|  |
| --- |
|  |
| Rapport - Sujet B3 : Requêtes et visualisation de données issues de Dbpedia |
|  |
|  |
| **Peter Serge** |
| **9/1/2015** |

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 2](#_Toc412107189)

[1.1 Rappel sur les objectifs 2](#_Toc412107190)

[1.2 Rappel sur les contraintes techniques 2](#_Toc412107191)

[2. Analyse 2](#_Toc412107192)

[2.1 Liste des artistes (Query 1.1) 3](#_Toc412107193)

[2.2 Détail sur un artiste (Query 1.2) 4](#_Toc412107194)

[2.3 Recherche d’artistes ou bands liés (Query 1.3) 4](#_Toc412107195)

[2.4 Recherche des genres (Query 1.4) 5](#_Toc412107196)

[3. Architecture technique 5](#_Toc412107197)

[3.1 Composants 6](#_Toc412107198)

[3.2 Dbpedia et sparql 6](#_Toc412107199)

[3.3 Jena 6](#_Toc412107200)

[3.4 Spring 7](#_Toc412107201)

[3.5 Junit & Spring test 7](#_Toc412107202)

[3.6 AngularJS 7](#_Toc412107203)

[3.7 Maven 7](#_Toc412107204)

[4. Mise en route 7](#_Toc412107205)

[4.1 Installation de Java 7](#_Toc412107206)

[4.2 Installation de Maven 8](#_Toc412107207)

[4.3 Installation de Eclipse 8](#_Toc412107208)

[4.4 Installation de Tomcat 8](#_Toc412107209)

[5. Réalisation 10](#_Toc412107210)

[5.1 Backend 10](#_Toc412107211)

[5.2 Frontend 10](#_Toc412107212)

[6. Conclusion 10](#_Toc412107213)

[7. Références 11](#_Toc412107214)

# 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.

Notre 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.

## Rappel sur les objectifs

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

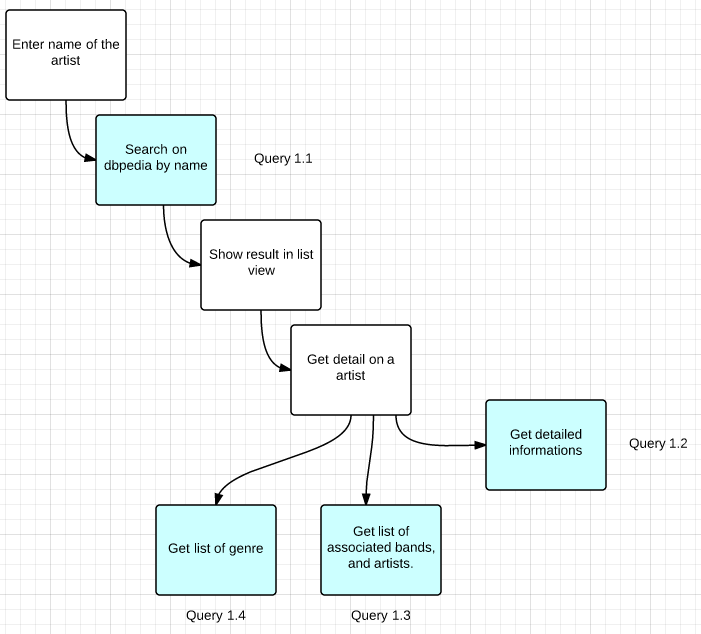
En partant d’un nom d’artiste, il faudra faire la liste des groupes musicaux liés, albums réalisés ainsi que des informations liées comme le style.

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

# Analyse

L’application obtiendra ces données via des requêtes sparql, faite 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 genre (Query 1.4) associés ainsi que les artistes (Query 1.2) et groupes liés (Query 1.3).

## 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#](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>



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'))

## 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 traité 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

## Recherche d’artistes ou bands liés (Query 1.3)

Là aussi la propriété sameAs permet de faire la sélection au niveau de la ressource. Ou 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#](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

## 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}

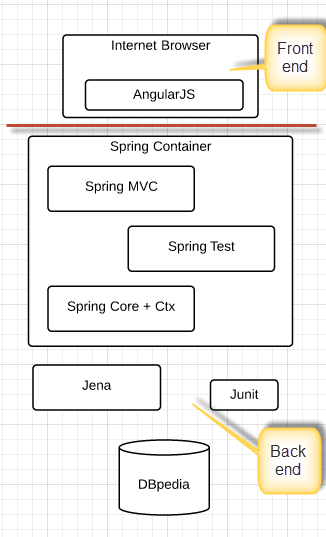
FILTER (langMatches(lang(?name), 'en')

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

} LIMIT 10

# 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.



## Composants

Voici une brève présentation de éléments composants notre architecture logicielle.

