

Rapport - Sujet B3 : Requêtes et visualisation de données issues de Dbpedia

Peter Serge

08/03/2015

Formation MAS-ICT CAS : WEB SEMANTIQUE

Julien Tscherrig, Omar Abou Khaled et Maria Sokhn

Rapport de réalisation d'une application de mise en relation de données issues de dbpedia.

Table des matières

1. Introduction	3
1.1 Rappel sur les objectifs	3
2. Analyse	4
2.1 Liste des artistes (Query 1.1)	4
2.2 Détail sur un artiste (Query 1.2).....	5
2.3 Recherche d'artistes ou bands liés (Query 1.3).....	6
2.4 Recherche des genres (Query 1.4)	7
3. Architecture technique.....	8
3.1 Composants	8
3.2 Dbpedia et sparql.....	8
3.3 Jena.....	8
3.4 Spring	9
3.5 Junit & Spring test.....	9
3.6 AngularJS	9
3.7 Maven.....	9
4. Mise en route	10
4.1 Installation de Java.....	10
4.2 Installation de Maven	10
4.3 Installation de Eclipse.....	10
4.4 Installation de Tomcat	11
5. Réalisation	14
5.1 Backend.....	14
5.2 Maven Web Application	14
5.3 Organisation des packages	16
5.4 Diagramme de classe simplifié	17
5.5 Beans	18
5.6 Services.....	19
5.7 Controller REST.....	22
5.8 Tests.....	23
5.9 Frontend.....	25
5.10 Bootstrapping et Templating	25
5.11 Construction de la vue	27

5.12	Vues de détails	29
5.13	Code JavaScript	30
6.	Exécution de l'application	32
6.1	Directement dans Eclipse	32
6.2	En mode ligne de commande avec Maven	33
6.3	Sur le site Openshift	33
7.	Conclusion	34
8.	Références	35

1. Introduction

Ce document est un rapport pour la réalisation d'un mini projet ; dans le cadre de la formation MAS-ICT CAS : WEB SEMANTIQUE Julien Tscherrig, Omar Abou Khaled et Maria Sokhn.

Mon application est composée de deux parties principales, le backend, application web java permettant d'opérer comme fournisseur de données et cachant la complexité des requêtes sparql ; et un frontend écrit en JavaScript avec l'aide du Framework structurant AngularJS. Ces deux principales composantes seront détaillées plus tard dans ce document.

1.1 Rappel sur les objectifs

Créer une application Web en Java permettant de visualiser des groupes musicaux, membres et albums.

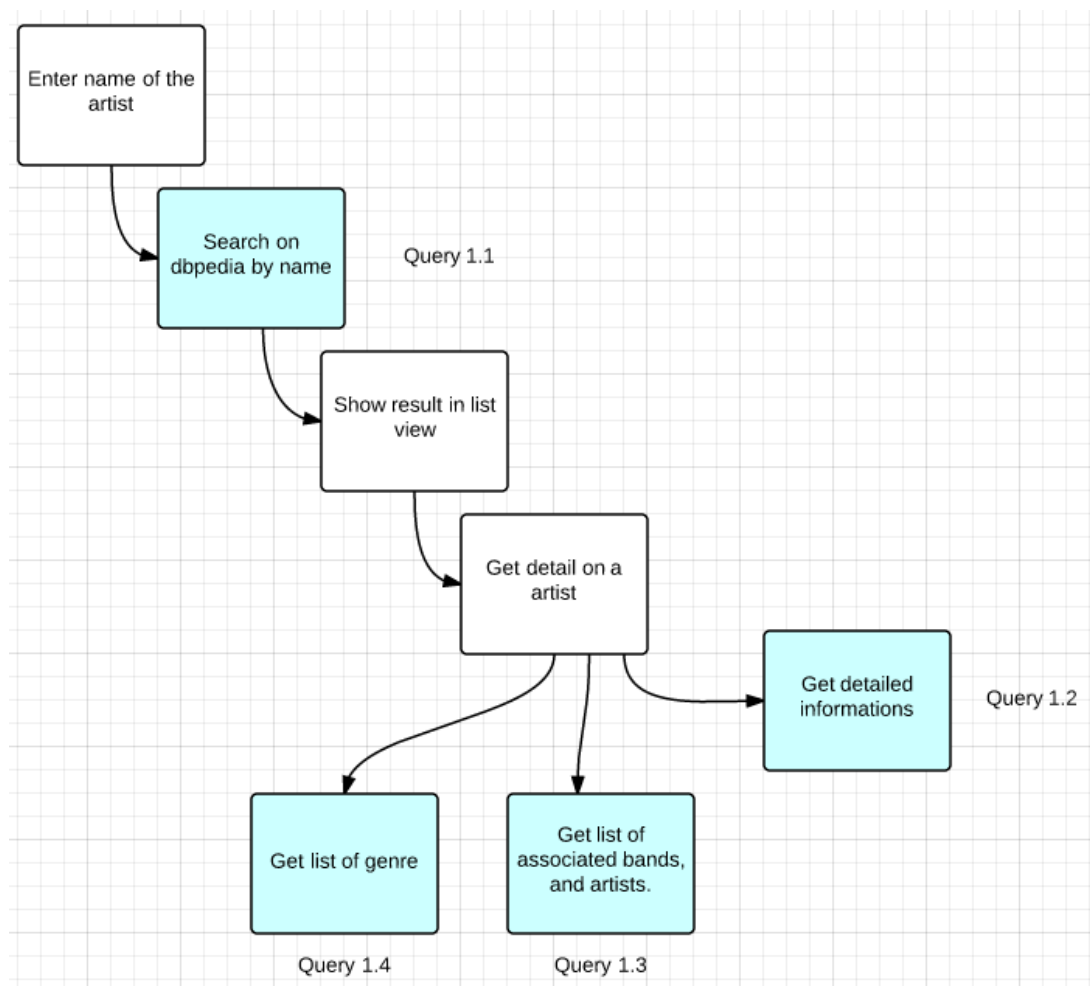
En partant d'un nom d'artiste, il faudra faire la liste des groupes musicaux liés, albums réalisés ainsi que des informations liées comme le style.

Il serait intéressant de pouvoir mettre en relation deux artistes par rapport aux informations trouvées (groupes, albums, style musical, ...).

2. Analyse

L'application obtiendra ces données via des requêtes sparql, faites directement sur les sources de données de dbpedia.org.

Le workflow principal correspond à ceci :



L'application commence par une recherche sur la source de données <http://dbpedia.org/ontology/MusicalArtist> et affichera une liste d'artiste (Query 1.1). On pourra ensuite afficher des détails supplémentaires dans un écran dédié, afficher la liste de genres (Query 1.4) associés ainsi que les artistes (Query 1.2) et groupes liés (Query 1.3).

2.1 Liste des artistes (Query 1.1)

Nous voulons dans un premier temps obtenir la liste des artistes par leur nom (ou fragment).

```

prefix prop: <http://dbpedia.org/property/>
prefix owl: <http://dbpedia.org/ontology/>
prefix res: <http://dbpedia.org/resource/>
prefix rdfs: http://www.w3.org/2000/01/rdf-schema#

select distinct ?artist ?name
where {
  ?artist a owl:MusicalArtist .

```

```
?artist rdfs:label ?name .

FILTER (regex(?name, 'Mark Knopfler', 'i')
        && langMatches(lang(?name), 'en'))

} LIMIT 100
```

A noter que notre source de données est : <http://dbpedia.org/ontology/MusicalArtist> qui est une sous-classe de <http://dbpedia.org/ontology/Artist>

About: musical artist

An Entity of Type : [Class](#), from Named Graph : <http://dbpedia.org/resource/classes#>, within Data Space : dbpedia.org

Property	Value
rdf:type	<ul style="list-style-type: none"> owl:Class
rdfs:isDefinedBy	<ul style="list-style-type: none"> http://dbpedia.org/ontology/
rdfs:label	<ul style="list-style-type: none"> musical artist musicien musikalischer Künstler
rdfs:subClassOf	<ul style="list-style-type: none"> http://schema.org/MusicGroup dul:NaturalPerson dbpedia-owl:Artist
wdrs:describedby	<ul style="list-style-type: none"> dbpedia-owl:data/definitions.xml dbpedia-owl:data/definitions.ttl dbpedia-owl:data/definitions.jsonld
prov:wasDerivedFrom	<ul style="list-style-type: none"> http://mappings.dbpedia.org/index.php/OntologyClass:MusicalArtist
is http://open.vocab.org/terms/defines of	<ul style="list-style-type: none"> http://dbpedia.org/ontology/
is http://open.vocab.org/terms/describes of	<ul style="list-style-type: none"> dbpedia-owl:data/definitions.xml dbpedia-owl:data/definitions.ttl dbpedia-owl:data/definitions.jsonld
is rdfs:domain of	<ul style="list-style-type: none"> dbpedia-owl:musicBand dbpedia-owl:numberOfAlbums dbpedia-owl:numberOfLiveAlbums dbpedia-owl:numberOfStudioAlbums
is rdfs:range of	<ul style="list-style-type: none"> dbpedia-owl:associatedMusicalArtist dbpedia-owl:musicComposer dbpedia-owl:musicalArtist dbpedia-owl:musicians
is rdfs:subClassOf of	<ul style="list-style-type: none"> dbpedia-owl:ClassicalMusicArtist dbpedia-owl:Instrumentalist dbpedia-owl:BackScene dbpedia-owl:MusicDirector dbpedia-owl:Singer

Dans cette requête nous filtrons les résultats qui sont en anglais en utilisant la fonction FILTER

```
FILTER (langMatches(lang(?name), 'en'))
```

