CERGY PARIS UNIVERSITÉ

Rapport de projet IED

Conception et réalisation d'un médiateur simple, exploitant différentes sources de données dans le domaine du cinéma.

Equipe du projet :

Amal ROUAS
Mourtada HOUARI

Encadré par :

Dan Vodislav
Clément AGRET

Remise du rapport : 27 Avril 2022

Table des matières

1. Introduction :

Dans le cadre d'un projet d'intégration de données, on a été amené à concevoir et implémenter un médiateur avec un langage au choix, durant une semaine qui englobe toutes les connaissances acquises au cours de notre formation, en exploitant 3 parties principales : Talend, SPARQL et Webservice.

2. Architecture globale du projet :

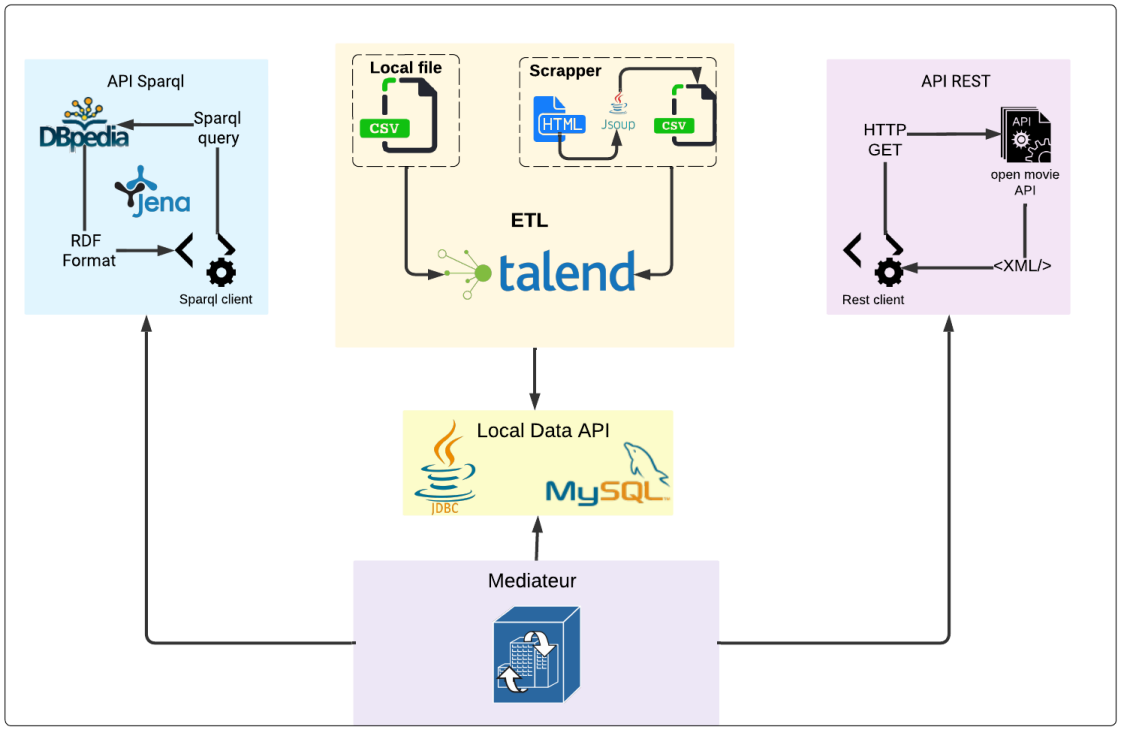


Figure 1 : Architecture Globale du projet

3. Mise en place des sources des données

3.1: Source locale (Mysql)

3.1.1: Web scraping:

Nous avons conçu un programme Java avec l'API Jsoup qui permet de récupérer les données qui figurent sur la table HTML depuis le site *the-numbers.com*.

