

Universidade de Coimbra

Faculdade de Ciências e Tecnologia

**Departamento de Engenharia Informática**

**Projecto N.º 6**

**Programação Avançada em Java**

João Pedro Oliveira Brandão Martins

Rafaela David Fernandes Lourenço

Coimbra, 4 de Julho de 2015

Índice

[1 Introdução 1](#_Toc423733453)

[1.1 Eclipse 1](#_Toc423733454)

[1.2 Maven 1](#_Toc423733455)

[1.3 JBoss - WildFly 1](#_Toc423733456)

[1.4 PostgreSQL 1](#_Toc423733457)

[1.5 GitHub 1](#_Toc423733458)

[2 Camada de Apresentação 2](#_Toc423733459)

[2.1 Tecnologias utilizada 2](#_Toc423733460)

[2.1.1 UI Composion Templates 2](#_Toc423733461)

[2.1.2 JSF 2](#_Toc423733462)

[2.1.3 AJAX 2](#_Toc423733463)

[2.1.4 AUTENTICAÇÃO 2](#_Toc423733464)

[2.1.5 Javascript 2](#_Toc423733465)

[2.2 Estrutura das páginas XHTML 3](#_Toc423733466)

[2.2.1 resources/template/common 3](#_Toc423733467)

[2.2.2 Zona não autenticada 3](#_Toc423733468)

[2.2.3 Zona autenticada 3](#_Toc423733469)

[2.3 Interface no servidor 4](#_Toc423733470)

[2.3.1 dei.uc.pt.ar.paj/AppMusicFilters.java 4](#_Toc423733471)

[2.3.2 pt.uc.dei.aor.paj/MusicFilters.java 4](#_Toc423733472)

[2.3.3 pt.uc.dei.aor.paj/MusicServlet.java 4](#_Toc423733473)

[2.3.4 pt.uc.dei.aor.paj/Playlistinterface.java 4](#_Toc423733474)

[2.3.5 pt.uc.dei.aor.paj/Uploader.java 4](#_Toc423733475)

[2.3.6 pt.uc.dei.aor.paj/UserSession.java 4](#_Toc423733476)

[2.3.7 pt.uc.dei.aor.paj/Userinterface.java 4](#_Toc423733477)

[2.3.8 pt.uc.dei.aor.paj.filter/AuthorizationFilter.java 4](#_Toc423733478)

[2.3.9 pt.uc.dei.aor.paj.filter/LoggedInFilter.java 4](#_Toc423733479)

[2.4 Algumas Funcionalidades do Front-End 4](#_Toc423733480)

[2.4.1 Login e Registo de um novo utilizador 4](#_Toc423733481)

[2.4.2 Account 5](#_Toc423733482)

[2.4.3 Playlist 6](#_Toc423733483)

[2.5 Diagrama UML da camada de apresentação 8](#_Toc423733484)

[2.6 testes 8](#_Toc423733485)

[3 Camada de Negócio 9](#_Toc423733486)

[3.1 Tecnologia Utilizada 9](#_Toc423733487)

[3.2 Data Transfer Objects 9](#_Toc423733488)

[3.3 Diagrama UML da camada de negócio 9](#_Toc423733489)

[3.4 Testes 9](#_Toc423733490)

[4 Camada de Dados 10](#_Toc423733491)

[4.1 Tecnologia Utilizada 10](#_Toc423733492)

[4.2 Diagramas ER 10](#_Toc423733493)

[4.2.1 Diagrama conceptual 10](#_Toc423733494)

[4.2.2 Diagrama Físico 10](#_Toc423733495)

[5 Empacotamento 11](#_Toc423733496)

[5.1 Parent 11](#_Toc423733497)

[5.2 P4-Playlist-WEB 11](#_Toc423733498)

[5.2.1 Pom.xml 11](#_Toc423733499)

[5.2.2 Web.xml 11](#_Toc423733500)

[5.3 P4-Playlist-EJB 12](#_Toc423733501)

[5.4 P4-Playlist-JPA 12](#_Toc423733502)

[5.4.1 Pom.xml 12](#_Toc423733503)

[5.4.2 Persistence.xml 12](#_Toc423733504)

[5.5 P4-Playlist-EAR 12](#_Toc423733505)

[6 Gráfico de Gantt 12](#_Toc423733506)

[7 Conclusões 13](#_Toc423733507)

# Introdução

Este projeto consiste em extender o projecto nº4 de forma a disponibilizar aos utilizadores as letras das músicas presentes nas suas playlists. As letras são obtidas através de web services públicos. Esta aplicação recorre a uma base de dados para guardar letras das músicas editadas pelo utilizador.

Em baixo estão listadas as ferramentas de *software* *opensource* utilizadas neste projeto.

## Eclipse

O *software* de desenvolvimento utilizado foi o Eclipse JEE Luna SR2.

Este foi configurado para aceder diretamente à base de dados por configuração de um Data Source adicionando o driver JDBC “*postgresql-9.4-1201.jdbc41.jar*”.

## Maven

Neste projeto recorreu-se à utilização de *Maven*, como ferramenta de *build* de todo o trabalho. No capítulo 5 será descrito em maior pormenor a estrutura do projeto utilizada.

## JBoss - WildFly

Como web server foi utilizado o Wildfly integrado no Eclipse. Foi configurado no wildfly para acesso directo à Base de Dados por *Data Source* “***p6lyrics***” em conformidade com o ficheiro de configuração “***persistence.xml***” adicionando o driver JDBC comum ao Eclipse “*postgresql-9.4-1201.jdbc41.jar*”. Foi também efetuada uma configuração para efetuar *logging* associado ao package deste projeto.

