requisições de dados ao servidor sempre que o administrador precisa de ter uma lista de eleições, usuários, departamentos, etc, à sua disposição.

Divisão MultiCast (Filipa)

A comunicação por multicast foi dividida em duas fases. A primeira fase é a procura dos terminais de voto que estão disponíveis para votação e a atribuição dos mesmos para um eleitor. A segunda fase, é a passagem do voto para o servidor multicast (mesa de voto) para posteriormente transmitir a informação para o RMI Server, para este poder guardar a informação que achar necessária. Sendo assim, foram criados dois grupos multicast distintos para fazerem as tarefas acima descritas. Foram desenvolvidas as seguintes classes: *MCServer*, *SecMultServer*, *MCClient*, *SecMultClient*, *Client_Received*, *HandlerMessage*, *MCServerData*

- **Grupo Multicast 1:**

  Para este grupo, no lado do Multicast Server, foi utilizada a classe MCServer, para a criação da thread designada por *mesa_voto*, e do lado do Multicast Client foram utilizadas duas Classes (*SecMultClient*, *Client_Received*) para a criação da thread *cliente2* e *cliente2_received*.

  A thread *mesa_voto* tem função tanto de sender como de receiver, enquanto que a thread *cliente2* tem função de sender e a thread *cliente2_received*, tem função de receiver. Esta distinção feita no multicast Client, deve-se ao facto de haver garantia de que quando é aberto um terminal de voto antes da mesa de voto, a mensagem seja enviada, sem haver perdas.

  Quando uma mesa de voto/terminal de voto é aberta, é pedido o ip e a porta do grupo multicast a que se querem conectar, para que consigam

comunicar entre si. Seguidamente o multicast client, envia uma mensagem com a sua identificação**(1)**, e fica à espera da mensagem de resposta**(2)**. Quando é encontrado um terminal de voto livre, este recebe uma mensagem a avisar que já lhe foi atribuído um eleitor**(3)**, incluindo o cc do mesmo, passando seguidamente para a fase de votação (Grupo multicast 2).

- **Grupo Multicast 2**

Para este grupo, no lado do Multicast Server, foi utilizada a classe *SecMultClient*, para a criação da thread designada por *mesa_voto2*, e do lado do Multicast Client foi utilizada a classe *MCClient* para a criação da thread cliente. Ambas as thread funcionam como sender e como receiver. Quando a um eleitor é atribuído a um terminal de voto, passa a existir uma troca de mensagens, que permite verificar se um eleitor fez a autenticação corretamente.Se sim, é lhe enviada a lista de candidaturas em que este pode votar, e posteriormente, é enviado o voto para o Multicast Server e o terminal de voto encerra automaticamente.

- **Protocolo Multicast (mensagens trocadas)**

Mensagens Trocadas no Grupo Multicast 1:

- ➢ *type|envia_id;id|5* **(1)**
- ➢ *type|mult;ip;porta;depar;id|5* ex: *type|mult;224.0.0.1;10;DEI;id|5* **(2)**
  Nota: esta mensagem, serve de verificação de que o multicast server está a comunicar com o multicast cliente, mas também, envia o ip e a porta que são gerados na mesa de voto para a comunicação multicast 2
- ➢ *type|connected;cc|85412048512;id|4* **(3)**

Mensagens Trocadas no Grupo Multicast 2:

- ➢ *type|login;username|ola;password|adeus;id|5*
- ➢ *type|login;status|sucessed;id|4*
- ➢ *type|login;status|unsucessed;id|4*
- ➢ *type|ask;id|852;*
- ➢ *type|listacandidaturas;lista1;lista2;lista3;...;id|6*
- ➢ *type|resultado;opcaovoto|1;cc|123654789;id|4*

Observação:

No Multicast Client, a thread *cliente2 e a thread cliente2_received, como elas pertencem ao mesmo grupo multicast, quando há um send de pacotes para o multicast server, a thread cliente2_received, também recebe os pacotes, no entanto, as mensagens duplicadas são ignoradas.*

RMI + MultiCast

A ativação de uma mesa de voto é o primeiro passo para o processo de voto. Portanto, ao correr um servidor *MultiCast* (que também assume a função de cliente RMI), conecta-se ao servidor principal (como referido anteriormente, através dum pedido de endereço ao utilizador) para que, desta forma, faça requisições ao servidor RMI (por *callback*) dos departamentos disponíveis para abertura de mesas de voto, das eleições disponíveis para o departamento selecionado (é nesta altura que a mesa de voto se subscreve efetivamente ao servidor RMI e este, por sua vez, a regista como cliente), e das autorizações / autenticações dos eleitores.

NOTA: Durante o processo de autorização e autenticação, é de ressaltar que nunca é feita a transferência de dados sensíveis dos eleitores por parte do servidor RMI, apenas confirmações!

Não só é feita uma requisição de dados como também uma reposição de dados, isto é, ao efetuar uma votação, a eleição, com resultados atualizados, é enviada e atualizada em tempo real no servidor RMI, para que, por sua vez, os administradores possam acompanhar o estado das eleições em simultâneo.

Fase de Testagem

Testes	Resultados
Validação dos *Inputs* Gerais	✅
Verificação de CCs, Departamentos, Faculdades ou Eleições já registadas(os)	✅
Verificação de Datas, números de telemóvel, validades do CC, entre outros	✅
Registo de Usuários	✅
Registo de Eleições	✅
Registo de Candidaturas	✅
Registo de Mesas de Voto	✅
Edição de Eleições (apenas as não começadas)	✅
Remoção de Mesas de Voto	✅
Consultas em Tempo Real	✅

Consulta de Eleições Atuais	✅
Consulta de Eleições Acabadas	✅
Consulta do estado das Mesas de Voto	✅
Voto	✅
Comunicação Multicast	✅
Votações Consecutivas no mesmo Terminal	✅
Pinging entre Servidores	✅
Pinging entre máquinas diferentes	❌/ ✅ (funciona num só sentido)
Substituição do Servidor Principal pelo Secundário	✅
Começo e Término das Eleições na hora marcada	✅
Autorização do Eleitor ao Voto (por cc)	✅
Autenticação do Eleitor (username,password)	✅
Verificação de votação prévia pelo mesmo Eleitor	✅
Verificação caso o Departamento do Eleitor não esteja incluído nas restrições da Eleição	✅
Autenticação do Eleitor (por username e password)	✅
Recuperação do Servidor Principal	✅
Ligação dum Cliente a um ex Servidor Secundário	❌
Falhas nos servidores são invisíveis para os Clientes	❌