Programação Avançada em Java

Relatório Projecto nº 4

Java Persistence (API) & Java Persistence Query Language

Gestor de *Playlists* na Web - GetPlayWeb

Grupo7:

2014237436 - Pedro Rodrigues

2014237447 - Rafaela Lourenço

Conteúdo

Introdução 3
Camada de apresentação 4
Camada de negócio 6
Camada de dados 7
Empacotamento 8
Conclusão 9
Anexo 10

Introdução

O objetivo deste projecto era a construção de um *website* capaz de fazer a gestão de *Playlists* de vários utilizadores. Para isso era necessário guardar a informação sobre utilizadores, *playlists* e músicas numa base de dados, neste caso recorrendo ao *PostgreSQL*. O programa foi todo escrito em *JavaEE* recorrendo ao *Wildfly* para se fazer o *deploy* da página *web*. Havia vários requisitos a cumprir na aplicação, e portanto foi necessário fazer uma correcta abordagem ao problema, pensando na arquitectura antes de começar propriamente a trabalhar. Foi feito o diagrama de classes inicial, o diagrama *ER* e também pensados os tempos necessários à execução de cada tarefa, recorrendo a um diagrama de *Gantt*.

Rapidamente percebemos que havia três partes muito importantes no projecto, e era sobre essas que devíamos concentrar os nossos esforços: a camada de apresentação, o nosso negócio (código java clássico) e o acesso à base de dados. Dominando bem estas três componentes em princípio teríamos o trabalho facilitado, e a ideia do grupo foi sempre de ambos estarmos por dentro de todas as partes do trabalho, até para nos podermos ajudar quando surgissem dúvidas/problemas.

O trabalho era bem direccionado para aquilo que vamos fazer de futuro, e portanto considerámos ambos ser da maior importância compreender bem a estrutura e adquirir os conhecimentos necessários para estarmos à vontade em futuros projectos com a mesma arquitectura.

Neste relatório focam-se os principais pontos do projecto: a camada de apresentação, a camada de negócio, a camada de dados, a arquitectura do programa e uma breve conclusão.

Camada de apresentação

A camada de apresentação foi feita sempre 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.

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*.

Ficando as páginas com o *template* apresentado na Figura 1.

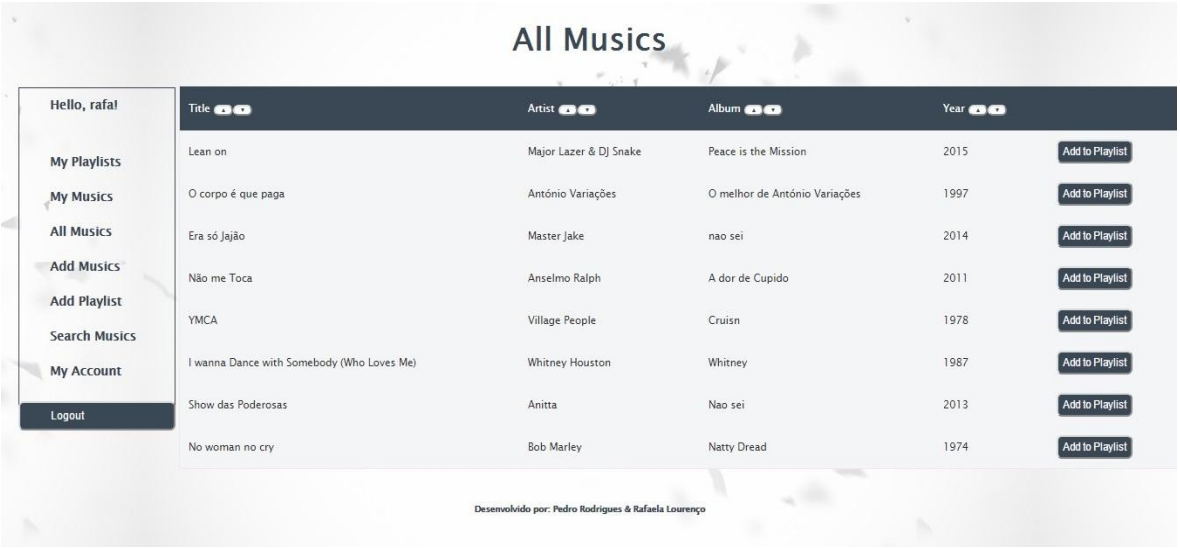

Figura 1 - Exemplo de uma página *xhtml* da nossa aplicação

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.