La sélection des données se fait aussi en utilisant FILTER avec l'option i pour ne pas tenir compte de la case:

```
FILTER (regex(?name, 'Mark Knopfler', 'i'))
```

2.2 Détail sur un artiste (Query 1.2)

Pour ce faire il faut utiliser l'URI de la ressource dans la sélection des données comme ceci et la propriété sameAs

```
?artist owl:sameAs? <http://dbpedia.org/resource/Mark_Knopfler> .
```

Attention aussi aux champs **null**, qui ici sont traités avec la fonction OPTIONAL afin de sortir les données partielles de la requête.

```
OPTIONAL { ?artist owl:birthPlace ?birthPlace }
```

Ce qui fait une requête comme ceci:

```
prefix prop: <http://dbpedia.org/property/>
prefix owl: <http://dbpedia.org/ontology/>
prefix res: <http://dbpedia.org/resource/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

select *
where {

?artist owl:sameAs? <http://dbpedia.org/resource/Mark_Knopfler> .
?artist rdfs:label ?name .

OPTIONAL { ?artist owl:birthDate ?birthDate }
OPTIONAL { ?artist owl:birthPlace ?birthPlace }
OPTIONAL { ?artist owl:thumbnail ?image}
OPTIONAL { ?artist prop:shortDescription ?shortDescription }
OPTIONAL { ?artist owl:abstract ?abstract }
OPTIONAL { ?artist prop:website ?website }
OPTIONAL { ?artist prop:yearsActive ?yearsActive }

FILTER (langMatches(lang(?name), 'en')
&& langMatches(lang(?abstract), 'en'))

} LIMIT 1
```

2.3 Recherche d'artistes ou bands liés (Query 1.3)

Là aussi la propriété sameAs permet de faire la sélection au niveau de la ressource. On procèdera de la même manière pour les propriétés fournissant une liste de valeurs come owl:associatedBand, owl:associatedArtist.

```
prefix prop: <http://dbpedia.org/property/>
prefix owl: <http://dbpedia.org/ontology/>
prefix res: <http://dbpedia.org/resource/>
prefix rdfs: http://www.w3.org/2000/01/rdf-schema#

select distinct * where { ?artist owl:sameAs?
http://dbpedia.org/resource/Mark_Knopfler> .
?artist rdfs:label ?name .
?artist owl:associatedBand ?associatedBand.
?associatedBand rdfs:label ?assName .

FILTER (langMatches(lang(?name), 'en')
&& langMatches(lang(?assName), 'en'))
} LIMIT 100
```

2.4 Recherche des genres (Query 1.4)

Voici un exemple de requête, similaire au point précédent en utilisant la propriété owl:genre.

```
prefix prop: <http://dbpedia.org/property/>
prefix owl: <http://dbpedia.org/ontology/>
prefix res: <http://dbpedia.org/resource/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

select *
where {
  ?artist owl:sameAs? <http://dbpedia.org/resource/Mark_Knopfler> .
  ?artist rdfs:label ?name .

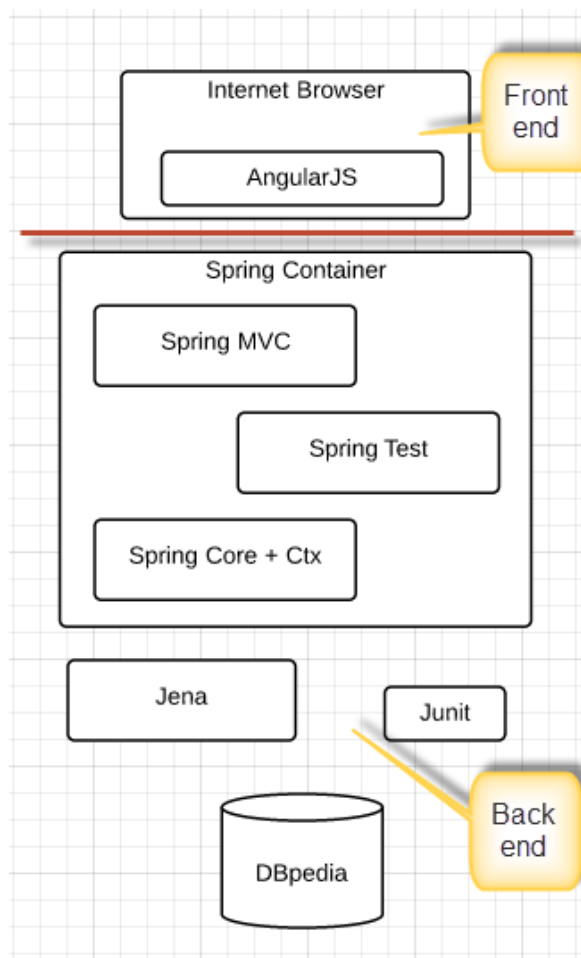
  OPTIONAL { ?artist owl:genre ?genre}
  OPTIONAL { ?genre rdfs:label ?genrelabel}

  FILTER (langMatches(lang(?name), 'en')
    && langMatches(lang(?genrelabel), 'en') )

} LIMIT 10
```


3. Architecture technique

Nous utiliserons ici Spring pour créer une application Web permettant de mettre en relation les diverses couches. Spring MVC nous permet de construire une couche service et de l'exposer pour qu'elle puisse être utilisée du côté front.



3.1 Composants

Voici une brève présentation des éléments composants notre architecture logicielle.

3.2 *Dbpedia et sparql*

Dbpedia fera office ici de source de données RDF. Nous utilisons des requêtes sparql pour requêter les données fournies dans <http://dbpedia.org>.

L'outil disponible sur internet <http://dbpedia.org/sparql> permet de mettre au point ces requêtes avant d'en faire l'intégration dans du code Java.

3.3 *Jena*

Jena est une librairie permettant d'exploiter des données issues de sources de données sémantiques.

Cette librairie fait pas mal de chose comme par exemple exécuter des commandes sparql à

l'image de ce qui se fait sur des bases de données traditionnelles.

Nous trouvons toutes les informations sur le site : <https://jena.apache.org>.

Nous utiliserons Jena pour exécuter des requêtes sparql et récupérer les données résultantes dans des ResultSets java.

3.4 *Spring*

Spring est un Framework très populaire dans le monde des applications d'entreprise Java JEE. Le but du Framework est de simplifier au maximum la mise en œuvre pour éviter de perdre trop de temps sur des aspects liés à la configuration ou au code dit technique.

Nous utiliserons Spring MVC pour nous faire l'exposition des services REST et de fournir une vue Web à notre application.

Le site de Spring <http://projects.spring.io/spring-framework/> fournit toute sorte d'information sur ce Framework.

3.5 *JUnit & Spring test*

JUnit est utilisé pour réaliser des tests en plus de Spring test, qui nous permet de charger les services, le contexte Spring et la configuration.

3.6 *AngularJS*

AngularJS est le Framework utilisé pour réaliser la partie front ou présentation de l'application. Nous utilisons aussi Twitter Bootstrap afin de disposer de toute sorte de composant graphique dans le rendu de l'application.

3.7 *Maven*

Tous les composants sont définis dans le fichier pom.xml de l'application et permet d'être obtenu via ce Framework dynamiquement.

4. Mise en route

Les outils nécessaires pour effectuer ces développements sont :

Composants	Version	Location
Java JDK	1.7+	http://www.oracle.com/technetwork/java/javase/downloads/index.html
Maven	3.0 et plus	http://maven.apache.org/download.cgi
Tomcat	8	http://tomcat.apache.org/download-80.cgi
Eclipse	Luna	https://www.eclipse.org/downloads/

4.1 Installation de Java

Java JDK est disponible ici :

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Une fois le logiciel téléchargé faite l'installation. Ceci ne pose pas de difficulté.

A noter que nous avons besoin du JDK et que le JRE ne suffit pas.

4.2 Installation de Maven

Maven est disponible ici :

<http://maven.apache.org/download.cgi>

Une fois le package apache-maven-3.*.*-bin.zip downloadé, décompressez dans un répertoire local (par exemple c:\apps).

Vous devez ensuite :

- 1) Créer une variable d'environnement M2_HOME correspondant au répertoire d'installation de Maven.
- 2) Créer une variable M2 correspondant a %M2_HOME%\bin
- 3) Ajouter M2 au PATH.

Il faut aussi créer un fichier settings, contenant les chemins locaux de repositories, si existant ; des informations d'utilisation de proxy ou autre.

Ce fichier doit être placé ici. <HOME>/.m2/settings.xml.

Plus d'info sont disponible ici : <http://maven.apache.org/settings.html>.


4.3 Installation de Eclipse

Eclipse Luna est disponible ici :

<https://www.eclipse.org/downloads>


Il faut ensuite faire attention de prendre la version EE :

Package Solutions
Filter Packages ▼



Eclipse IDE for Java EE Developers, 254 MB
Downloaded 2,222,024 Times

Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn...



Windows 32 Bit
Windows 64 Bit

Une fois le package téléchargé, décompresser et exécuter le fichier eclipse.exe.

