UNIVERSIDADE DE COIMBRA -FACULDADE DE CIÊNCIAS E TECNOLOGIA

Departamento de Engenharia Informática

Programação Avançada em Java

Projecto N.º 6

Web Services: SOAP e REST & Segurança

João Pedro Oliveira Brandão Martins

Rafaela David Fernandes Lourenço

Coimbra, 4 de Julho de 2015

Índice

1 Introdução

Este projeto consiste em extender o projecto nº4 de forma a disponibilizar aos utilizadores as letras das músicas presentes na aplicação.

As letras são obtidas através de *web services* públicos apresentados no enunciado do projecto: o *LyricWiki* (http://api.wikia.com/wiki/LyricWiki) e o *ChartLyrics* (http://www.chartlyrics.com/api.aspx). Esta aplicação recorre a uma base de dados para guardar letras das músicas editadas pelo utilizador.

1.1 Ferramentas utilizadas

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

1.1.1 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*".

1.1.2 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.

1.1.3 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.

1.1.4 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.

1.1.5 GitHub

Para o desenvolvimento e conceção simultaneos 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*.

2 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.

2.1 Tecnologias utilizadas

2.1.1 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.

2.1.2 *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.

2.1.3 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 base de dados para validar credenciais do utilizador por ***form*** e respectivas ***roles***, por forma a impedir o acesso direto de um utilizador a uma área protegida. Essa protecção 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*.

2.1.4 *Javascript*

Neste projeto foi utilizado *Javascript residualmente*, incorporado nas paginas *xhtml* para operações associadas ao menu da página *web*.

2.2 Estrutura das páginas *XHTML*

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

2.2.1 resources/template/common

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

2.2.2 Zona não autenticada

2.2.2.1 index.xhtml

Página de arranque inicial que permite o utilizador efetuar ***sign-in*** (login de um utilizador já registado) e ***sign-up*** (registo de um novo utilizador).

2.2.3 Zona autenticada

2.2.3.1 resources/Authorized

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

2.2.3.1.1 addMusics.xhtml

Página que permite adicionar músicas à aplicação.

2.2.3.1.2 addPlaylist.xhtml

Página que permite adicionar playlists.

2.2.3.1.3 allMusics.xhtml

Página que permite a visualização de todas as músicas existente na aplicação.

2.2.3.1.4 changeAccount.xhtml

Página que permite ao utilizador editar a sua conta de utilizador e apaga-la do sistema.

2.2.3.1.5 editLyrics.xhtml

Página que é apenas acedida quando o utilizador deseja editar a letra de uma música de músicas.

2.2.3.1.6 myMusics.xhtml

Página que lista apenas as músicas do utilizador logado, permitindo a sua edição.

2.2.3.1.7 myPlaylist.xhtml

Página que permite gerir as playlists do utilizador logado.

2.2.3.1.8 searchMusics.xhtml

Página que permite efectuar pesquisas de músicas.

2.2.3.1.9 thePlaylist.xhtml

Página que permite lista as músicas de uma dada playlist selecionada, permitindo também a gestão das musicas adicionadas a essa playlist.

Interface no servidor

Para efetuarmos a interface das novas funcionalidades com a camada de negócio, alteramos e acrescentamos as classes descritas nos passos seguintes.

2.2.4 dei.uc.pt.ar/UserInput.java

Foi alterada esta classe para compatibilizar a autenticação por *container*, tendo sido alteradas os métodos de *login* e *logout*.

2.2.5 dei.uc.pt.ar/EditLyrics.java

Esta classe foi criada para permitir ao utilizador editar a letra as musicas. Gravando-a na base de dados.