Pour chaque genre de film, on a récupéré toutes les pages *HTML* qui correspondent aux films de ce genre et qui sont produits entre les années 2000 et 2015. Ces pages sont représentées par des objets de la classe *Document* qui expose une fonction *select()* permettant de récupérer un ou des élément de la page avec un sélecteur *CssClass* ou un *TagName*. Les éléments auxquels on s'intéressait sont les lignes de la table qui figure sur la page et qui ont le chemin suivant : *div#main -> div#page_filling_chart 2nd-> table -> tbody -> tr* .
On a transformé chaque ligne de la table *HTML* en une ligne *CSV* séparant chaque colonne avec un caractère tabulation.

Certains films ayant un titre incomplet sur la table nous ont obligés à faire une deuxième requête pour récupérer la page du film en question avec le lien contenu dans l'attribut *href* de la balise *<a/>* contenue dans la colonne du titre. Le titre figure à l'intérieur d'une balise *<h1/>* mais avec son année de sortie aussi, donc il a fallu traiter aussi ce problème en enlevant l'année.

```
Titre sur la table: The Flintstones in Viva Roc…
Titre sur la page du film: The Flintstones in Viva Rock Vegas (2000)
Titre après traitemnts: The Flintstones in Viva Rock Vegas
```

Figure 2 : Traitement des titres incomplets

A la fin du traitement on a produit un fichier *CSV* pour chaque genre, avec toutes les informations nécessaires pour le médiateur.

	Movie	Release Date	Theatrical Distributor	Gender
1	Mission: Impossible 2	2000-05-24	Paramount Pictures	Action
2	Gladiator	2000-05-05	Dreamworks SKG	Action
3	X-Men	2000-07-14	20th Century Fox	Action
4	Charlie's Angels	2000-11-03	Sony Pictures	Action
5	Gone in 60 Seconds	2000-06-09	Walt Disney	Action
6	U-571	2000-04-21	Universal	Action
7	Shaft	2000-06-16	Paramount Pictures	Action

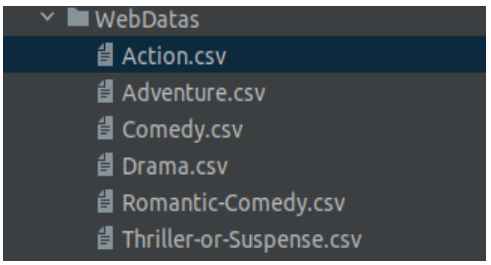


Figure 3 : Structure des fichiers de sortie

3.1.2: Alimentation de la base de données avec Talend :

Le fichier *movieBudgets.csv* qui nous a été fourni sur l'ent, contient les informations sur chaque film, à savoir le titre, la date de sortie, le budget, les revenus aux Etats Unis et les revenus mondiaux. Ces informations seront complétées en ajoutant le genre et le distributeur par la suite grâce à l'API Jsoup et transformées avec Talend pour alimenter la base de données.

Les images ci-dessous montrent les étapes qu'on a suivies pour l'intégration des données du fichier *movieBudgets.csv* et des *fichiers.csv* extraits des pages HTML afin de créer notre base de données locale sous l'ETL Talend.

Afin d'importer le fichier *movieBudgets.csv,* on a utilisé le composant Talend "tFileInputDelimited", pour lequel on a modifié le schéma, comme suivant :