4.4 Installation de Tomcat

Tomcat est disponible ici :

<http://tomcat.apache.org/download-80.cgi>

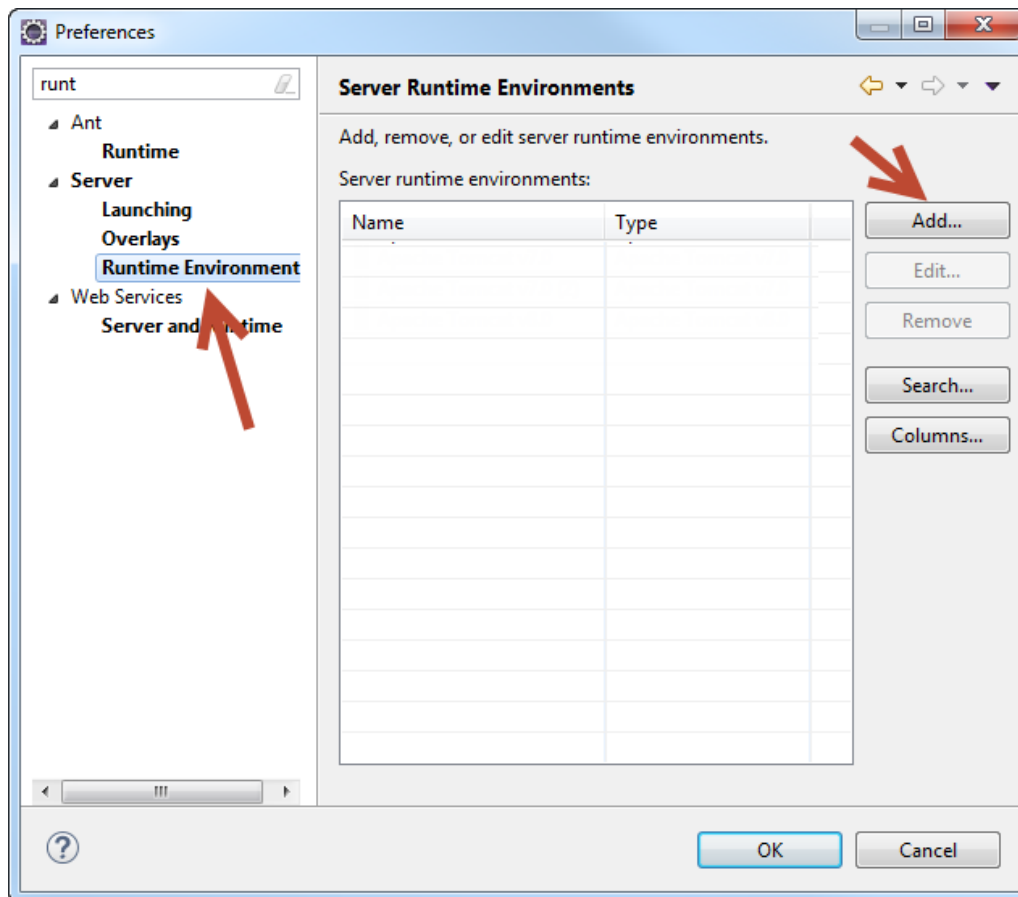
Une fois le package téléchargé, il faut le décompresser en local (par exemple sur c:\apps\tomcat8).

Nous utiliserons Eclipse pour les opérations de start/stop ainsi que le déploiement de l'application.

Pour ce faire il faut créer un serveur Tomcat8 dans Eclipse comme ceci :

Ajouter un runtime Tomcat :

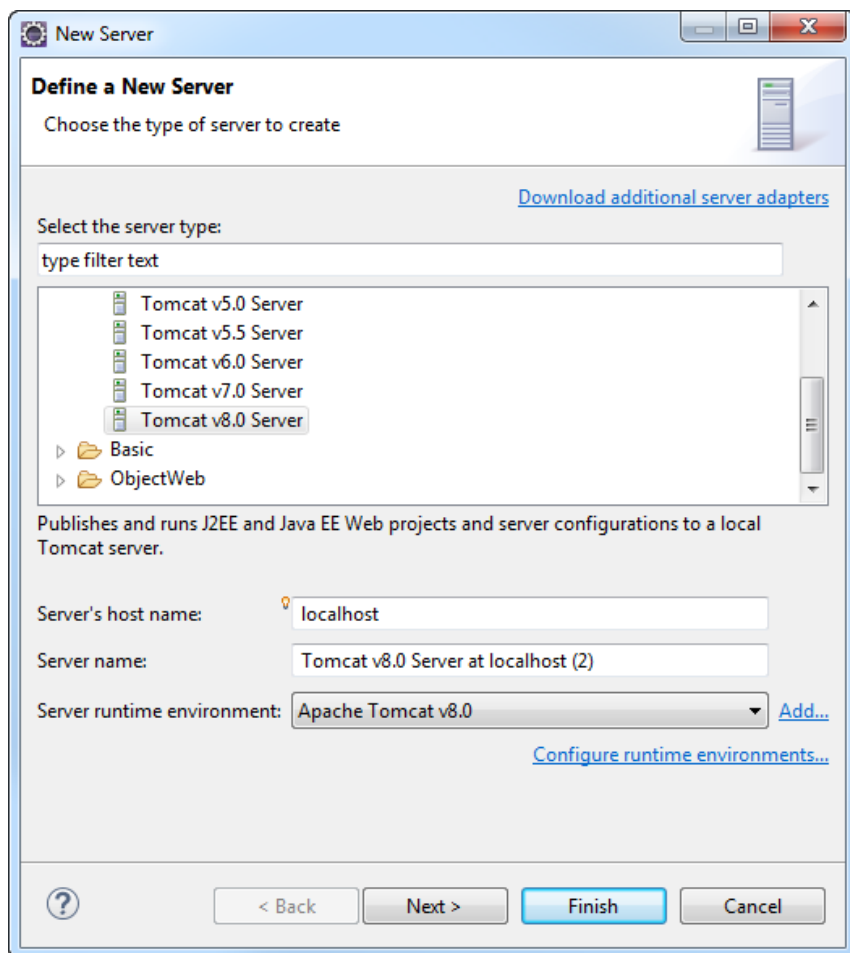
- Dans l'onglet Préférences de Eclipse, rechercher : runtime ; cliquer sur Add...



- Définissez le répertoire Tomcat de travail, ainsi qu'un JRE, qui doit être préalablement défini dans Eclipse, ou le runtime par défaut.

Ajouter un serveur Tomcat :

- Dans l'onglet « Servers », visible dans la perspective JEE, créez un nouveau serveur basé sur le runtime Tomcat 8 :



5. Réalisation


5.1 Backend

5.2 Maven Web Application

Nous commençons le projet par la création d'une application web Maven (basée sur l'archétype maven-archetype-webapp ; le contenu sera adapté à notre besoin.

Pour commencer nous nommons notre application dont le packaging sera war

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.
2   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apaci
3   <modelVersion>4.0.0</modelVersion>
4
5   <groupId>com.music.dbpedia</groupId>
6   <artifactId>artistfinder</artifactId>
7   <version>1.0.0</version>
8   <packaging>war</packaging>
9   <name>finder</name>
```



Voici les dépendances nécessaires :

```

19      <!-- JENA API -->
20      <dependency>
21          <groupId>org.apache.jena</groupId>
22          <artifactId>apache-jena-libs</artifactId>
23          <type>pom</type>
24          <version>2.12.1</version>
25      </dependency>
26
27      <!-- SPRING -->
28      <dependency>
29          <groupId>org.springframework</groupId>
30          <artifactId>spring-core</artifactId>
31          <version>4.0.2.RELEASE</version>
32      </dependency>
33      <dependency>
34          <groupId>org.springframework</groupId>
35          <artifactId>spring-webmvc</artifactId>
36          <version>4.0.2.RELEASE</version>
37      </dependency>
38      <dependency>
39          <groupId>org.springframework</groupId>
40          <artifactId>spring-context-support</artifactId>
41          <version>3.2.11.RELEASE</version>
42          <scope>compile</scope>
43      </dependency>
44
45      <!-- Servlet et JSP, provided -->
46      <dependency>
47          <groupId>javax.servlet</groupId>
48          <artifactId>javax.servlet-api</artifactId>
49          <version>3.0.1</version>
50          <scope>provided</scope>
51      </dependency>
52      <dependency>
53          <groupId>javax.servlet.jsp</groupId>
54          <artifactId>jsp-api</artifactId>
55          <version>2.2</version>
56          <scope>provided</scope>
57      </dependency>
58
59      <!-- TEST framework -->
60      <dependency>
61          <groupId>org.springframework</groupId>
62          <artifactId>spring-test</artifactId>
63          <version>4.0.2.RELEASE</version>
64      </dependency>
65
66      <dependency>
67          <groupId>junit</groupId>
68          <artifactId>junit</artifactId>
69          <version>4.11</version>
70          <scope>test</scope>
71      </dependency>

```

- 1) Apache Jena (de type pom), nous permet d'obtenir d'un coup les librairies nécessaires à la librairie,
- 2) Les librairies Spring, en version 4.0.2. Spring Mvc nous permettra de créer les services REST,
- 3) Les classes servlets utile à l'IDE Eclipse en provided,
- 4) Les librairies de test Junit et Spring Test.

Nous définissons aussi quelques plugins Maven dont le plugin Tomcat afin de faciliter l'exécution du projet en dehors d'Eclipse.