A parte mais complexa de implementar foi sem dúvida a adição de músicas a *playlists*, visto que o utilizador teria sempre de dar duas indicações antes de adicionar uma música a uma *playlist* sua. Pensámos então em mostrar ao utilizador todas as músicas da base dados e quando ele pretende-se adicionar uma a uma *playlist*, é feito o render de uma lista com as suas *playlists*, onde o utilizador selecciona a *playlist* onde quer adicionar a música. Acabou por ficar bastante intuitivo, e rápido, sem que o utilizador precise de carregar em muitos sítios para adicionar uma música a uma *playlist*.

De referir que foi implementado um filtro, que veda o acesso às páginas, caso o utilizador não tenha feito o seu *login*.

Outro aspecto interessante da camada de apresentação foi a listagem de todas as músicas, onde o utilizador tem acesso a todas as músicas existentes na base de dados, podendo ordená-las por artista, título da música, album ou ano de lançamento, além do menu de pesquisa de músicas, este é um menu que ajuda e bastante a pesquisa de músicas pelo utilizador.

Foi também garantida a segurança no armazenamento da *password* dos utilizadores, visto que esta é sempre encriptada quando há input, e nunca desencriptada, para que não seja de todo possível aceder às *password* de um utilizador em claro. Por exemplo, se um utilizador tentar fazer *login* com determinada *password*, esta é primeiro encriptada e só depois é comparado o seu valor com o que está na base de dados, para permitir ou não, o acesso à aplicação. Esta é sem dúvida uma boa prática, até porque mesmo em caso de haver uma fuga da informação da base de dados, esta não dá acesso às *password* dos utilizadores.

Para a encriptação propriamente dita, importámos a biblioteca *java*.*security* que dá acesso a várias classes e interfaces para uma *framework* de segurança.

Camada de negócio

Nesta camada, tratámos todo o negócio da nossa aplicação, é a camada responsável por fazer a ponte entre a camada da *web*, e a camada que irá fazer o acesso à base de dados. Decidimos usar *Local Beans*, *Stateless*, tomámos essa opção já que a aplicação apenas utiliza uma máquina e portanto todos os recursos são locais, bem como a base de dados. Também é feito o tratamento de grande parte das excepções da aplicação, tendo sempre o cuidado de apresentar ao utilizador mensagens de informação ou de erro, em todos os passos que este dá.

Obviamente tivémos sempre o cuidado de apenas deixar passar para a camada de apresentação os dados a que o utilizador deve ter acesso, sem permitir o acesso a informação da base de dados que não lhe diga respeito.

À excepção do *EJB* responsável por popular a base de dados no arranque da aplicação, todos os outros foram devidamente designados para que se saiba qual a sua função. Por exemplo, ao *EJB* responsável pelo acesso à parte das músicas da base de dados, chamámos *MusicDAO*, em que *DAO* significa *data acess object*, ou objecto responsável por aceder aos dados.

Assim fica bem claro, quais os *EJB's* responsáveis por aceder a que dados. Aprofundando um pouco mais, o *UserDAO* por exemplo, tem métodos para pesquisar todos os utilizadores da aplicação, pesquisar apenas um utilizador com o seu *id*, mudar dados de uma conta ou até apagar dados de um utilizador. Neste último caso decidimos que seria sensato, passar todas as suas músicas para um gestor, para que estas não deixassem de estar disponíveis para outros utilizadores. O *UserRegister* é também um *EJB* responsável por aceder a dados dos utilizadores, mas neste caso concreto, serve para verificar se o login está correcto, ou para adicionar um novo utilizador à base de dados.