## PostgreSQL

O software de base de dados utilizado foi PostgreSQL, versão 9.4.1, tendo sido criada a base de dados **p6lyrics** para utilização neste projecto.

## GitHub

Para o desenvolvimento e conceção simultâneos e sincronizados do projeto, foi utilizado o *GitHub*. Este consiste num serviço de *Web Hosting* compartilhado para projetos que usam o controlo de “versionamento” *Git*.

# Camada de Apresentação

A camada de apresentação foi efetuada a pensar num *layout* que fosse bastante intuitivo para o utilizador, sem demasiada informação em cada página, mas com o suficiente para não deixar o utilizador sem saber o que fazer.

Nos passos seguintes vamos descrever as principais funcionalidades na utilização do projecto desenvolvido.

## Tecnologias utilizada

### UI Composion Templates

Esta camada foi desenvolvida segundo um template de CommonLayout, onde existem quarto secções que são comuns a todas as páginas: um CommonHeader, um CommonMenu, um CommonContent e um CommonFooter.

O conteúdo da secção relativa ao *CommonContent*, na figura é a área onde está a tabela com as músicas, vai sendo atualizada de acordo com a página que estamos a consultar.

A utilização deste tipo de template na construção de um website permite que não seja necessário repetir o código dos elementos que são comuns a todas as páginas, sendo apenas escrito na página xhtml correspondente a essa secção. Permite também que a correção e formatação desse código seja efetuada de um modo mais fácil e rápido.

### JSF

Neste projeto utilizamos para a construção de interfaces de utilizador baseadas em componentes para aplicações *web* é feita através do *framework:* *JavaServer* *Faces* (*JSF*). O JSF possui um modelo de programação dirigido a eventos, abstraindo os detalhes da manipulação dos eventos e organização dos componentes, permitindo que nos focássemos somente na lógica da nossa aplicação.

### AJAX

Um dos aspetos fundamentais deste trabalho residiu no recurso à tecnologia *AJAX* (*Asynchronous Javascript and XML*). De forma a tornar as páginas *web* do projeto mais interativas com o utilizador implementou-se o AJAX, como método de uso de solicitações assíncronas de informação. Deste modo, conseguiu-se garantir que o *browser* não carregasse constantemente toda a página, nomeadamente, aquando o clique de botões, resultando todas as interações existentes em chamadas ao servidor.

### AUTENTICAÇÃO

Neste trabalho foi desenvolvido um sistema de proteção recorrendo à tecnologia autenticação pelo container. O Wildfly pode ser configurado no standalone.xml um security domain por forma a ter a função de efetuar o acesso à BD para validar credenciais do utilizador por ***form*** e respectivas ***roles***, por forma a impedir o acesso direto de um utilizador a uma area protegida. Essa protecçao é complementada com configurações no ficheiro web.xml. Neste caso, protegeu-se o diretório ***resources/authorized***. Quando existir um utilizador que não efetuou o “login” com sucesso e tenta aceder à área protegida, é redirecionado para a página *index.xhtml*.

### Javascript

Neste projeto foi utilizado *Javascript residualmente*, incorporado nas paginas xhtml para operações associadas a botões.

## Estrutura das páginas XHTML

Nos pontos seguintes será efetuada uma pequena descrição dos ficheiros do front-end deste projecto.

### resources/template/common

Esta camada foi desenvolvida segundo um template de CommonLayout, onde existem quarto secções que são comuns a todas as páginas: um CommonHeader, um CommonMenu, um CommonContent e um CommonFooter.

### Zona não autenticada

#### index.xhtml

Página de arranque inicial que permite o utilizador efetuar sign-in e sign-up.

### Zona autenticada

#### resources/Authorized

O container ***resources/Authorized*** agrupa as páginas cujo acesso requer autenticação.

##### addMusics.xhtml

Página que permite ver as músicas de todos os utilizadores.

##### addPlaylist.xhtml

Página que permite adicionar playlists.

##### allMusics.xhtml

Página que permite adicionar músicas.

##### changeAccount.xhtml

Página que permite ao utilizador editar os seus dados pessoais e credenciais de acesso.

##### editLyrics.xhtml

Página que permite editar letras de músicas.

##### myMusics.xhtml

Página que permite editar dados das suas músicas.

##### myPlaylist.xhtml

Página que permite gerir as suas playlists.

##### searchMusics.xhtml

Página que permite efectuar pesquisa de músicas.

##### thePlaylist.xhtml

Página que permite ver e apagar musicas da playlist seleccionada.

## Interface no servidor

Para efetuarmos a interface com a camada de negócio, implementamos as classes descritas nos passos seguintes.

### dei.uc.pt.ar.paj/AppMusicFilters.java

Mantém a listagem de filtros de pesquisa que o utilizador está a usar para pesquisar músicas da aplicação.

### pt.uc.dei.aor.paj/MusicFilters.java

Mantém a listagem de filtros de pesquisa que o utilizador está a usar para pesquisar as suas músicas.

### pt.uc.dei.aor.paj/MusicServlet.java

Servlet que associa os ficheiros mp3 uploaded a um url.