### 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.

### Jena

Jena est une librairie permettant d’exploiter des données issues de sources de données sémantique.

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 ResultSet java.

### Spring

Spring est un Framework assez populaire dans le monde des applications d’entreprise Java JEE. Le but du Framework est de simplifier au maximum la mise en route 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 permettre d’exposer 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.

### Junit & Spring test

Junit est utiliser pour réaliser des tests en plus de Spring test, qui nous permet de charger les services, contexte Spring et configuration.

### 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.

### Maven

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

# 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/> |

## 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.

## Installation de Maven

Maven est disponible ici :

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

Une fois le package [apache-maven-3.\*.\*-bin.zip](http://mirror.switch.ch/mirror/apache/dist/maven/maven-3/3.2.5/binaries/apache-maven-3.2.5-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.

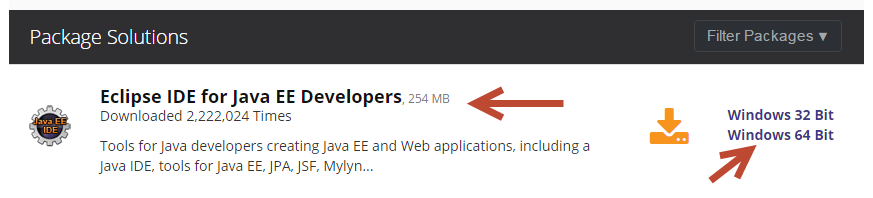
Plus d’info sont disponible ici : <http://maven.apache.org/settings.html>.

## Installation de Eclipse

Eclipse Luna est disponible ici :

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

Il faut ensuite faire attention de prendre la version EE :



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

## Installation de Tomcat

Tomcat 8 est disponible ici :

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

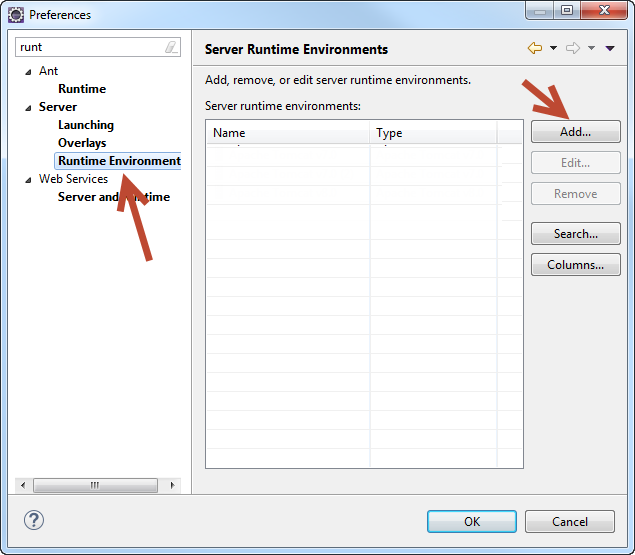
Une fois le package téléchargé, il faut le décompresser en local (par exemple sur c:\apps\tomcat8).

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 8** :

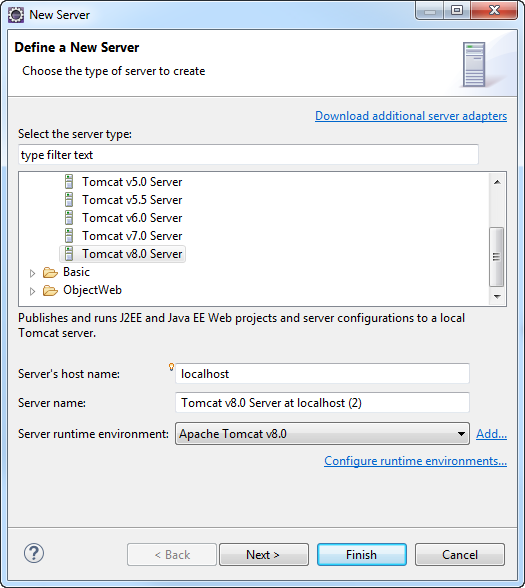
* Dans l’onglet Préférences de Eclipse, rechercher : runtime ; clicker sur Add…



* Définissez le répertoire Tomcat de travail, ainsi qu’un JRE, qui doit être préalablement définit dans Eclipse, ou le runtime par défaut.

**Ajouter un serveur Tomcat 8 :**

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



# Réalisation

## Backend

### Maven web app

## Frontend

# Conclusion

# Références

Dbpedia :

* http://dbpedia.org/About

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

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/