No *PlaylistDAO* são feitos todos os acessos à base de dados necessários para pesquisar todas as *playlists*, pesquisar as *playlists* de um só utilizador, criar *playlists* novas, apagar *playlists* ou até alterar dados de uma *playlist*.

Por fim, no *MusicDAO* são feitos as *querys* necessárias para aceder a dados sobre músicas de formas muito variadas, tais como, pesquisar todas as músicas, pesquisar as músicas de determinada *playlist*, adcionar ou remover músicas de *playlists*, ordenar segundo critérios a *playlist* e pesquisar segundo critérios a *playlist*.

Na nossa opinião não era necessário fazer demasiadas *NamedQuerys*, pois estas são carregadas logo no arranque da aplicação e isso poderia sobrecarregar de forma desnecessária a aplicação, assim tendo a certeza que cada método chama apenas uma *query* temos a garantia de mais rapidez para o utilizador.

Camada de dados

De acordo com o nosso diagrama *ER*, neste projecto havia três entidades relacionadas entre si: o utilizador, que poderia ter várias *playlists* e que tinha a possibilidade colocar várias músicas na base de dados; as *playlists*, que poderiam conter várias músicas e que pertenciam, cada uma delas, apenas a um utilizador; as músicas que estariam possivelmente em várias *playlists* e que teriam sido colocadas na base de dados por um utilizador. Assim o diagrama ER final era o seguinte:

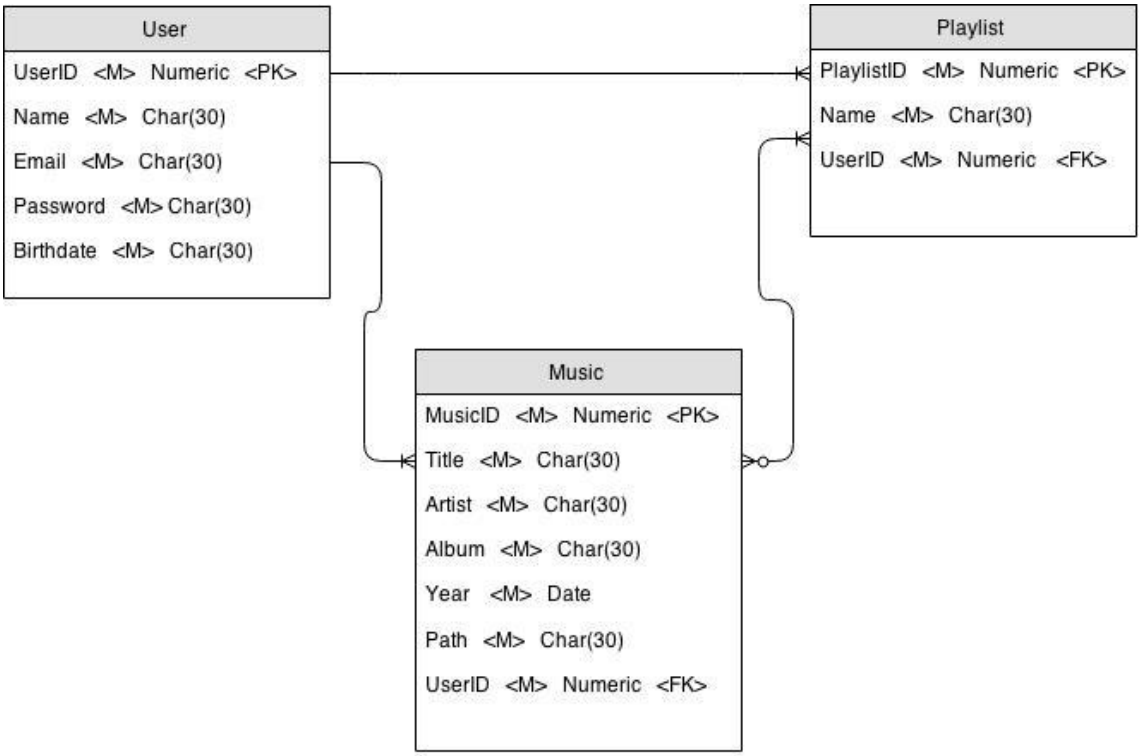

Figura 2 - Diagrama ER (obtido a partir do site draw.io)

É com base no relacionamento entre estas três entidades que funciona toda a ideia deste projecto, tivémos por isso especial cuidado na criação das entidades, com todas as anotações necessárias para termos a garantia de ter uma base de dados confiável, e que nos desse boas garantias de não poder ser corrompida de alguma forma.