```

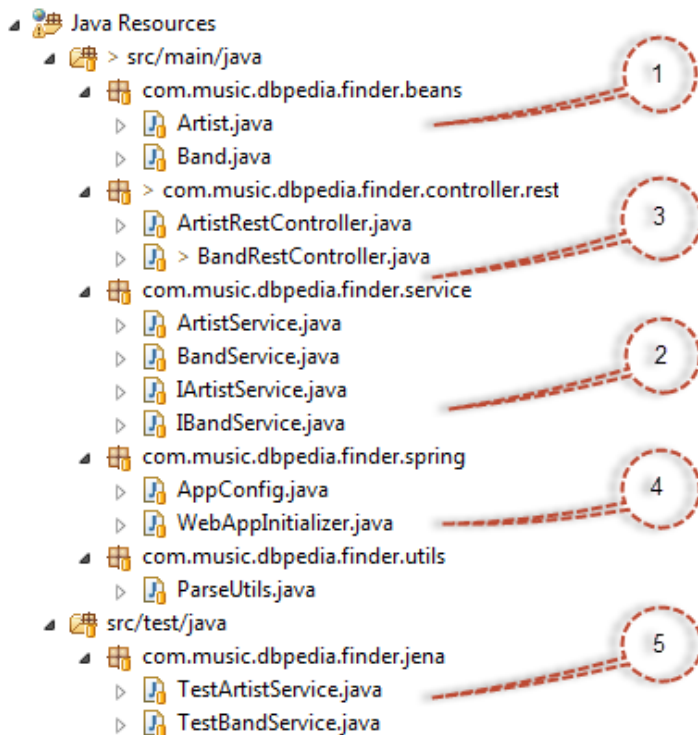
74 <build>
75   <finalName>artistfinder</finalName>
76   <plugins>
77     <plugin>
78       <groupId>org.apache.maven.plugins</groupId>
79       <artifactId>maven-compiler-plugin</artifactId>
80       <version>3.1</version>
81       <configuration>
82         <source>1.7</source>
83         <target>1.7</target>
84       </configuration>
85     </plugin>
86     <plugin>
87       <groupId>org.apache.maven.plugins</groupId>
88       <artifactId>maven-deploy-plugin</artifactId>
89       <version>2.7</version>
90     </plugin>
91     <plugin>
92       <groupId>org.apache.tomcat.maven</groupId>
93       <artifactId>tomcat7-maven-plugin</artifactId>
94       <version>2.1</version>
95       <configuration>
96         <port>8380</port>
97       </configuration>
98     </plugin>
99     <plugin>
100       <groupId>org.apache.maven.plugins</groupId>
101       <artifactId>maven-resources-plugin</artifactId>
102       <version>2.6</version>
103       <configuration>
104         <encoding>UTF-8</encoding>
105       </configuration>
106     </plugin>
107     <plugin>
108       <groupId>org.apache.maven.plugins</groupId>
109       <artifactId>maven-dependency-plugin</artifactId>
110       <version>2.8</version>
111     </plugin>
112   </plugins>
113   <resources>
114     <resource>
115       <directory>src/main/resources</directory>
116       <filtering>true</filtering>
117     </resource>
118   </resources>
119 </build>

```

- 1) Compilation plugin en java 1.7
- 2) Plugin de déploiement,
- 3) Plugin Tomcat7, permettant d'exécuter l'application via un ***mvn tomcat7:run***,
- 4) Gestion des ressources,
- 5) Plugin de gestion de dépendance
- 6) Définition des fichiers ressources à packager.

5.3 Organisation des packages

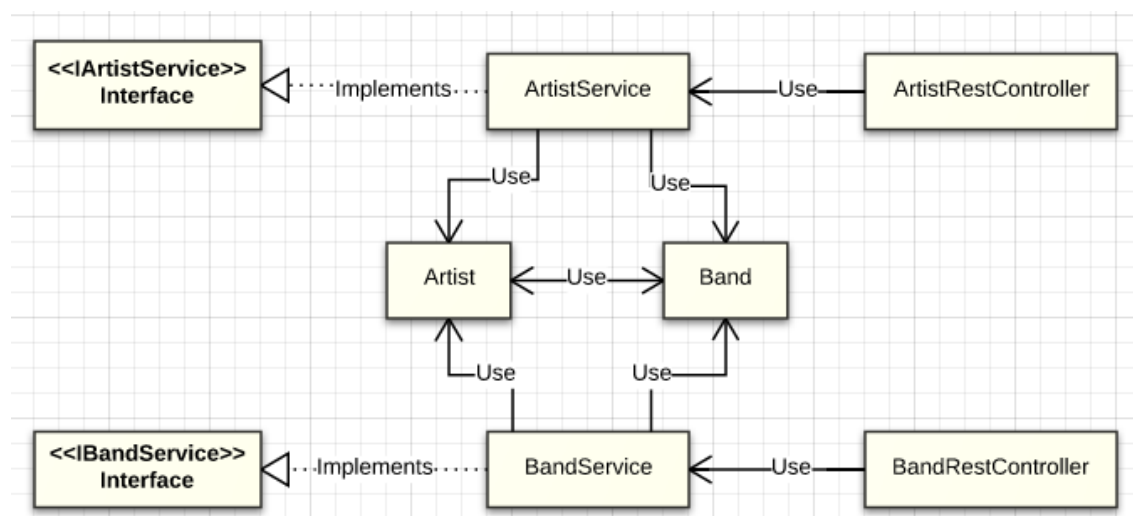
Notre application est très simple et ne comporte pas beaucoup de composant :



- 1) Beans contient deux structures utilisées pour stocker les informations relatives aux deux principales vues,
- 2) Service contient les classes qui exécutent les requêtes sparsql,
- 3) Les contrôleurs REST sont les classes qui font l'exposition des services REST, interface entre le backend et le frontend,
- 4) Les classes application et configuration Spring (on oublie les fichiers de contexte pour faire la configuration par annotation),
- 5) Les classes de tests.

5.4 Diagramme de classe simplifié

Voici une vue simplifiée de l'application backend :



Voici les différents types de classe de manière plus détaillée.

5.5 Beans

Les Beans sont des classes simples permettant de stocker les données issues de manière plus organisée.

En voici un exemple :

```

8
9 /**
10  * Artist bean
11  * @author speter
12  *
13  */
14 public class Artist {
15
16     @JsonIgnore
17     private Resource resource;
18
19     private String name;
20     private String birthDate;
21     private String deathDate;
22     private String comment;
23
24     private String imageURL;
25
26     @JsonIgnore
27     private Resource birthPlace;
28
29     private String birthPlaceStr;
30     private String country;
31
32     @JsonIgnore
33     private Resource deathPlace;
34
35     private String deathPlaceStr;
36
37     private String shortDescription;
38     private String abstractStr;
39     private String yearsActive;
40     private URL website;
41
42     private List<String> instruments;
43     private List<Artist> associatedArtists;
44     private List<Band> associatedBands;
45     private List<String> genres;

```

On peut noter les points suivants :

- 1) Champs de type Resource, permettant de stocker une ressource au sens sémantique du terme, @JsonIgnore empêche cet objet d'être passé au frontend.
- 2) Champs de tout type (String, Date, URL).
- 3) Listes de valeurs (de tout type).

Nous fournirons à la partie front les URI correspondant aux ressources comme ceci :

```

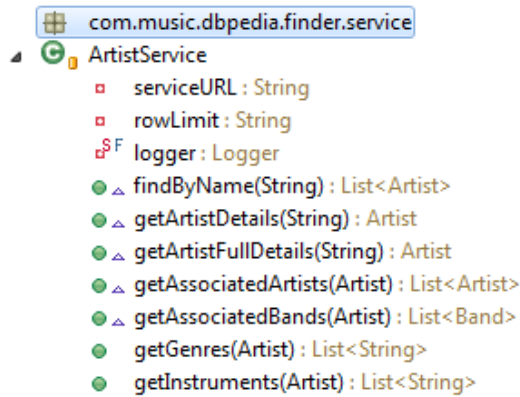
public String getArtistURI() {
    if (resource != null && resource instanceof Resource){
        return resource.getURI();
    } else {
        return null;
    }
}

```

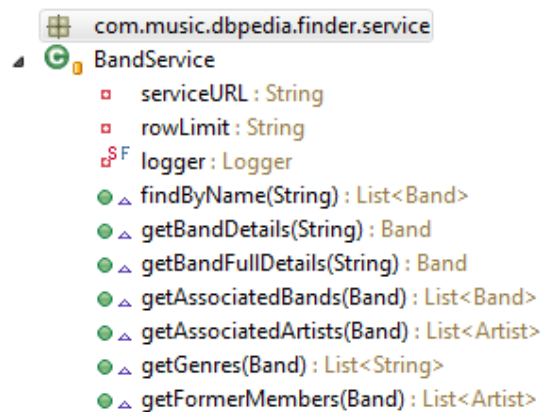
5.6 Services

Les services implémentent les interfaces correspondantes. C'est cette couche qui exécute les commandes sparql.

Par exemple pour un artiste, voici les méthodes implémentées qui permette de fournir les informations à la vue :



Et pour un groupe



Les commandes sont exécutées comme ceci :