### pt.uc.dei.aor.paj/Playlistinterface.java

Classe que faz a interface associada às listagens de playlists de utilizadores com as várias possibilidades de ordenação, listagens de músicas associadas a cada playlist, e as respectivas operações de adicionar, remover, visualizar detalhes, ordenar respectivas músicas.

### pt.uc.dei.aor.paj/Uploader.java

Classe que permite fazer o upload das músicas e a sua edição.

### pt.uc.dei.aor.paj/UserSession.java

Classe que mantém a informação do utilizador ativo.

### pt.uc.dei.aor.paj/Userinterface.java

Classe que permite fazer o registo e login de utilizadores.

### pt.uc.dei.aor.paj.filter/AuthorizationFilter.java

Classe com o filtro responsável pelo redireccionamento da página calculadora para a página login, quando o utilizador tenta aceder à página account/playlist sem que tenha efetuado o login/registo ou cuja sessão tenha espirado.

### pt.uc.dei.aor.paj.filter/LoggedInFilter.java

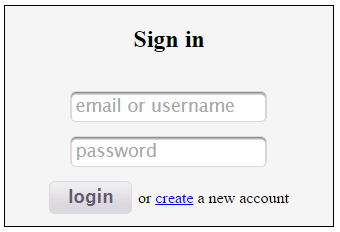
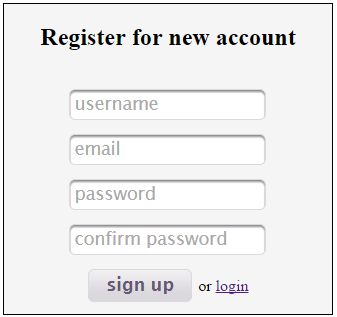
Classe com o filtro responsável pelo redireccionamento da página login para a página playlist, quando o utilizador tenta aceder à página login já estando loggado.

## Algumas Funcionalidades do Front-End

### Login e Registo de um novo utilizador

A página de login é a página inicial de arranque. Para aceder à página da playlist é necessário passar por um processo de autenticação.

Para isso basta ao utilizador introduzir o nome de registo na caixa de texto 1, a password na caixa de texto 2 e efetuar um clique no botão 3.

Quando é efetuado o login ou registo aparece a página principal Account.

No caso do utilizador ainda não ter efetuado um registo, basta carregar no ***create*** para ser direcionado para a página de criação de um novo registo.

Para registo do utilizador introduzir o nome de registo na caixa de texto, a password, confirmar a password e efetuar um clique no ***sign up***.

Ao criar o registo, é efetuado automaticamente o login no registo criado.

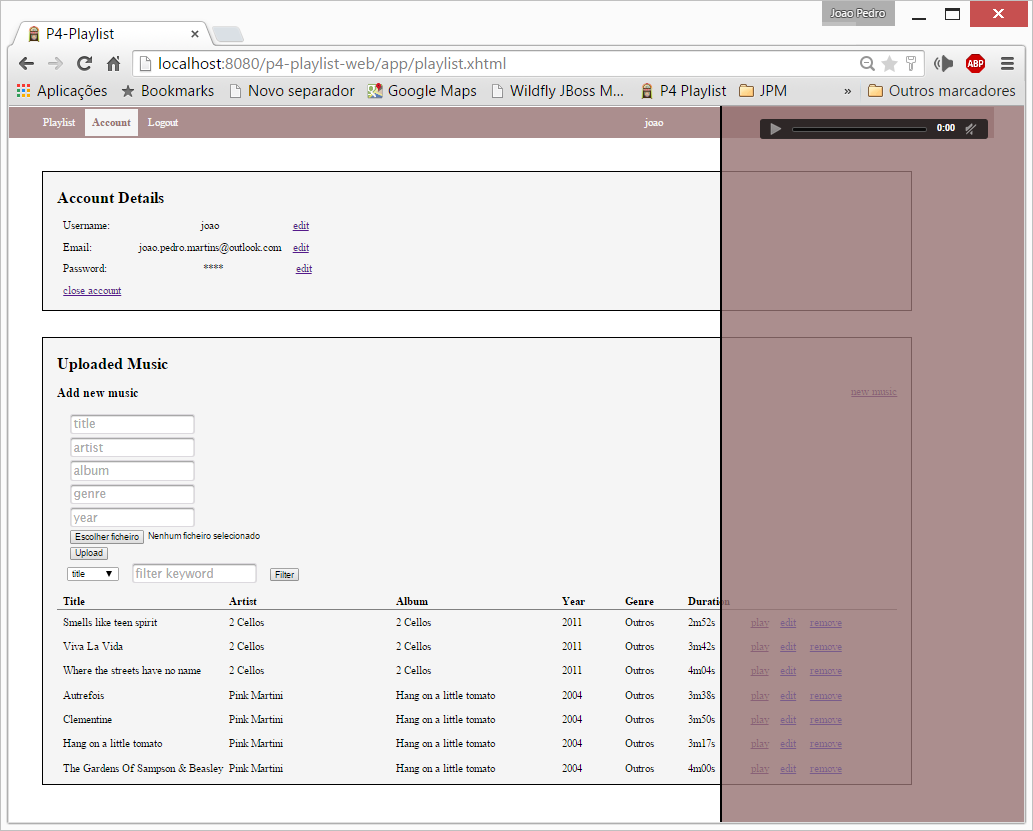
No caso de já ter um registo efetuado, o utilizador pode sempre voltar ao menu anterior carregando no *link* de login.

