

DropMusic

Arquivo de música com partilha de ficheiro

Relatório meta 1

Sistemas Distribuídos | 17 de dezembro 2018

Indice

[**Introdução** 2](#_Toc532844901)

[**Arquitetura de Software** 2](#_Toc532844902)

[**Integração do Struts2 com servidor RMI** 3](#_Toc532844903)

[**Integração de websockets com o servidor RMI** 3](#_Toc532844904)

[As websockets são usadas para receber ou enviar informação em substituição do Callback da meta 1. Ou seja, a informação é recebida em real time, caso o utilizador não esteja online para as receber, as informações, mais precisamente, as notificações são são guardadas num ficheiro obj onde as informações são separadas por “;” e a palavra incial será ou “login” ou “critica” ou “privilégios. 3](#_Toc532844905)

[**rest** 5](#_Toc532844906)

[**Testes de software** 6](#_Toc532844907)

## **Introdução**

Este projeto tem como objetivo criar um sistema de gestão de partilha de músicas. O sistema tem todas as informações relevantes sobre as músicas, tais como autores, álbuns, grupos musicais, compositores ou outros que se considere importantes. O sistema tem utilizadores que acedem às funcionalidades com a aplicação DropMusic. Alguns utilizadores são também editores que podes acrescentar e modificar informações.

## **Arquitetura de Software**

Este projeto tem é conseguido através da junção de duas versões, versão SD\_1819\_projeto\_versao01 que foi apresentado para a primeira meta e versão SD\_1819\_projeto\_meta2 onde tem a junção da meta 1 com a meta 2 onde constam algumas otimizações do servidor RMI e Multicast em relação à meta 1.

A aplicação DropMusic consiste em três programas: servidor RMI, servidor Multicast e cliente web.

* O servidor RMI trata dos pedidos do cliente e traduz o pedido para o servidor Multicaste de acordo com a sua API. O mesmo é aplicado para a resposta do pedido que desta vez é traduzida para o cliente web ou RMI.
* O servidor Multicast guarda todas as informações precisas para o conteúdo do sistema como informações de artista, de músicas, de álbuns, clientes registados e histórico de alterações que possam ter ocorrido de acordo com o estatuto de cada utilizador.
* O cliente web é providenciado com uma interface web intuitiva para a apresentação do conteúdo desejado ou possíveis movimentos dentro da plataforma e DropBox dependendo do seu estatuto.

A aplicação web corre num servidor HTTP(Apache Tomcat), versão 9.0.12 e em grande maioria, o projeto é coordenado por Java versão 1.8.0\_181 com um linguagem do nível 8.

O desenvolvimento web do projeto é dominado por Struts2, JavaServer Pages e JavaBeans.

## **Integração do Struts2 com servidor RMI**

Struts2 é usado para o desenvolvimento do aplicativo na Web. Está construído por cima do Java Servlet API e adota o modelo MVC.

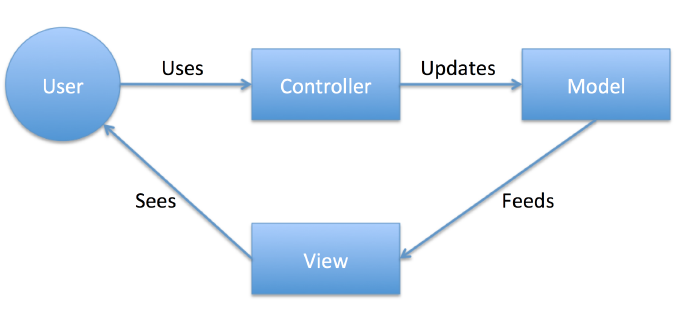


Figura 1, Modelo MVC

Os pedidos do cliente são enviados para o Controller na forma de action, esta action chama a respetiva class cujo nome acaba em “action”. O pedido é processado e para o view, jsp, é retornado um “Action Forward”, onde lhe diz qual vai ser a paginá a ser enviada para o cliente. A informação é passada para uma class cujo o nome acaba em bean , dependendo do resultado daquela action o cliente é direcionado para uma página e começa tudo de novo na view, ou seja, no jsp.