Cada entidade foi também criada com os atributos que pensámos serem essenciais para persistir na base de dados, não iremos fazer uma lista detalhada dos atributos de cada entidade por acharmos redundante.

Empacotamento

A aplicação foi organizada em camadas, por forma a permitir a separação de artefactos, facilitando a manutenção de código, e futuros desenvolvimentos da aplicação.

Garantimos que cada camada possa existir de forma independente, por exemplo a camada de mais baixo nível (*JPA*), onde estão as entidades, não tem conhecimentos das outras camadas, não comunicando com os outros artefactos que fazer parte do projecto. A informação relativa a esta camada, era persistida na base de dados, sem saber das outras camadas.

Os *EJB's* (*Entreprise Java Beans*) estão numa outra camada, denominada camada de serviço, aqui é feita a gestão das entidades da aplicação, oferecendo a possibilidade de criar, actualizar, listar ou remover dados das entidades. Mais uma vez, esta camada não tem conhecimento das camadas acima de si, devolvendo dados sem saber para onde.

Depois temos a camada de apresentação, que contém todas as páginas que o utilizador vai ver, e que é responsável por toda a interação com o utilizador da aplicação. Aqui é feito o acesso aos *EJB's* que transmitem a informação necessária a apresentar ao utilizador. O módulo *EAR* é apenas responsável por empacotar todos os módulos num só para possibilitar a integração, transporte e instalação num servidor.

Todos estes quatro artefactos têm o seu próprio *pom.xml* mas, estão dentro de um projecto *Maven* a que se deu o nome de *theparent* que tem a função de definir e fornecer as dependências e *plugins* que os outros módulos utilizam.

Conclusão

Este era um projecto de todo aliciante, com uma arquitectura já muito mais avançada e com um tema que nos interessava particularmente.

Iniciámos com o esquema do projecto e começámos a ter várias ideias, mas toda a configuração do projecto e comunicação entre base de dados, parte de serviço e web, com a ferramenta *Maven* foi bastante complicada. Não tendo nesta fase tido o acompanhamento que que pensámos que teríamos da parte dos docentes e dos monitores. Assim o arranque do projecto atrasou-se demasiado tempo, e das três semanas disponíveis para o trabalho, apenas conseguimos começar a trabalhar efectivamente a mais de meio da duração estabelecida.

Devido a este atraso não nos foi possível testar a aplicação tão exaustivamente como desejávamos, e não pudémos implementar algumas das ideias que tínhamos. Foi mesmo bastante complicado cumprir o prazo, e apenas foi possível com bastante esforço de ambos os elementos do grupo, que diga-se funcionou na perfeição para o pouco tempo disponível, e falta de ajuda em certos momentos.

Apesar de todos os contratempos sentidos consideramos que a aplicação é bastante apelativa, funcional/intuitiva e segura para o utilizador.

Anexo

ID	Task Mode	Task Name	Duration	Start	Finish
1		Configuração do eclipse	6 days	Tue 19/05/15	Tue 26/05/15
2		Planeamento do Projecto	2 days	Tue 19/05/15	Wed 20/05/15
3		Criação do Projecto Maven	1 day	Mon 25/05/15	Mon 25/05/15
4		Implementação do Logging	1 day	Tue 26/05/15	Tue 26/05/15
5		Criação do Login	4 days	Wed 27/05/15	Sat 30/05/15
6		Ligação à base de dados	8 days	Wed 27/05/15	Fri 05/06/15
7		Criação dos filtros	2 days	Fri 29/05/15	Sat 30/05/15
8		Criação das Playlists	5 days	Mon 01/06/15	Fri 05/06/15
9		Implementação das alterações às Playlists	3 days	Wed 03/06/15	Fri 05/06/15
10		Criação de testes	3 days	Wed 03/06/15	Fri 05/06/15
11		Desenvolvimento do relatório	2 days	Fri 05/06/15	Sat 06/06/15