Todos os campos de password usam caixas de texto ***inputSecret*** para não serem visíveis no ecrã e foram encriptadas com o algoritmo SHA-2 e usando um “*salt*” para não serem visíveis na base de dados.

Existe um filtro para impedir que os utilizadores já com login efetuado façam login de novo ou criem registos novos. Para isso terão de fazer logout.

### Account

A gestão da conta dos utilizadores é efetuada nestes painéis. Os comandos do *player* estão num “drop-down menu” horizontal com abertura e recolha automática quando se passa o cursor por cima.



Por vezes acontece o clicar no play só resultar à 2ª tentativa em algumas situações. Trata-se de um bug conhecido, cujo o nome é JSF spec issue 790.

Foi implementado um filtro para impedir acessos não autorizados a esta página. Caso o utilizador não esteja autenticado, quer por ter efetuado o loggout, ou por a sessão ter sido espirada por time-out, é redirecionado automaticamente para a página de login.

Para adicionar uma música, clicar no *link* *newmusic* e preencher os campos título, artista, álbum, género, ano de publicação, selecionar o ficheiro e *upload*. Como extra, depois do upload implementamos um servlet que mapeia os ficheiros para um url e permite tocar as músicas.

Pode ser efetuada pesquisa de uma música, permitindo a pesquisa por álbum, autor, título, género, ano e por vários campos ao mesmo tempo.

As músicas adicionadas podem ser utilizadas nas playlists de todos os utilizadores.

Quando é removida uma música, esta é removida da base de dados e de todas as playlists de todos os utilizadores (mantendo os ficheiros no servidor).

Ao ser efetuado o ***close account***, todas as músicas do utilizador são removidas da base de dados e de todas as playlists de todos os utilizadores.

### Playlist

A gestão das playlists de um utilizador é efetuada na janela conforme a figura seguinte.

Para criar uma playlist, selecionar *radio botton* para criar, preencher a janela de texto com o nome/designação da playlist pretendida e carregar no botão criar. É efetuada uma pesquisa na base de dados se já existe uma playlist desse utilizador com esse nome. Caso exista é devolvida uma mensagem de erro. Podem existir na base de dados playlists com o mesmo nome desde que sejam de utilizadores diferentes. Casa utilizador só pode visualizar as playlists criadas por si. No entanto pode adicionar músicas existentes que tenham sido adicionadas por qualquer outro utilizador.

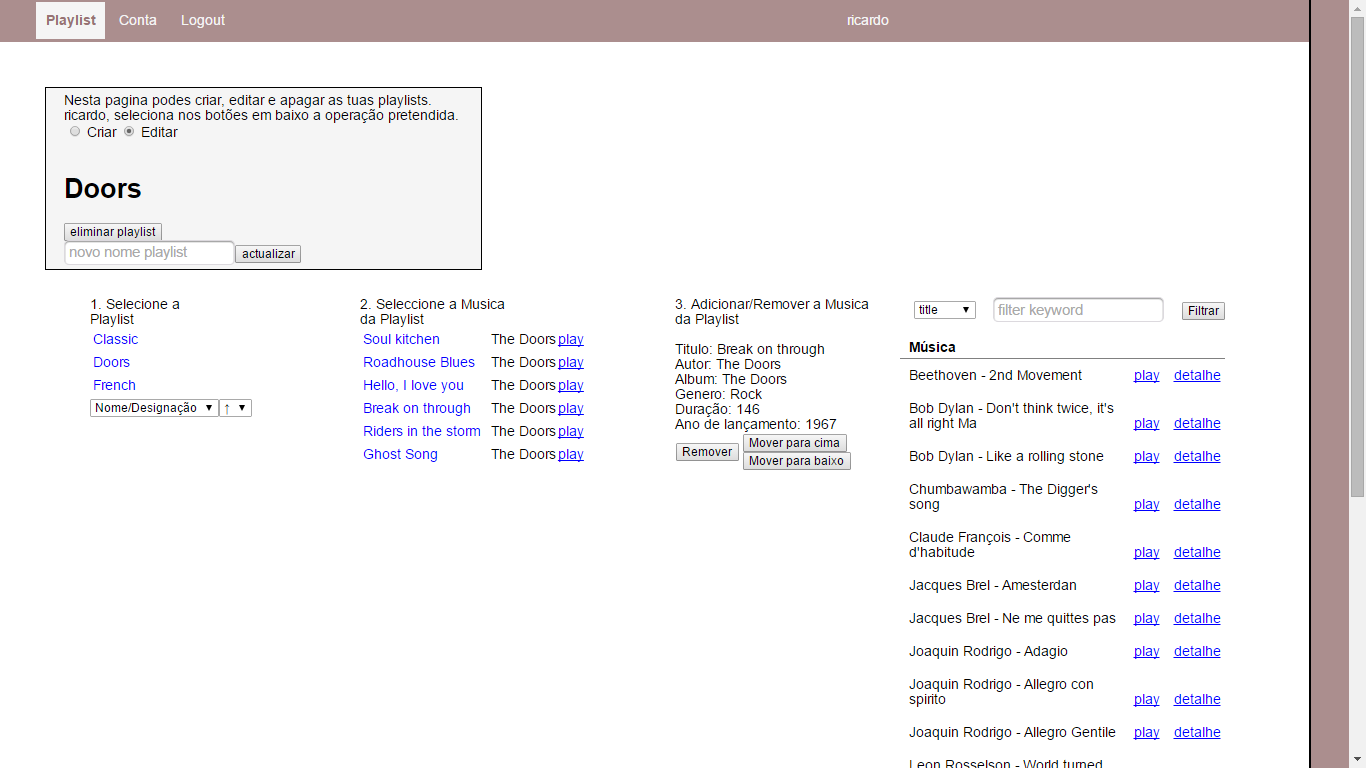
As playlists podem ser ordenadas por nome/designação, Data de criação e tamanho, quer por ordem ascendente quer descendente. A ordenação foi efetuada com recurso a *queries JPQL* à base de dados.