A integração com o RMI é feita no bean, mais especificamente no ficheiro “Login\_bean”. É neste ficheiro que são feitos os pedidos ao servidor cuja instância é RMIServerInterface, no localhost, porto 7000. Nesta classe é confirmado o login, e são pedidos os conteúdos, adicionados conteúdos ou atualizados. Através do RMIServerInterface são feitos, posteriormente os pedidos e respostas ao multicast.

## **Integração de websockets com o servidor RMI**

As websockets são usadas para receber ou enviar informação em substituição do Callback da meta 1. Ou seja, a informação é recebida em real time, caso o utilizador não esteja online para as receber, as informações, mais precisamente, as notificações são são guardadas num ficheiro obj onde as informações são separadas por “;” e a palavra incial será ou “login” ou “critica” ou “privilégios.

Antes de tudo é preciso haver um handshake inicial para haver uma conexão entre servidor e cliente. Isto é feito imediatamente ao abrir o ficheiro .jsp que contenha o javascript necessário para a sua realização, “menu.jsp”. Assim, quando o menu é carregado faz-se a conexão da localização da pasta ws do projeto.

window.onload = **function** () {  
 connect('ws://' + window.location.host + '/SD\_1819\_projeto\_meta2/ws');  
}

Assim verifica-se caso a janela precise de um “WebSocket” como por exemplo para Chrome ou um “MozWebSocket” como por exemplo para Mozilla Firefox, caso nenhum destes dê, pede-se ao cliente para usar a aplicação noutro browser. Definem-se as funcõe

Na classe WebSocketAnnotation.java define-se como ServerEndpoint, ou seja declara-se que aqui são feitas as as annotations usadas no menu.jsp, assim define-se um caminho mais dinâmico

@ServerEndpoint(value = "/ws")

As anotações são as:

@OnOpen  
public void start(Session session)

Quando a ligação é estabelecida a sessão é passada por argumento e guarda-se a sessão num dos atributos da classe.

@OnClose  
public void end()

Quando a janela fecha, fecham-se os recursos que devem ser fechados.

@OnMessage  
public void receiveMessage(String message)

Recebe a String enviada a partir do cliente e é automaticamente dispultado quando o cliente envia mensagem tal como é feito no sendmessage(). Dependendo do início da mensagem, são manipuladas de maneira diferente.

@OnError  
public void handleError(Throwable t)

São também usados métodos como:

private void sendMessage(String text)

e getters and setters dos atributos da própria classe.

Caso o cliente esteja online a notificação é enviada e posteriormente recebida de imediato caso esteja na view certa:

item.getSession().getBasicRemote().sendText(mensagemNotificacao);

Caso não esteja na página a informação é guardada no formato certo para o tipo de notificação:

Para o caso de ser uma crítica é guardado o cabeçalho “critica”, o nome do album que foi criticado, é passada a pontuação dada para aquele álbum, a pessoa que criticou e a pontuação médio com que o álbum ficou.

"critica:Album " + radioListaAlbuns + " foi criticado, atualizado[" + pontuacaoFeita + ", " + login\_bean.getUsername() + "] com pontuacao media de " + pontuacaoMediaAlbum

Caso seja avisado de que os privilégios foram mudados:

mensagem = "Teus privilegios foram mudados, parabens es editor [ " + sdf.format(cal.getTime()) + " ]" + ", " + login\_bean.getUsername();

mensagem = "Teus privilegios foram mudados, lamento ja nao es editor [ " + sdf.format(cal.getTime()) + " ]" + ", " + login\_bean.getUsername();

Dependendo de cada caso(Remover/Dar privilégios) é passada a notificação demonstrando isso, a data da alteração e o editor que os alterou.