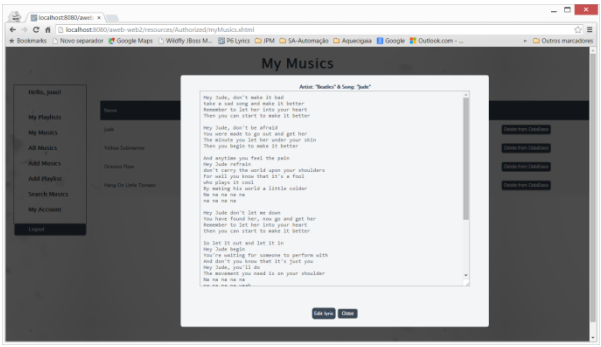
2.2.6 dei.uc.pt.ar/SearchMusicServer.java

Esta classe é a classe que vai fazer a busca das letras das musicas, efectuando a ligação para

2.3 Funcionalidades acrescentadas ao *Front-End*

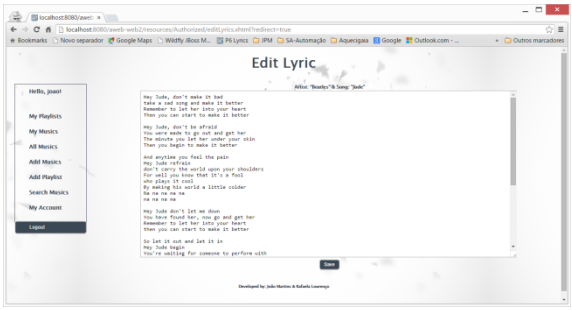
2.3.1 Popup Lyrics

A vizualização da letra da musica escolhia é efectuada através da abertura de uma *popup* na página web. Quando um utilizador altera e grava uma letra é esta letra que é apresentada na popup, já ná é feita a pesquisa no *web-service*.



2.3.2 Edição de *Lyrics*

Foi criada um novo *container* ao *CommonContent*, onde é apresentada a letra escolhida e é possível edita-la e salvá-la na base de dados.



3 Camada de Negócio

3.1 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.

3.2 LyricDAO.java

A classe LyricDao (*Data Transfer Objects)* foi criada para permitir fazer consultas à base de dados (queries) à entidade *Lyric*.

3.3 Classes Adicionais

3.3.1 UserLoged.java

Foi adicionada esta classe para possibilitar o controlo de utilizadores logados. Sempre que são utilizadas os métodos login e *logout* nas classes de interface da camada de apresentação, são guardados de forma não persistente os utilizadores que se autenticam. Para evitar erros na contagem de utilizadores logados em áarias sessões recorreu-se a *HashMaps*.

4 Aplicação de Serviços *Web*

4.1 Tecnologia Utilizada

Na camada de serviços *Web* foi utilizada a tecnologia *Rest*.

4.2 Classes

Foram criadas classes para implementar os vários *Web Services*: *SimpleUserService* tem os serviços associados aos utilizadores, *SimpleMusicService* associados às músicas e *SimplePlaylistService* associados às *Playlists*.

4.3 *Pojos*

Foram utilizados *POJOS* utilizando e utilizamos o próprio construtor de cada pojo para converter os dados provenientes da camada de negócio quer para os objetos já existentes para o projecto 4, quer para as novas funcionalidades desenvolvidas neste projeto.

4.4 Testes

Foram desenvolvidos vários testes que permitem verificar o estado (*Status*) que é devolvido quando é feita uma consulta na camada de serviços Web. A ferramenta *Postman*, foi crucial para o desenvolvimento dos testes.

5 Camada de *web service-client* das *Lyrics*

Para efectuar a pesquisa das letras nos *webservices* públicos foram utilizados dois tipos de *API* (*SOAP* e *REST*). Sendo que a partir do serviço LyricWiki apenas foi possível desenvolver o *API REST*.

5.1 Search-lyrics

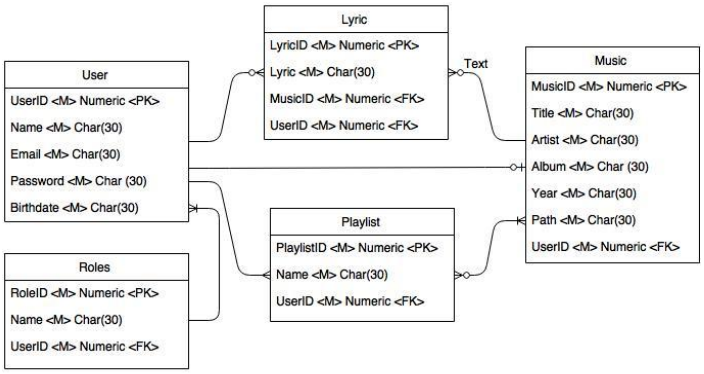
Foram criadas várias classes para pesquisa das *lyrics* nos diferentes servidores. Para o servidor *ChartLyrics*: *ChartLyricsRest*, *ChartLyricsSoap* e *GetLyricsChart*. Para o *LyricWiki*: *LyricsWikiaRest* e *GetLyricsWikia*.