Pode ser visualizado o conteúdo de uma playlist clicando no nome desta. Aparecera na coluna ao lado a lista de músicas da playlist selecionada. Clicando ainda numa música da lista de músicas, aparecem os detalhes da música, e um botão para apagar, um botão mover a para a posição de ordem de reprodução anterior por permuta com a música anterior e de forma análoga, um botão permuta da posição na ordem de reprodução com a música seguinte. Esta ordenação é persistente, ficando guardada na tabela *PlaylistEntry*. Todas as listas acedem a DTO’s obtidos por recurso a *queries JPQL* à base de dados.

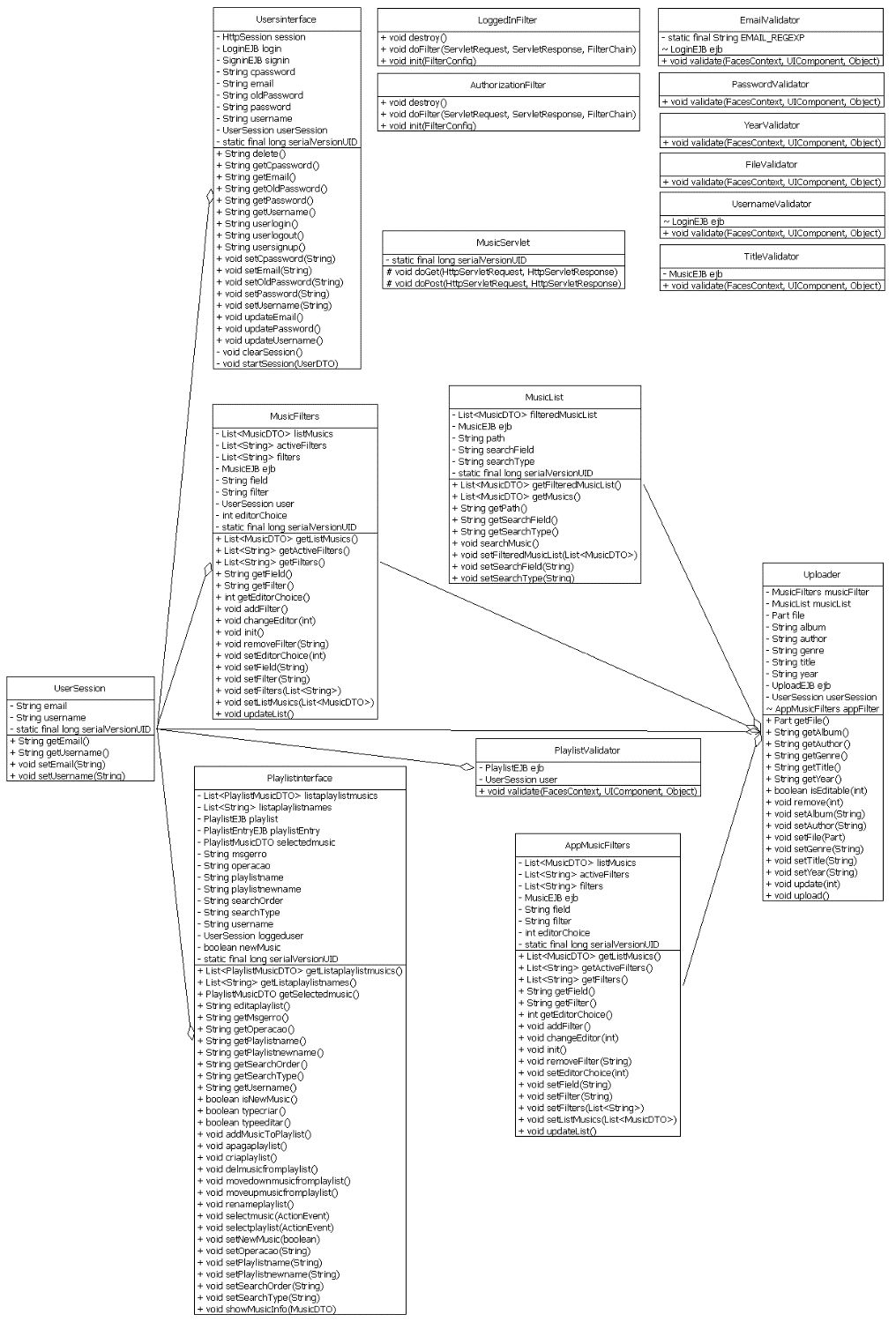
Para adicionar uma música, clicar no detalhe da música pretendida na 4ª coluna, ficando activo um botão adicionar à playlist em baixo dos detalhes da música na 3ª coluna. Carregando no botão adicionar, esta além se ser adicionada à playlist, fica automaticamente selecionada para poder ser ajustada a posição de reprodução na playlist pelos botões mover para cima e para baixo.