"login;" + username

Caso o utilizador entre na página é passado o username para o WebSocketAnnotation para lhe devolver o histórico de notificações.

Para enviar notificação a mais de um cliente pega-se na lista de clientes online da classe WebSocketAnnotation. Esta lista é guardada de forma estática e é partilhada por qualquer instância.

private static Set<WebSocketAnnotation> *users1* = new CopyOnWriteArraySet<>();

## **rest**

## **Testes de software**

|  |  |
| --- | --- |
| Requisitos Funcionais |  |
| Registar novo utilizador | PASS |
| Login protegido com password | PASS |
| Introduzir artistas, álbuns e músicas | PASS |
| Pesquisar álbuns por artista e por título de álbum | FAIL |
| Consultar detalhes de álbum (incluindo músicas e críticas) | PASS |
| Editar detalhes de álbum (incluindo músicas) | FAIL |
| Escrever crítica sobre um álbum (com pontuação) | PASS |
| Consultar detalhes de artista (e.g., discografia, biografia) | PASS |
| Dar privilégios de editor a um utilizador | PASS |
| Notificação imediata de privilégios de editor (online users) | FAIL |
| Notificação imediata de re-edição de descrição de álbum (online users) | FAIL |
| Entrega posterior de notificações (offline users) | FAIL |
| Upload de ficheiro para associar a uma música existente | FAIL |
| Partilhar um ficheiro musical e permitir o respetivo download | FAIL |
| Tratamento de Exceções |  |
| Avaria de um servidor RMI não tem qualquer efeito nos clientes | PASS |
| Não se perde/duplica músicas se os servidores RMI falharem | PASS |
| O serviço funciona desde que haja um servidor multicast disponível | PASS |
| Avarias temporárias (<30s) dos N servidores são invisíveis para clientes | PASS |
| Pedidos são sempre processados por N>=1 servidores multicast | FAIL |
| Pedidos de leitura são respondidos apenas por um servidor multicast | PASS |
| Failover |  |
| Servidor RMI primário testa periodicamente o secundário | PASS |
| Em caso de avaria longa os clientes RMI ligam ao secundário | PASS |
| Servidor RMI secundário substitui o primário em caso de avaria longa | PASS |
| O failover é invisível para utilizadores (não perdem a sessão) | PASS |
| O servidor original, quando recupera, torna-se secundário | PASS |
| Relatório |  |
| Arquitetura de software detalhadamente descrita | PASS |
| Detalhes do funcionamento do servidor UDP (protocolo multicast, etc.) | FAIL |
| Detalhes do funcionamento do servidor RMI (interface, javadocs, etc.) | PASS |
| Distribuição de tarefas pelos elementos do grupo | PASS |
| Testes de software (tabela com descrição + pass/fail de cada teste) | PASS |
| Extra (até 5 pontos) |  |
| Servidores multicast replicados mantêm os dados sincronizados (5p) | FAIL |
| Balanceamento da carga dos servidores (3p) | FAIL |
| STONITH nos servidores RMI (3p) | FAIL |
| Outros (a propor pelos alunos) | FAIL |
| Pontos Obrigatórios |  |
| Pontualidade (-10) |  |
| O projeto corre distribuído por várias máquinas (-5) |  |
| Configuração não requer recompilação (-5) |  |
| A aplicação não apresenta erros/exceções/avarias (-5) |  |
| Código legível e bem comentado (-5) |  |
| No dia da defesa |  |
| Os estudantes chegam 15 minutos antes para prepararem tudo |  |
| Trazem duas máquinas (e.g., dois portáteis) |  |
| Portátil #1 corre 1 servidor multicast e 1 cliente RMI |  |
| Portátil #2 corre 1 servidor multicast, 2 servidores RMI e 1 cliente RMI |  |
| Há dados de teste (3 utilizadores, 3 álbuns e várias músicas) |  |