A *API Soap* do *ChartLyrics*, através da criação de um cliente do *web service* (*WSDL*) criou várias classes de apoio ao acesso à aplicação, nomeadamente *GetLyricResult*, *SearchLyricResult eApiv1SoapProxy*.

6 Camada de Dados

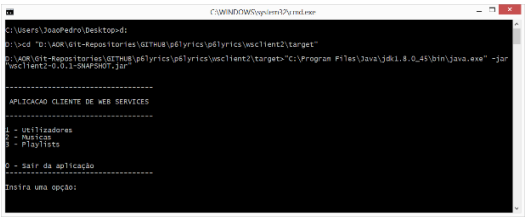
Na camada de dados a tecnologia utilizada foi *JPA* (*Java Persistence API*), que consiste numa API padrão do Java para a persistência de dados que o utiliza o modelo relacional. Esta *API* trás uma vantagem de possibilitar uma integração com a base de dados ao nível de entidades sem sequer ser necessário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

Ao desenho da base de dados existente, acrescentamos 2 entidades. Uma para as *roles* e outra para as *lyrics*. As relações estabelecidas entre estas novas entidades e as já existentes encontram-se na imagem apresentada ao lado.



7 Cliente *WebServices*

Foi criada uma aplicação cliente de *Web Services Stand-alone*. Trata-se de uma aplicação muito simples em modo texto. No entanto permite testar todas as funcionalidades desenvolvidas no módulo *WebServices producer*.



8 Empacotamento

O *Maven* recorre a um arquivo *XML* (*POM – Project Object Model*) para descrever todo o projeto de *software* em causa, sendo construídas as suas dependências sobre módulos e componentes externos, a ordem de compilação, diretórios 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 arquitetura baseada em *plugin*, a qual permite que ele faça uso de qualquer aplicação controlável através da entrada padrão.

Este projecto consiste em 7 sub-projetos.

- ***aweb-web2*** consiste na camada de apresentação.
- ***aservice-services2*** consiste na camada de negócio.
- ***thews-ws2*** consiste na camada de servidor de *web services*.
- ***thews-wslyrics*** consiste na camada de cliente de *web services* externos.
- ***adomain-domain2*** consiste na camada de dados.
- ***anear-ear2*** consiste na junção dos 5 anteriores.
- ***wsclient2*** trata-se de uma aplicação cliente *standalone* capaz de utilizar os *web services* do projecto.

8.1 *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: "*theparent2*" e possui diversas dependências tais como as necessárias para as tarefas comuns nos sub-projetos.

8.2 aweb-web2

8.2.1 Pom.xml

As principais dependências são JAVA EE, JAVAX, JUnit, Hamcrest, JFaces e Servlet. Gera um ficheiro *war* após *deploy*.