```

37 @Override
38 public List<Artist> findByName(String name) {
39
40     List<Artist> artists = new ArrayList<Artist>();
41
42     String queryString = " prefix prop: <http://dbpedia.org/property/> \n"
43         + " prefix owl: <http://dbpedia.org/ontology/> \n"
44         + " prefix res: <http://dbpedia.org/resource/> \n"
45         + " prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> \n"
46         + " select distinct * \n"
47         + " where { \n"
48         + " ?artist a owl:MusicalArtist . \n"
49         + " ?artist rdfs:label ?name . \n"
50         + " OPTIONAL { ?artist owl:birthDate ?birthDate } \n"
51         + " OPTIONAL { ?artist prop:shortDescription ?shortDescription } \n"
52         + " OPTIONAL { ?artist prop:website ?website } \n"
53         + " OPTIONAL { ?artist prop:yearsActive ?yearsActive } \n"
54         + " FILTER (regex(?name, '.*' + name + '.*', 'i') "
55             + "&& langMatches(lang(?name), 'en')) \n"
56         + " } LIMIT " + rowLimit + " \n";
57
58     Query query = QueryFactory.create(queryString);
59
60     QueryExecution qe = QueryExecutionFactory.sparqlService(serviceURL, query);
61
62     try {
63
64         ResultSet res = qe.execSelect();
65
66         Artist artist = new Artist();
67
68         while (res.hasNext()) {
69
70             QuerySolution row = res.next();
71
72             artist = new Artist();
73             artist.setResource(ParseUtils.parseXMLResource(row.get("artist").asResource()));
74             artist.setBirthDate(ParseUtils.parseXMLDate(row.get("birthDate")));
75             artist.setName(ParseUtils.parseXMLDate(row.get("name")));
76             artist.setShortDescription(ParseUtils.parseXMLString(row.get("shortDescription")));
77             artist.setYearsActive(ParseUtils.parseXMLString(row.get("yearsActive")));
78             artist.setWebsite(ParseUtils.parseXMLURL(row.get("website")));
79
80             artists.add(artist);
81             logger.log(Level.INFO, "Artist found : " + artist.toString());
82
83         }
84
85         return artists;

```

- 1) Mise en forme d'une requête (sous la forme d'une chaîne de caractère) permettant ici de rechercher un artiste,
- 2) Connexion au service de données : <http://dbpedia.org/sparql>,
- 3) Exécution de la requête,
- 4) Récupération des données sous forme d'un ResultSet.

Pour des raisons de performance j'ai séparé la méthode `getArtistDetails` en deux pour éviter de multiplier les requêtes successives sparql. Il serait judicieux de le faire encore plus, afin de rendre l'obtention des données plus fluide.

Voici comment les requêtes d'obtention des détails sont faites, par exemple pour un artiste :

```
String queryString = " prefix prop: <http://dbpedia.org/property/> \n"
+ " prefix owl: <http://dbpedia.org/ontology/>\n"
+ " prefix res: <http://dbpedia.org/resource/> \n"
+ " prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>\n"
+ " select *\n"
+ " where {\n"
+ "   ?artist owl:sameAs? <" + resourceURI + "> .\n"
+ "   ?artist rdfs:label ?name .\n"
+ "   OPTIONAL { ?artist owl:birthDate ?birthDate } \n"
+ "   OPTIONAL { ?artist owl:deathDate ?deathDate } \n"
+ "   OPTIONAL { ?artist owl:birthPlace ?birthPlace } \n"
+ "   OPTIONAL { ?birthPlace rdfs:label ?birthPlaceStr }\n"
+ "   OPTIONAL { ?birthPlace prop:country ?country } \n"
+ "   OPTIONAL { ?artist owl:deathPlace ?deathPlace } \n"
+ "   OPTIONAL { ?deathPlace rdfs:label ?deathPlaceStr }\n"
+ "   OPTIONAL { ?artist owl:thumbnail ?image } \n"
+ "   OPTIONAL { ?artist prop:shortDescription ?shortDescription } \n"
+ "   OPTIONAL { ?artist owl:abstract ?abstract } \n"
+ "   OPTIONAL { ?artist prop:website ?website } \n"
+ "   OPTIONAL { ?artist prop:yearsActive ?yearsActive } \n"
+ "   FILTER (langMatches(lang(?name), 'en') \n"
+ "     && langMatches(lang(?birthPlaceStr), 'en') \n"
+ "     && langMatches(lang(?abstract), 'en')) \n"
+ " } LIMIT 1\n";

Query query = QueryFactory.create(queryString);

QueryExecution qe = QueryExecutionFactory.sparqlService(serviceURL, query);

try {
    ResultSet res = qe.execSelect();

    // should have only one row
    while (res.hasNext()) {

        QuerySolution row = res.next();

        artist = new Artist();

        artist.setResource(ParseUtils.parseXMLResource(row.get("artist").asResource()));
        artist.setAbstractStr(ParseUtils.parseXMLString(row.get("abstract")));
        artist.setBirthDate(ParseUtils.parseXMLDate(row.get("birthDate")));
        artist.setDeathDate(ParseUtils.parseXMLDate(row.get("deathDate")));
        artist.setCountry(ParseUtils.parseXMLString(row.get("country")));

        if (row.get("image") != null) {
            artist.setImageURL(ParseUtils.parseXMLURL(row.get("image")).toString());
        }

        artist.setBirthPlace(ParseUtils.parseXMLResource(row.get("birthPlace")));
        artist.setDeathPlace(ParseUtils.parseXMLResource(row.get("deathPlace")));
        artist.setBirthPlaceStr(ParseUtils.parseXMLString(row.get("birthPlaceStr")));
        artist.setDeathPlaceStr(ParseUtils.parseXMLString(row.get("deathPlaceStr")));
        artist.setName(ParseUtils.parseXMLDate(row.get("name")));
        artist.setShortDescription(ParseUtils.parseXMLString(row.get("shortDescription")));
        artist.setYearsActive(ParseUtils.parseXMLString(row.get("yearsActive")));
        artist.setWebsite(ParseUtils.parseXMLURL(row.get("website")));

        Logger.log(Level.INFO, "Artist found : " + artist.toString());
    }
}
```

- 1) La construction de la requête,
- 2) Attention au filtrage sur les langues, afin d'éviter les doublons,
- 3) Exécution de la requête,
- 4) Obtention des données issues du resultset,
- 5) J'ai fait des fonctions utilitaires de filtrage, car les données reçues ne sont pas toujours correctement typée et parfois même complètement fausse :

```

public static String parseXmlDate(RDFNode node) {
    String toReturn = "";
    try {
        if (node != null && node.isLiteral()) {
            toReturn = node.asLiteral().getValue().toString();
        }
    } catch (DatatypeFormatException e) {}

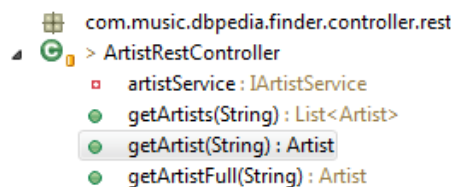
    logger.log(Level.WARNING, "Error on date field parsing : " + e.getMessage());
    try {
        return node.toString();
    } catch (Exception e1) {
        logger.log(Level.WARNING, "Error on date field parsing : " + e1.getMessage());
        return toReturn;
    }
}
return toReturn;
}

```

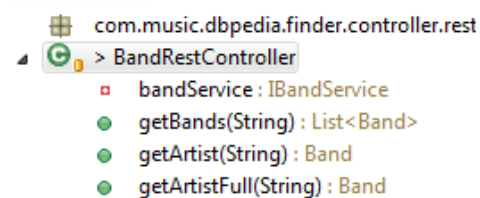
5.7 Controller REST

Les contrôleurs fournissent la donnée sous forme de service REST au frontend AngularJS.

Nous avons ceci pour un artiste :



Et pour un band :



Voici en détail le contenu pour BandRestController :

```

13 /**
14  * Band main REST controller
15  *
16  * @author speter
17  *
18  */
19 @RestController
20 @RequestMapping("/rest") 1
21 public class BandRestController {
22
23     @Autowired
24     private IBandService bandService; 2
25
26     @RequestMapping("/bands")
27     public List<Band> getBands(@RequestParam(value = "name",required = false) String name) {
28         List<Band> l = bandService.findByName(name);
29         return l;
30     }
31
32     @RequestMapping("/band")
33     public Band getArtist(@RequestParam(value = "uri",required = false) String uri) {
34         Band artist = bandService.getBandDetails(uri);
35         return artist;
36     }
37
38     @RequestMapping("/bandfull")
39     public Band getArtistFull(@RequestParam(value = "uri",required = false) String uri) {
40         Band artist = bandService.getBandFullDetails(uri);
41         return artist;
42     }
43 }
44

```

- 1) Définition par annotation du controller REST ; les services seront disponibles par l'URL relative au contexte de l'application par /rest.
- 2) BandService est injecté par la mécanique Spring (@Autowired),
- 3) Les services REST sont ensuite définits :
 - a. /bands pour obtenir la liste des bands ; utilisation d'un paramètre Get pour le champ name.

5.8 Tests

Nous sommes au bout de la création des services REST. Nous pouvons créer des classes de test pour la couche Service comme ceci :

```

@Test
public void testGetDetails() {
    Artist artist = null;

    List<Artist> artists = artisteService.findByName("knopfler");
    if (artists != null && artists.size() > 0) {
        artist = artists.get(0);
    }

    if (artist != null && artist.getResource() != null) {
        Artist artistDetail = artisteService.getArtistFullDetails(artist.getArtistURI());
        logger.info("Name is " + artistDetail.getName() + " associated artists found : " + artistDetail.getAssociatedArtists().size());
        logger.info("Name is " + artistDetail.getName() + " associated bands found : " + artistDetail.getAssociatedBands().size());

        assertEquals("Name must be the same. (" + artist.getName() + "/"
            + artistDetail.getName() + ")", artist.getName().equals(artistDetail.getName()), true);

        assertEquals("Must be at least one associated artist. (Found " + artistDetail.getAssociatedArtists().size()
            + " record.)", artistDetail.getAssociatedArtists().size() > 0, true);

        assertEquals("Must be at least one associated band. (Found " + artistDetail.getAssociatedBands().size()
            + " record.)", artistDetail.getAssociatedBands().size() > 0, true);