Para facilitar a pesquiza se músicas a adicionar às playlists, os critérios de pesquisa de músicas descritas no passo anterior estão também disponíveis nesta *view* (4ª coluna).

Quer ao nível de músicas existentes nas playlists do utilizador, quer ao nível de músicas existentes na aplicação de todos os utilizadores, existe um *link* para o *play* associado a cada música.



## Diagrama UML da camada de apresentação



## testes

Este módulo inclui um conjunto de testes unitários que visam abordar de forma qualitativa o correto funcionamento da calculadora. Para implementar os devidos testes recorreu-se a JUnit e a uma framework de Mocking, neste caso o Mockito.

Foram realizados testes ao nível de validação de username, endereço de email, ano de publicação, e upload de ficheiro mp3 os quais podem ser observados nas Classes "Validators", para prevenir a introdução de inputs indesejados e/ou perigosos na aplicação por parte dos utilizadores.

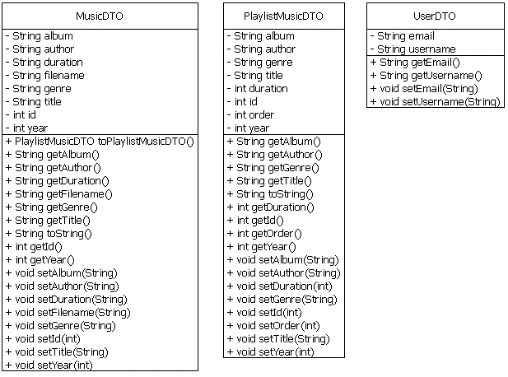
# Camada de Negócio

## Tecnologia Utilizada

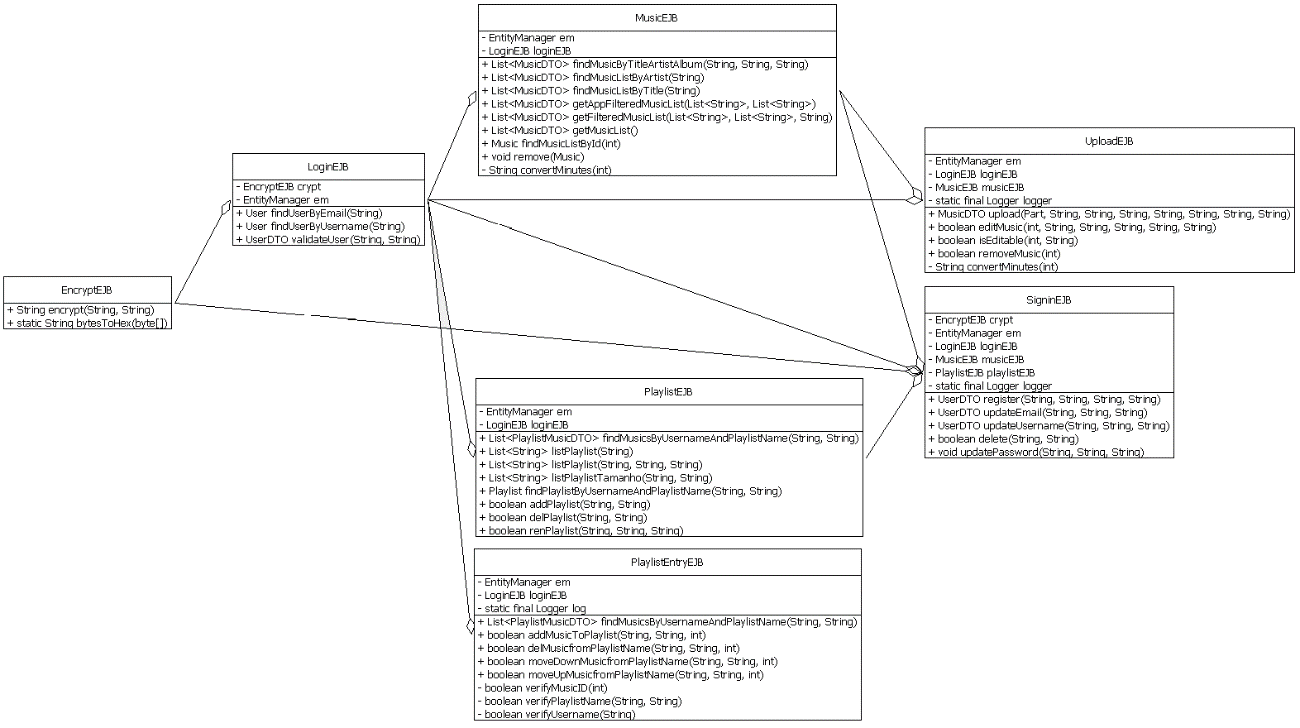
Na camada de negócio a tecnologia utilizada foi EJB (Enterprise Java Beans), consistindo em componentes que são executados nos servidores de aplicação e possuem como principais objetivos, fornecer facilidade e produtividade no desenvolvimento de componentes distribuídos, transacionados, seguros e portáteis.

## Data Transfer Objects

Foram criados 3 DTO’s (*Data transfer objects*) para facilitar a transferência de objetos para a camada de apresentação.



## Diagrama UML da camada de negócio



## Testes

Este trabalho inclui um conjunto de testes unitários que visam abordar de forma qualitativa o correto funcionamento da calculadora. Para implementar os devidos testes recorreu-se a JUnit e a uma framework de Mocking, neste caso o Mockito.