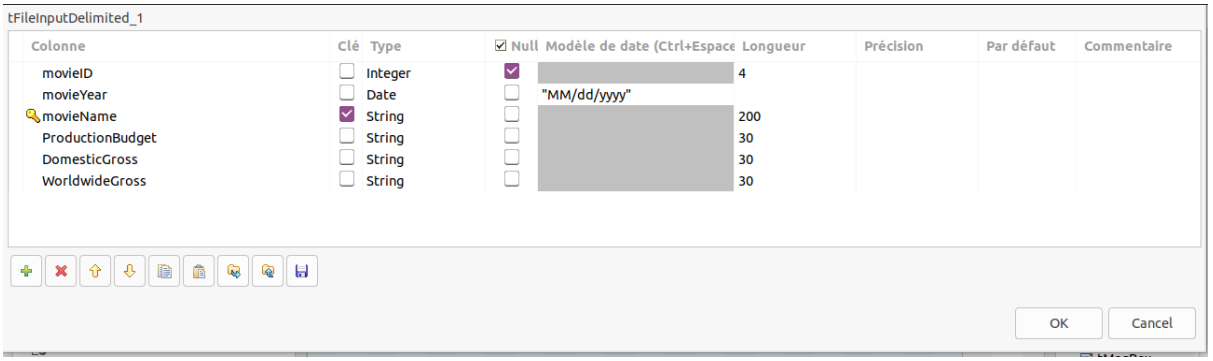


Figure 4 : Description du schéma du fichier *movieBudgets.csv* sous Talend

Pour intégrer les *fichiers.csv* extraits du web, on a utilisé le composant Talend "tFileList" qui nous permet de récupérer tous les fichiers. Pour pouvoir itérer sur la liste des fichiers, on a ajouté un composant "tFileInputDelimited" qu'on a lié à un composant "tUnite" qui produit un fichier csv englobant toutes les données de tous les *fichiers.csv*, comme le montre l'image ci-dessous :