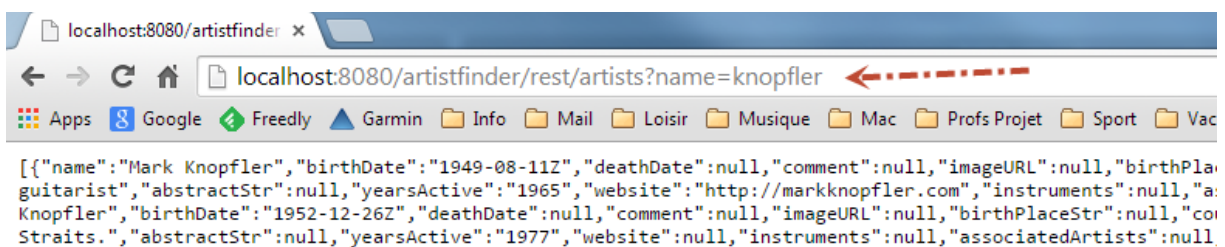
        assertEquals("Must be at least one genre. (Found " + artistDetail.getGenres().size()
            + " record.)", artistDetail.getGenres().size() > 0, true);
    }
}

```

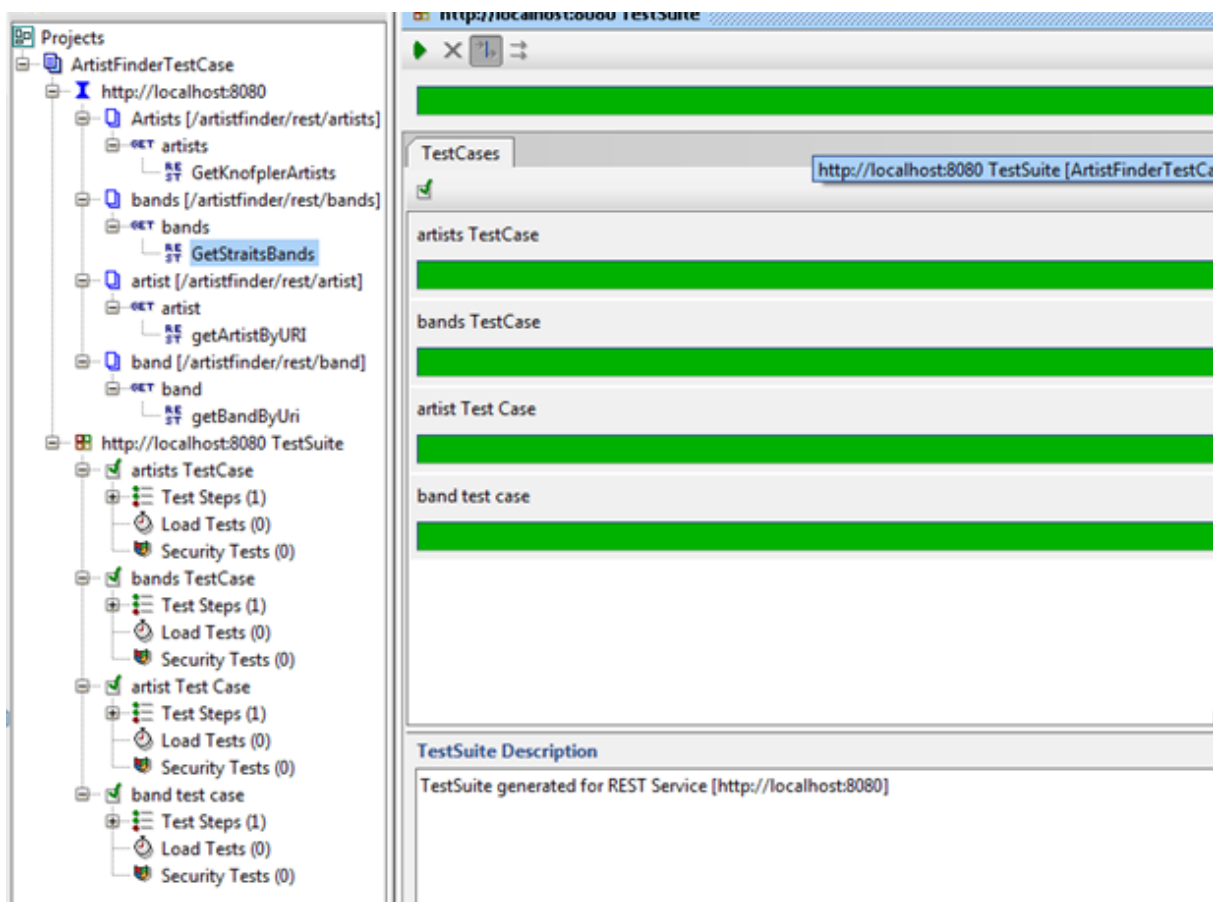
Et lancer les tests :



Nous pouvons aussi contrôler que les services REST fonctionnent correctement



Ou en utilisant SoapUI qui permet d'automatiser des tests sur les web services. Voici un exemple d'utilisation :



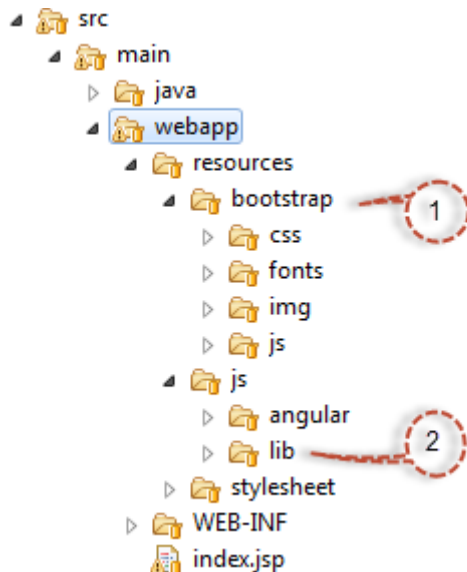
5.9 Frontend

Le frontend est une application JavaScript faite en utilisant le Framework AngularJS et les composants graphiques Twitter Bootstrap.

Mais avant tout il faut télécharger et importer les divers fichiers librairies JS et CSS dans l'application.

- 1) Bootstrap : <http://getbootstrap.com/getting-started/#download>
- 2) AngularJS : <https://angularjs.org/>

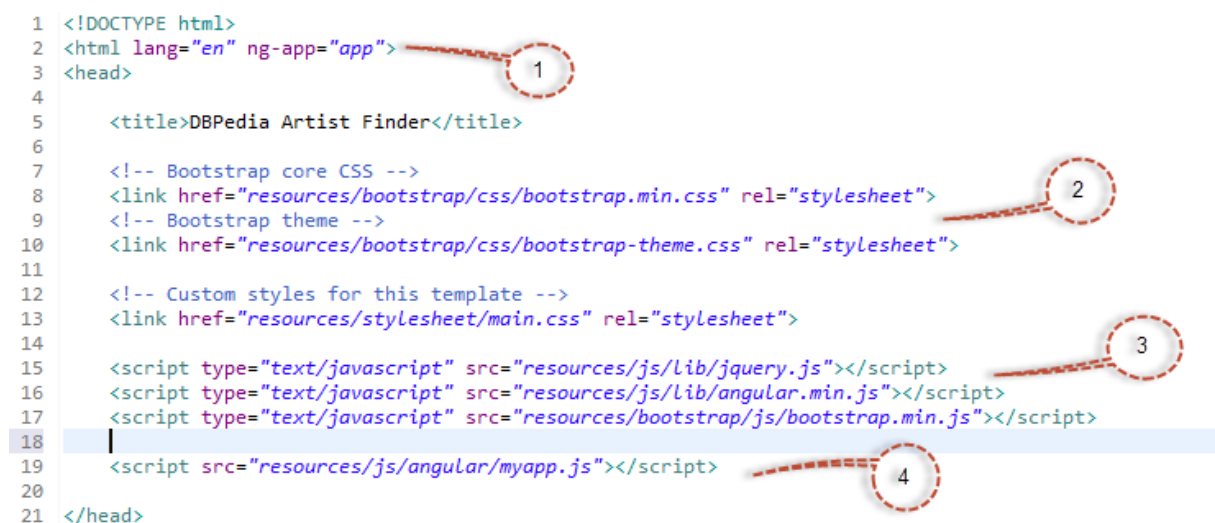
Il faut ensuite les rendre disponibles pour la partie webapp de l'application :



- 1) Librairies bootstrap,
- 2) Librairies angular.

5.10 Bootstrapping et Templating

Nous allons créer une page d'index qui contiendra le code HTML de l'application.



- 1) Signal de chargement du Framework AngularJS (Bootstrapping), le DOM, les directives, les binding sont chargés,
- 2) Feuilles de style Twitter Bootstrap, ainsi que celle de l'application,
- 3) Les scripts jquery.js, angular.js et bootstrap.js,
- 4) Le fichier contenant les sources de notre application (du côté JavaScript).

AngularJS est ainsi un MVC disposé dans le Browser. Afin de finaliser le chargement d'AngularJS nous créons un contrôleur, chargé dans la page HTML de cette façon :

```
<body ng-controller="artistSearchController">...
```

Code mettant ainsi en correspondance le code JavaScript suivant :

```
1 var app = angular.module('app', []);
2
3 /**
4  * Main application controller
5  */
6 app.controller('artistSearchController', function($scope, $http) {
```

Nous remarquons l'utilisation de la variable `$scope`, permettant de lier les données depuis le contrôleur jusqu'à la vue de manière bidirectionnelle, par exemple la variable `$scope.artists` (tableaux de valeurs) sera initialisée dans le contrôleur AngularJS comme ceci :

```
16 $scope.pages = []; // page list
17 $scope.artists; // artist list (results)
18 $scope.bands; // band list
19 $scope.loading=false; // manage loading info
20
21
22
23
24
25
26
27
28
29 $http.get("rest/artists?name=" + searchName).success(
30     function(response) {
31
32         $scope.artist = null;
33         $scope.artists = response;
34
35         console.log("Nb artist found " + $scope.artists.length);
36         $scope.searchName = searchName;
```

Et mise à disposition dans la vue de cette façon :

```
72<tr ng-repeat="artist in artists">
73    <td>{{ artist.name }} (<a ng-href="{{artist.artistURI}}">URI</a></td>
74    <td>{{ artist.shortDescription }}</td>
75    <td>{{ artist.yearsActive }}</td>
76    <td><a ng-href="{{ artist.website }}" target=" blank" />{{ artist.website }}</a></td>
77    <td><button class="btn btn-default" type="button" ng-click="moreInfo(artist.artistURI)">Mo
78</tr>
```

En utilisant la directive `ng-repeat` permettant de boucler sur les valeurs du tableau.

Remarquons aussi cette instruction `{{...}}` qui permet de reprendre et d'afficher les valeurs des variables.