Foram realizados testes a lógica de negócio e a respectiva verificação do acesso correcto por parte da camada Service à camada JPA/Base de dados ao nível das operações de login e signup de utilizadores com respectiva validação de passwords encriptadas, upload de músicas, edição de playlist, etc…

# Camada de Dados

No desenho da base de dados, optamos por um desenho com quatro tabelas. Uma com a informação dos utilizadores. Para cada registo desta tabela podem existir 0 ou mais musicas associadas a cada utilizador. De igual modo podem existir 0 ou mais playlists para cada utilizador. Para cada playlist podem existir 0 ou mais registos para cada playlist, relativo a 0 ou mais músicas da tabela músicas. A tabela PlaylistEntry possibilita um número de ordem sequencial de reprodução de uma música com um determinado music\_ID pertencentes a qualquer utilizador e uma determinada playlist\_ID pertencentes ao utilizador. Esta última tabela garante a persistência da ordem sequencial de reprodução de uma música na playlist.

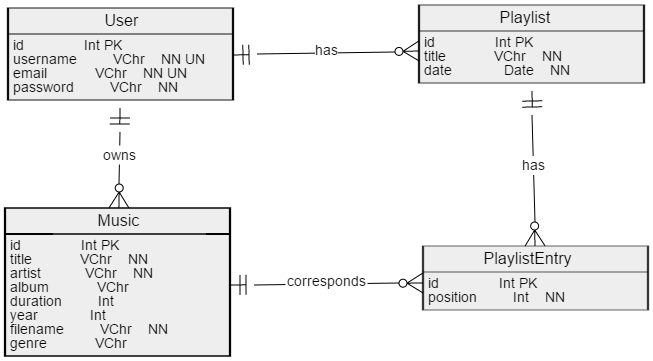
Para garantir integridade na base de dados, tivemos o cuidado de gerar automaticamente todas as *primary keys* das tabelas e além disso tivemos o cuidado de proteger na camada de negócio na verificação do nome de utilizador que efetuou o login e de verificar sempre a playlist por nome/designação em todas as operações de leitura, adição ou remoção de registos da base de dados, quer ao nível de playlist quer ao nível de conteúdos das playlists.

## Tecnologia Utilizada

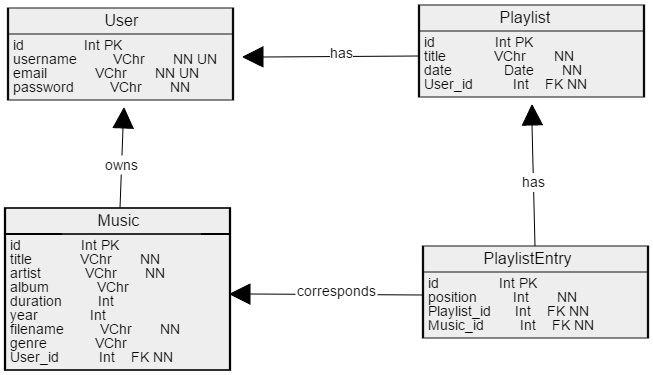
Na camada de dados a tecnologia utilizada foi JPA (*Java Persistence API*), que consiste numa API padrão do Java para a persistência de dados que o utiliza o modelo relacional. Esta API trás uma vantagem de possibilitar uma integração com a base de dados ao nível de entidades sem sequer ser necessário escrever uma linha de SQL, o que liberta o programador para a camada de negócio.

## Diagramas ER

### Diagrama conceptual



### Diagrama Físico



# Empacotamento

O *Maven* recorre a um arquivo *XML* (*POM* – *Project Object Model*) para descrever todo o projecto de *software* em causa, sendo construídas as suas dependências sobre módulos e componentes externos, a ordem de compilação, directórios e *plug-ins* que sejam necessários. Para além de realizar os propósitos de compilação de código e seu empacotamento, o *Maven* descarrega bibliotecas Java e seus plug-ins dinamicamente de um ou mais repositórios, como o “*Maven 2 Central Repository”*, e armazena-os numa área de *cache* local. A construção do *Maven* assenta assim numa arquitectura baseada em *plugin*, a qual permite que ele faça uso de qualquer aplicação controlável através da entrada padrão.

Este projecto consiste em 4 sub-projectos.

* P4-Playlist-WEB consiste na camada de apresentação.
* P4-Playlist-EJB consiste na camada de negócio.
* P4-Playlist-JPA consiste na camada de dados.

P4-Playlist-EAR consiste na junção dos 3 anteriores.

## Parent

O *POM* (*Project Object Model*) é a peça fundamental de um projeto como este, do tipo *Maven*, pois possui as informações básicas de um projeto, bem como as diretivas de como o artefacto final deste projecto deve ser construído. O ficheiro “pom.xml” deste projeto é constituído pelo artefacto: “P4-JoaoMartins-RicardoRodrigues” e possui diversas dependências tais como as necessárias para as tarefas comuns nos sub-projetos.

## P4-Playlist-WEB

### Pom.xml

As principais dependências são JAVA EE, JAVAX, PRIMEFACES, JUNIT, HAMCREST, JFACES e SERVLET.

### Web.xml

O ficheiro “web.xml” consiste num descritor de implantação que é utilizado como arquivo de configuração do artefacto do projecto.