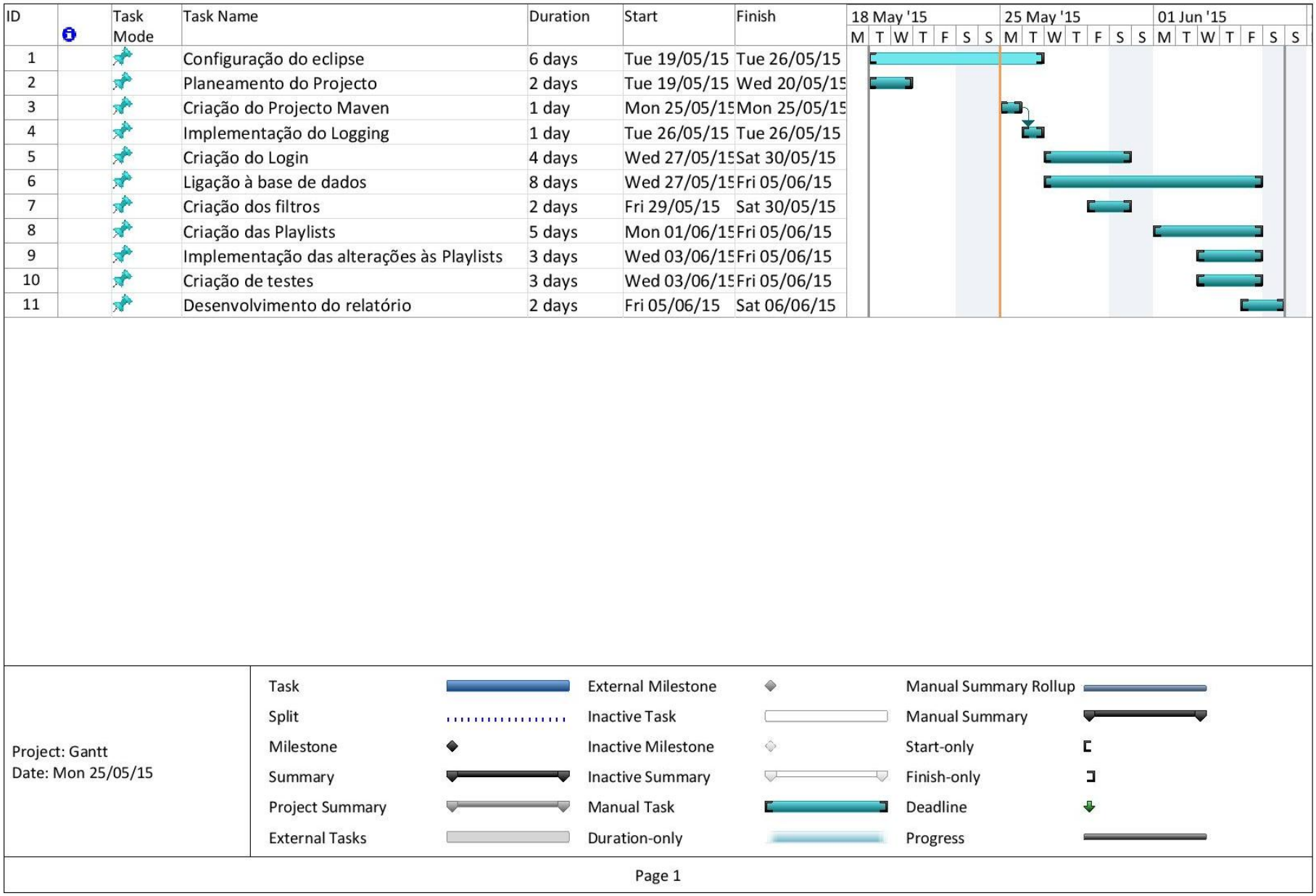

Figura 3 - Diagrama *Gantt* (Microsoft Project 2010)