```
<td>{{artist.shortDescription}}</td>
```

5.11 Construction de la vue

L'application permet de faire des recherches sur des noms d'artiste ou de groupe. En réponse donc à la requête de recherche, deux listes apparaissent :

- 1) Artiste
- 2) Groupes

Pour autant que des occurrences existent pour l'un et l'autre.

En voici un exemple :

DBPedia Artist Finder
Search: knopfler 2
Search: van halen 4
Sources hosted on Github

Search

Artist Results

Name	Description	Active Year	Web Site	More details
Alex Van Halen (URI)	American musician	1962	http://www.van-halen.com	<input type="button" value="More info"/>
Wolfgang Van Halen (URI)	American bass guitarist	2004	http://www.van-halen.com	<input type="button" value="More info"/>
Eddie Van Halen (URI)	Dutch-American rock musician	1971	http://van-halen.com/	<input type="button" value="More info"/>

Band Results

Name	Description	Active Year	More details
Van Halen (URI)	Van Halen is an American hard rock band formed in Pasadena, California, in 1972. From 1974 until 1985 the band comprised guitarist Eddie Van Halen, vocalist David Lee Roth, drummer Alex Van Halen and bassist Michael Anthony. This line-up was changed when David Lee Roth was replaced as vocalist by Sammy Hagar.	1972	<input type="button" value="More info"/>

Information © DBpedia
Application © Serge Peter 2015

Voici comment cela fonctionne :

- 1) Le champ recherche aliment la variable : `searchName` du model `$scope`,

```

49<div class="input-group">
50<input type="text" class="form-control" ng-model="searchName" placeholder="Search for..."> <span class="input-group-btn">
51<button class="btn btn-default" type="submit" ng-click="searchArtist(searchName)">Go!</button>
52</span>

```

- 2) La fonction `searchArtist(searchName)` est exécutée dans le contrôleur :

```

21  /**
22   * search for artist by name
23   */
24  $scope.searchArtist = function(searchName) {
25      console.log("Search for artist " + searchName);
26
27      $scope.loading=true;
28
29      $http.get("rest/artists?name=" + searchName).success(
30          function(response) {
31
32              $scope.artist = null;
33              $scope.artists = response;
34
35              console.log("Nb artist found " + $scope.artists.length);
36              $scope.searchName = searchName;
37
38              $http.get("rest/bands?name=" + searchName).success(
39                  function(response) {
40
41                      $scope.band = null;
42                      $scope.bands = response;
43
44                      $scope.page = {
45                          label : "Search: " + searchName,
46                          type : "searchArtist",
47                          what : searchName,
48                          id : $scope.pages.id + 1,
49                          length : $scope.artists.length + $scope.bands.length
50                      }
51                      $scope.addPage($scope.page);
52                      console.log("Nb band found " + $scope.bands.length);
53                  });
54
55              $scope.loading=false;
56
57          });
58  }
59

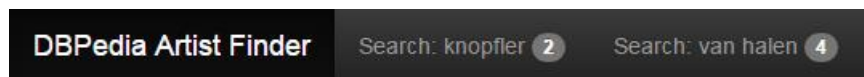
```

La fonction searchArtist prend en argument la chaîne de caractères searchName et fera deux choses :

- 1) Recherche des artistes correspondants en utilisant le service REST **rest/artists?name...**
- 2) Rechercher des groupes dont le nom correspond en utilisant le service RESR **rest/bands?name...**

Les variables du \$scope.artists et \$scope.bands sont alimentées en fonction des données en retours.

Une variable \$scope.page sera initialisée et ajoutée au tableau \$scope.pages, ceci afin de permettre l'affichage d'onglets en haut de la page :



(Ceci est fait de manière assez basique mais est utile si on veut revenir sur une recherche antérieure).

5.12 Vues de détails


Artist Results

Name	Description	Active Year	Web Site	More details
Alex Van Halen (URI)	American musician	1962	http://www.van-halen.com	More info
Wolfgang Van Halen (URI)	American bass guitarist	2004	http://www.van-halen.com	More info
Eddie Van Halen (URI)	Dutch-American rock musician	1971	http://van-halen.com/	More info


Band Results

Name	Description	Active Year	More details
Van Halen (URI)	Van Halen is an American hard rock band formed in Pasadena, California, in 1972. From 1974 until 1985 the band comprised guitarist Eddie Van Halen, vocalist David Lee Roth, drummer Alex Van Halen and bassist Michael Anthony. This line-up was changed when David Lee Roth was replaced as vocalist by Sammy Hagar.	1972	More info

Artist Detailed Info

Name	Eddie Van Halen (URI)
Short Description	Dutch-American rock musician
Birth Date	1955-01-25Z
Birth Place	Netherlands ()
Image	
Abstract	Edward Lodewijk "Eddie" Van Halen (born January 26, 1955 in Nijmegen) is a Dutch-born American musician, songwriter and producer best known as the lead guitarist, keyboardist and co-founder of the hard rock band Van Halen. He is ranked as one of the world's great guitarists, and one of the most influential rock guitarists of the 20th century. In 2011, Rolling Stone magazine ranked Van Halen #8 in its 100 Greatest Guitarists. In 2012, he was voted in a Guitar World magazine reader's poll as the #1 of "The 100 Greatest Guitarists" over Brian May of Queen (#2) and Alex Lifeson of Rush (#3).
Year Active	1971
Main Role	Musician, Guitarist, Producer


Band Detailed Info

Name	Van Halen (URI)
Full Description	Van Halen is an American hard rock band formed in Pasadena, California, in 1972. From 1974 until 1985 the band comprised guitarist Eddie Van Halen, vocalist David Lee Roth, drummer Alex Van Halen and bassist Michael Anthony. This line-up was changed when David Lee Roth was replaced as vocalist by Sammy Hagar. Critics and fans alike consider their 1975 self-titled debut album Van Halen to be one of the most "original and revolutionary" albums to change rock and roll. The band went on to further success, and by the early 1980s they were one of the most successful rock acts of the time. 1984 was their most successful album. The lead single, "Jump", became an international hit and their only single to reach number one on the Billboard Hot 100. The following singles, "Painkiller" and "1.9.85", both hit number 13 on the US charts. The album went on to sell over 12 million copies in the US alone. In 1985, the band replaced lead singer David Lee Roth with ex-Axis modex lead vocalist Sammy Hagar. With Hagar, the group would release four US number-one albums over the course of 11 years. Hagar left the band in 1996 shortly before the release of the band's first greatest hits collection, Best Of... Volume I. Ex-Estreme frontman Gary Cherone was quickly recruited as lead singer to replace Hagar, and Van Halen III was released in 1998. Cherone left the band in frustration in 1999 after the tour due to the poor commercial performance of the album. Van Halen went on hiatus until 2003 when they reunited with Hagar for a worldwide tour. The reunited band released a second greatest hits collection the following year, The Best of Both Worlds, Like Volume 1 before 6. The Best of Both Worlds included material from both the Roth and Hagar eras but omitted any Cherone era tracks. The album featured three brand new tracks recorded by the reunited band, two of which were released as singles. Hagar again left Van Halen in 2005 and in 2006, Roth returned as lead vocalist for their highest-grossing tour, and one of the highest-grossing tours of that year. Anthony was not invited to participate in the tour and was essentially fired from the band, replaced by Wolfgang Van Halen, Eddie's son. In 2012, the band released the commercially and critically successful A Different Kind of Truth, with Roth as lead vocalist. According to the RIAA, Van Halen is the 13th best-selling band/artist in United States history, selling 56 million albums in the U.S. and 96.5 million albums worldwide. They were also revealed at number 4 on the Billboard's top money makers list in 2013. Van Halen is one of only five rock bands that have had two studio albums sell more than 10 million copies in the U.S. Additionally, Van Halen charted the most number-one hits in the history of Billboard's Mainstream Rock chart: Van Halen achieved worldwide fame for their many popular songs and larger-than-life stage performances; unfortunately, they also became known for the drama surrounding the departures of former members. Controversy surrounded the band following the exits of Roth, Hagar, and Anthony; this controversy often included numerous conflicting press statements between the former members and the band. In 2007, Van Halen was inducted into the Rock and Roll Hall of Fame. VH1 ranked them 7th on their list of the top 100 hard rock artists of all time.
Home Town	Pasadena, California ()
Image	

J'ai construits ensuite deux vues détaillées pour Artist et Band. Afin d'accélérer le chargement des pages, j'ai décomposé les appels en deux (bien qu'une décomposition plus détaillée devraient être faite) :

- Obtention des informations basiques
 - service REST **rest/artist?uri=...** et
 - rest/band?uri=...**
- Obtention des informations détaillées, nécessitant des requêtes supplémentaire du côté backend (et donc DBpedia) :
 - service REST **rest/artistfull?uri=...** et
 - rest/bandfull?uri=...**

Artist Detailed Info

Name	Eddie Van Halen (URI)
Short Description	Dutch-American rock musician
Birth Date	1955-01-25Z
Birth Place	Netherlands ()
Image	 <p>Informations directes</p>
Abstract	Edward Lodewijk "Eddie" Van Halen (born January 26, 1955 in Nijmegen) is a Dutch-born American musician, songwriter and known as the lead guitarist, keyboardist and co-founder of the hard rock band Van Halen. He is ranked as one of the world's one of the most influential rock guitarists of the 20th century. In 2011, Rolling Stone magazine ranked Van Halen #8 in the Guitarists. In 2012, he was voted in a Guitar World magazine reader's poll as the #1 of "The 100 Greatest Guitarists of All Time" (Queen (#2) and Alex Lifeson of Rush (#3)).
Year Active	1971
Web Site	http://van-halen.com/
Genres	<ul style="list-style-type: none"> Heavy metal music Hard rock
Instruments	<ul style="list-style-type: none"> Guitar Charvel Bass guitar Singing Kramer Guitars Keyboard instrument Frankenstrat Ibanez Peavey EVH Wolfgang Drum kit <p>Informations détaillées</p>
Associated Artists	<ul style="list-style-type: none"> Brian May (URI) LL Cool J (URI) Sammy Hagar (URI) Steve Vai (URI)
Associated Bands	<ul style="list-style-type: none"> Van Halen (URI)

Note :

Si vous tester l'application, vous noterez que les performances de DBPedia sont relativement faibles, ce qui se comprend aisément car la base de données est distante et très vaste. Une indication à l'écran vous permet de voir que les requêtes sont en cours d'exécution ou terminée.

⌂ Loading...

5.13 Code JavaScript

Le chargement des détails Artist et Band se font de la même manière. En voici un exemple

pour Band :