#### Welcome file list

No ficheiro “web.xml” é descriminado o “*welcome-file-list*” que possui indicação da página de acesso à nossa aplicação:

<welcome-file-list>

<welcome-file>index.xhtml</welcome-file>

</welcome-file-list>

#### Filter

Neste projecto o ficheiro “web.xml” possui também informação relativa ao filtro implementado:

<filter>

<filter-name>LoggedInFilter</filter-name>

<filter-class>pt.uc.dei.aor.paj.filter.LoggedInFilter</filter-class>

</filter>

Esta informação indica-nos o nome do filtro (“LoggedInFilter”) e da sua classe (“pt.uc.dei.aor.paj.LoggedInFilter”).

## P4-Playlist-EJB

As principais dependências são o sub-projecto p4-playlist-jpa, JAVAX, JUNIT, HAMCREST, MOCKITO, SLF4J e com.googlecode.soundlibs.

## P4-Playlist-JPA

### Pom.xml

As dependências são JAVAX e HIBERNATE.

### Persistence.xml

O datasource responsável pela ligação à base de dados é:

<class>java:jboss/datasources/p4playlist.PostgreSQL<class>

As classes associadas à entity manager são:

<class>pt.uc.dei.aor.paj.User</class>

<class>pt.uc.dei.aor.paj.Playlist</class>

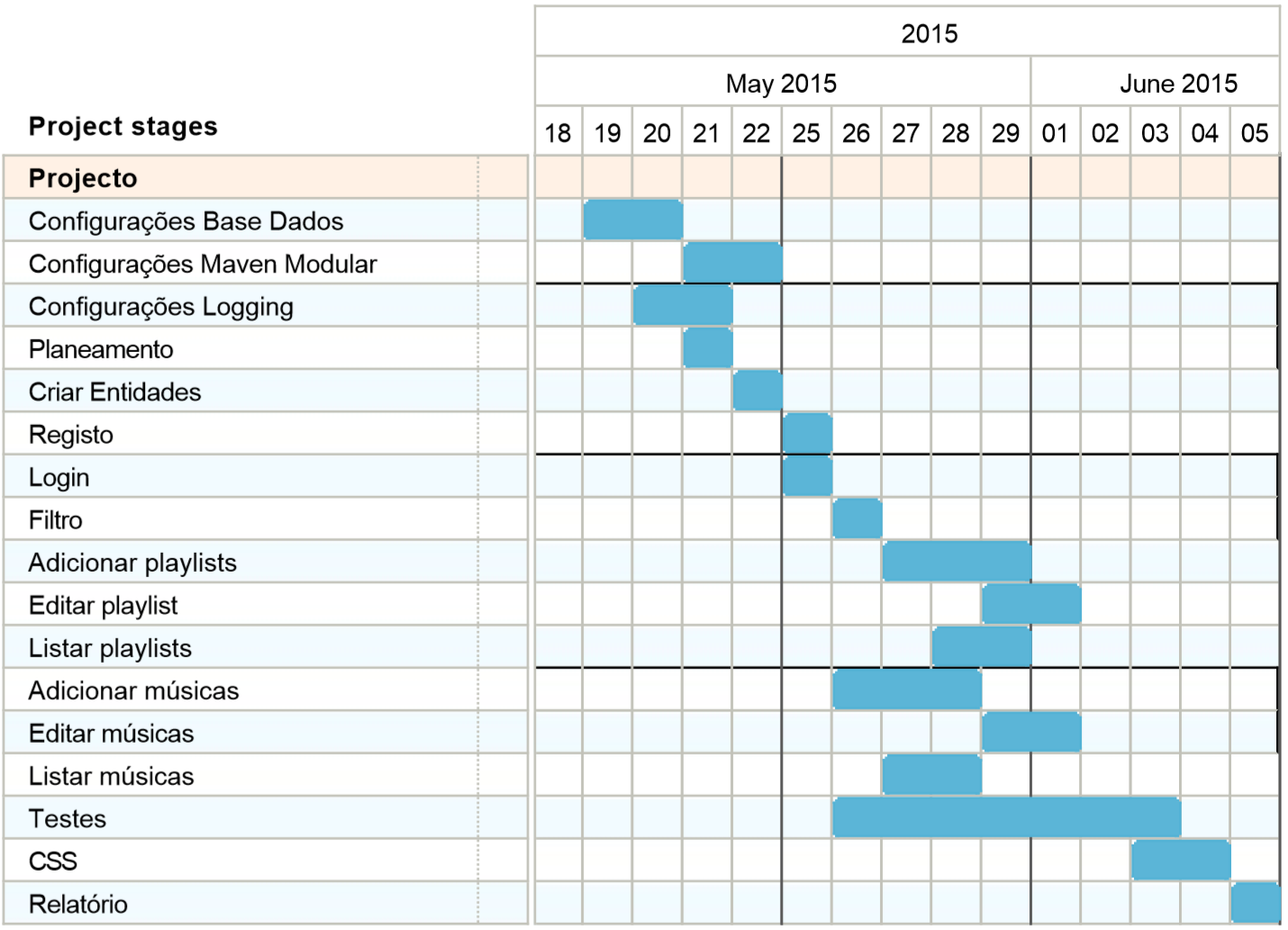
<class>pt.uc.dei.aor.paj.PlaylistEntry</class>

<class>pt.uc.dei.aor.paj.Music</class>

## P4-Playlist-EAR

Este trabalho inclui um conjunto de testes unitários que visam abordar de forma qualitativa o

# Gráfico de Gantt



# Conclusões

Com este trabalho comprovamos a vantagem na utilização de persistência em Bases de Dados.