8.2.2 Web.xml

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

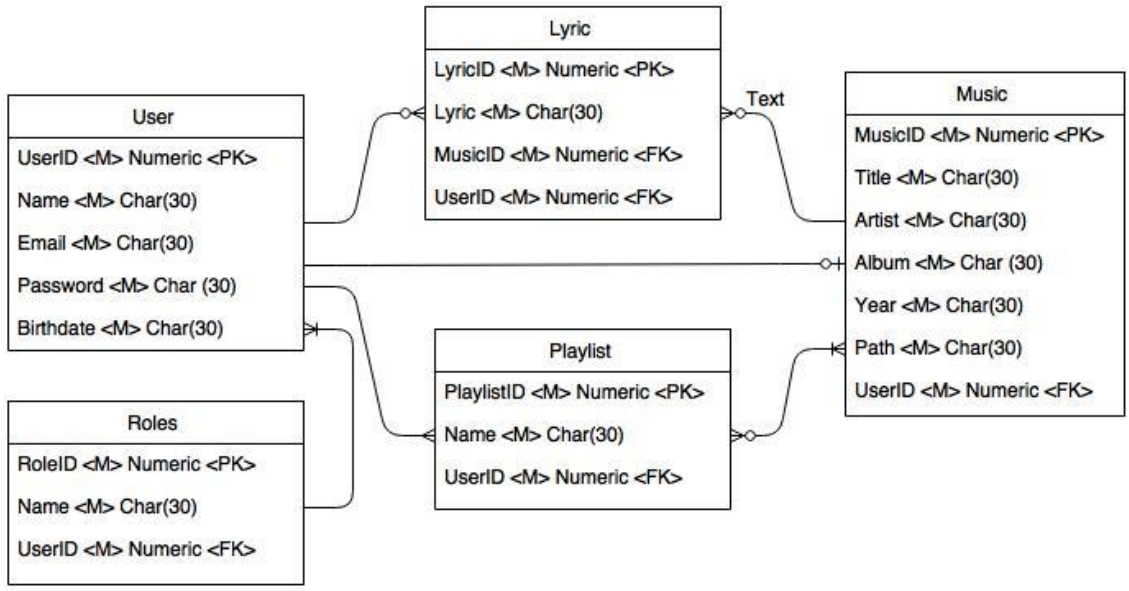
8.2.2.1 Welcome file list

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>
```

8.2.2.2 Autenticação

Neste projecto o ficheiro "*web.xml*" possui também informação relativa à configuração da autenticação por web *container* com o *FORM* do *log*



in, definição dos *containers* com acesso autenticado por utilizador/*password* e *roles*. Essas configurações tem de estar em conformidade com as configurações utilizadas *no standalone.xml* do *Wildfly*.

8.2.3 Jboss-web.xml

O ficheiro "*jboss-web.xml*" consiste na configuração do *security domain* utilizado nas configurações do *Wildfly*.

```
<jboss-web>
   <security-domain>p6lyricsDomain</security-domain>
</jboss-web>
```

8.3 thews-wsLyrics

As principais dependências são o sub-projecto *adomain-domain2*, *JAVAEE-API*, *org.apache.axis*, *JUnit*, *RESTEasy, commons-discovery, javax.activation, com.sun.mail, commons-logging, javax.xml.rpc* e *SLF4J*. Gera um ficheiro *ejb* após *deploy*.

8.4 thews-ws2

As principais dependências são o sub-projecto *adomain-domain2, JAVAEE-API, JUnit, RESTEasy, Hamcrest, Mockito, SLF4J*. Gera um ficheiro *war* após *deploy*.

8.5 aservice-services2

As principais dependências são o sub-projecto *adomain-domain2, JAVAX, JUnit, Hamcrest, Mockito, SLF4J*. Gera um ficheiro *ejb* após *deploy*.

8.6 adomain-domain2

8.6.1 Pom.xml

As dependências são *JAVAX* e *HIBERNATE*. Gera um ficheiro *ejb* após *deploy*.

8.6.2 *Persistence.xml*

O *datasource* (base de dados) criado no *Wildfly* para efectuar a ligação à base de dados é: *java:jboss/datasources/p6lyrics*

As classes acrescentadas à *entity manager* são:

```
<class>dei.uc.pt.ar.Roles</class>
<class>dei.uc.pt.ar.Lyric</class>
```

Roles foi criada para agregar *Roles* aos utilizadores para possibilitar acesso ao *container*.

Lyric agrega letra de músicas editadas associadas às *playlists*.

8.7 *anear-ear2*

As suas dependências são os sub-projectos anteriores. Após o dep*l*oy desta aplicação é gerado um ficheiro *ear* que engloba todos os resultados dos "*deploys*" das restantes dependencias

9 Conclusões

Com este trabalho comprovamos a vantagem na utilização de serviços web.

Não foram encontradas grandes dificuldades no desenvolvimento do projecto, sendo apenas a compreensão inicial da estrutura do projecto a mais relevante.

A implementação da autenticação pelo *container* revelou-se um pouco complicada, devido ao facto de já existir métodos de login através de filtros implementados no projecto 4.

A criação dos *web services REST* pedidos no enunciado foram criados com sucesso.

A pesquisa das letras ocorreu sem grandes percalços, sendo o desenvolvimento da *popup* o ponto mais delicado.

Como resultado final ambas as aplicações *standalone* desenvolvidas possuem um visual limpo e um caracter bastante intuitivo de utilização.