```

98  /**
99  * get band info
100 */
101 $scope.bandInfo = function(bandURI) {
102
103     console.log("Get more info for : " + bandURI);
104
105     $scope.loading=true;
106
107     $http.get("rest/band?uri=" + bandURI).success(function(response) {
108
109         $scope.band = response;
110
111         console.log("Band " + $scope.band.name);
112
113         $scope.page = {
114             label : "Info : " + $scope.band.name,
115             type : "band",
116             what : bandURI,
117             id : $scope.pages.id + 1
118         };
119
120         $scope.addPage($scope.page);
121
122         // second call for performance issue
123         $http.get("rest/bandfull?uri=" + bandURI).success(function(response) {
124
125             $scope.band = response;
126             console.log("Band full " + $scope.band.name);
127
128             $scope.loading=false;
129
130         });
131     });
132
133 }
134

```

Ici aussi il y a deux appels aux services rest ; band et bandfull. Ceci pour des nécessités de fluidité dans le chargement de la page.

La variable \$scope.band est ainsi enrichi, et mise à disposition pour la vue.

La variable \$scope.page est aussi mise à jour afin de faciliter la navigation dans l'application.

Note :

Ceci est fait de manière un peu basique et nécessiterait un découpage plus précis des informations mise à jour et peut-être plus d'appel au backend fluidifier le chargement de la page.

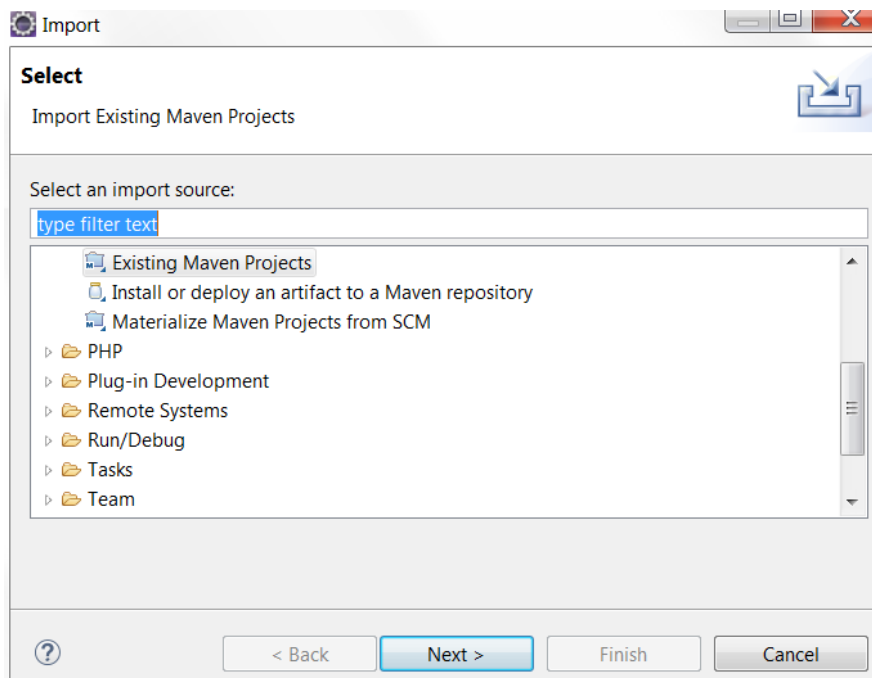
6. Exécution de l'application

Il y a plusieurs possibilités pour exécuter cette application. Attention de faire des recherches d'artistes anglophone car je suis connecter à la source de données DBpedia par défaut.

6.1 Directement dans Eclipse

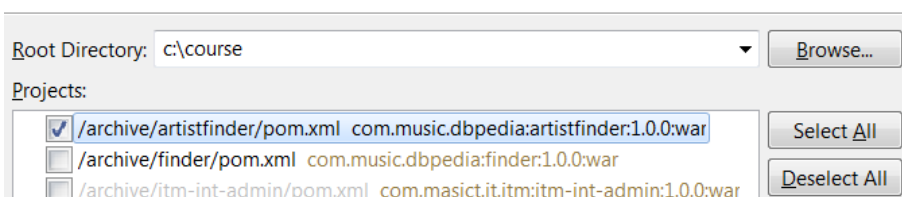
Une fois toutes les opérations du chapitre « Mise en route », chapitre 4 faites, il reste à faire ceci :

- 1) Mettre les sources sur le disque local,
- 2) Importer le projet en tant que projet Maven,



Maven Projects

Select Maven projects



- 3) Installer l'application sur le serveur Tomcat intégré et installé dans Eclipse,

Add and Remove

Modify the resources that are configured on the server



Move resources to the right to configure them on the server

Available:

itm-int-admin
TD_JavaEE
TD_MAS_REST

Add >

< Remove

Configured:

artistfinder(finder)

- 4) Ouvrir l'application sous : <http://localhost:8080/artistfinder> (8080 étant le port définit pour l'instance de Tomcat).

6.2 En mode ligne de commande avec Maven

Là aussi il faut faire l'installation des logiciels prérequis, notamment Maven. Une fois les sources chargées sur le disque local, il faut exécuter depuis le répertoire du projet la commande

```
mvn clean tomcat7:run
```

6.3 Sur le site Openshift

J'ai installé l'application Artist Finder sur le site Openshift de Redhat.

<http://artistfinder-spe.rhcloud.com/>

Note : il se peut qu'Openshift mette en veille l'application si elle n'est pas utilisée. Il faut donc attendre un peu et recharger l'application si elle ne s'ouvre pas directement.

7. Conclusion

Dans le cadre de ce travail mon but était d'explorer et d'utiliser les données issues de DBPedia et de faire un outil de recherche afin de mettre en correspondance les informations d'artistes et de groupes. J'ai pu apprendre à construire des requêtes sparql basées sur les triples, en ajouter des informations de filtrage et des sous-requêtes.

Il est à noter qu'il faut faire attention aux typages des données reçues, car ces données comportent des erreurs (par exemple sur des formats de dates). J'ai construit quelques fonctions utilitaires permettant de récupérer les données même en cas d'erreurs.

Il est intéressant d'utiliser les types fournis par Jena comme par exemple `Ressource`, qui permet ensuite d'être réutilisé par la suite en tant que type pour faire des requêtes plus détaillées ; par exemple, obtention des détails d'une ressource.

Bien que cantonné dans ce travail à l'ontologie « `MusicalArtist` » de DBPedia, la quantité et variété de source de données à disposition est vraiment très importante et le système de requête sparql adapté à la situation.

J'ai trouvé que Spring et la mise à disposition en tant que service REST des données pratiques et faciles d'accès. Du fait que les requêtes sont faites sur des sources de données distantes, il est important de découper les choses afin de rester fluide. Chose que j'ai fait partiellement. Ceci d'autant plus que l'application front permet un chargement asynchrone des informations.

AngularJS est un bon Framework JavaScript qui fait particulièrement bien le binding des données issues du backend et l'affichage dans la vue. Une partie du code « business » est déporté dans le frontend bien qu'ici les choses restent assez simples. Je remarque donc que cette pratique de découplage est assez populaires et surement une tendance pour les années à venir.

Pour conclure, j'ai pris beaucoup de plaisir à découvrir ces technologies ; notamment sparql et Jena ainsi que AngularJS.

8. Références

Dbpedia :

- <http://dbpedia.org>

Sparql

- <https://jena.apache.org/tutorials/sparql.html>

Jena :

- <http://www.ibm.com/developerworks/xml/library/j-sparql/>
- <https://jena.apache.org/tutorials/index.html>

Livre sur AngularJS

- AngularJS de Brad Green, Shyam Seshadri, publié par O'Reilly en avril 2013; ISBN:978-1-4493-4485-6
- Mastering Web Application Development with AngularJS de Pawel Kozlowski et Peter Bacon Darwin, publié par Packt Publishing en Août 2013; ISBN 978-1-78216-182-0
- <https://docs.angularjs.org/tutorial>

Spring

- <http://projects.spring.io/spring-data>
- <http://www.programming-free.com/2014/07/spring-data-rest-with-angularjs-crud.html>
- <http://docs.spring.io/spring/docs/current/spring-framework-reference/html>