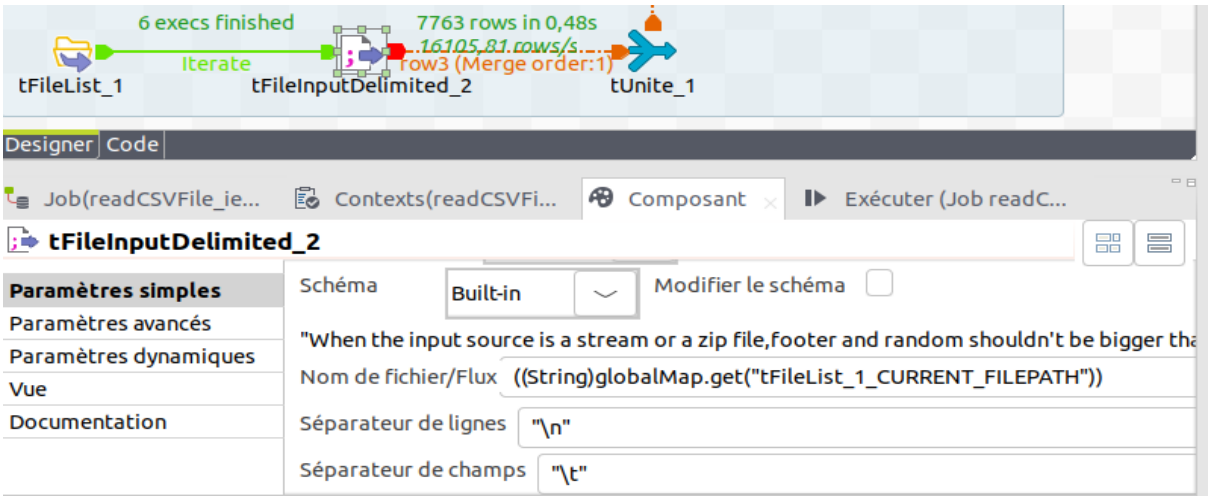


Figure 5 : Intégration des fichiers.*csv* extraits des pages HTML sous Talend

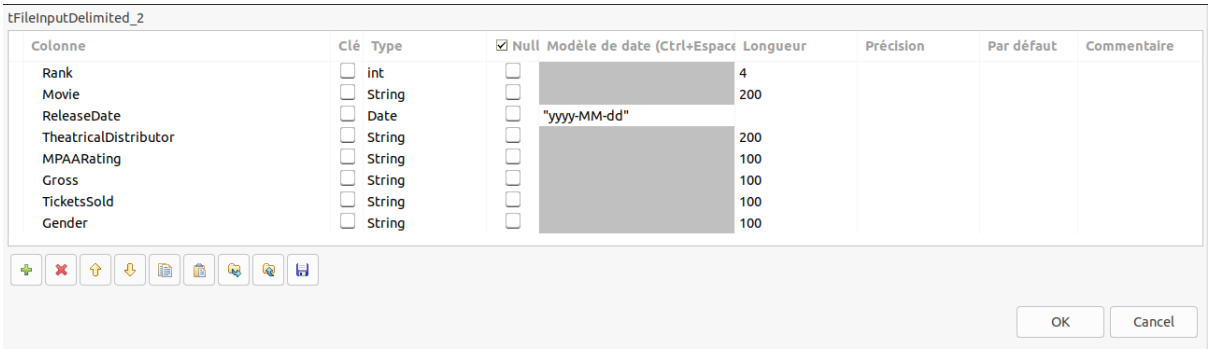


Figure 6 : Schéma des fichiers.*csv* extraits des pages HTML sous Talend

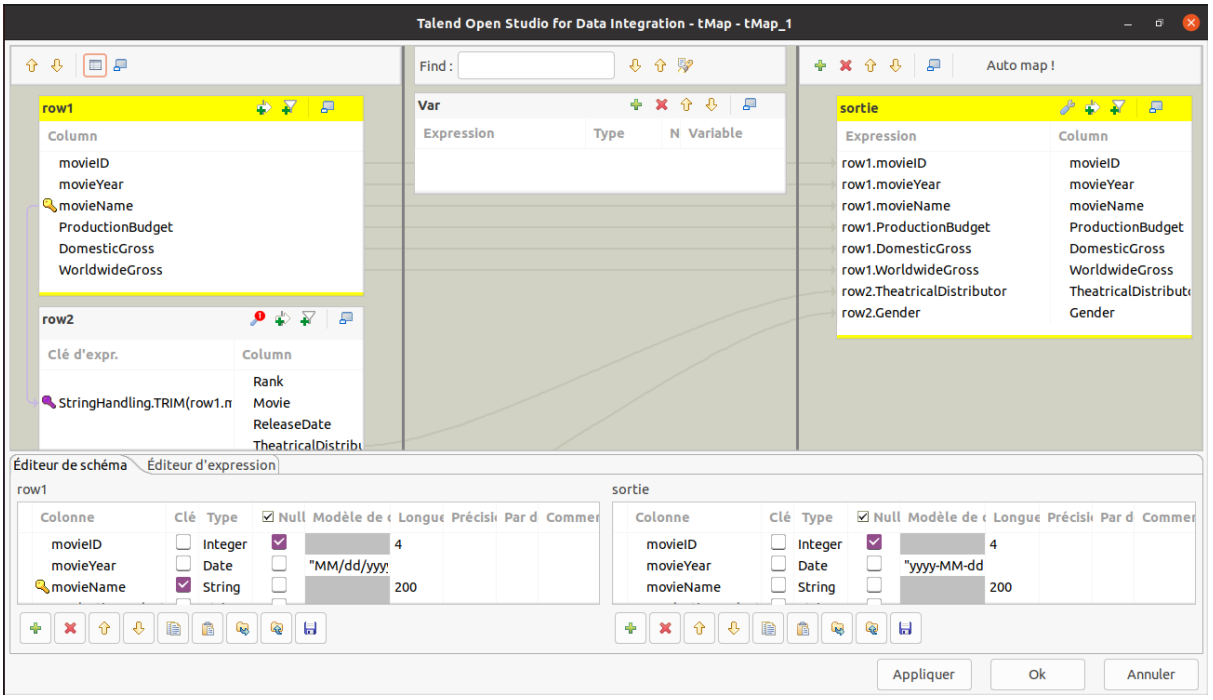


Figure 7 : Mapping des deux schémas décrits ci-dessus sous Talend

On a fait la jointure entre les deux schémas par le titre, et comme les titres extraits des pages HTML ont un formats particulier (un espace à la fin), on a utilisé la Routine “StringHandling.TRIM(row1.movieName)” pour y remédier. On a opté pour une type de jointure LeftJoin afin de récupérer tous les films même ceux dont les informations, genre et distributeur n’ont pas été renseignées.

A la suite de cette étape on à changé le format des dates du schéma de l’output pour l’adapter au format des dates dans notre base de données.

Pour la fin, afin de créer notre base de données, on a ajouté un composant “tDBOutputBulkExec” qu’on a configuré comme le montre l’image ci-dessous :

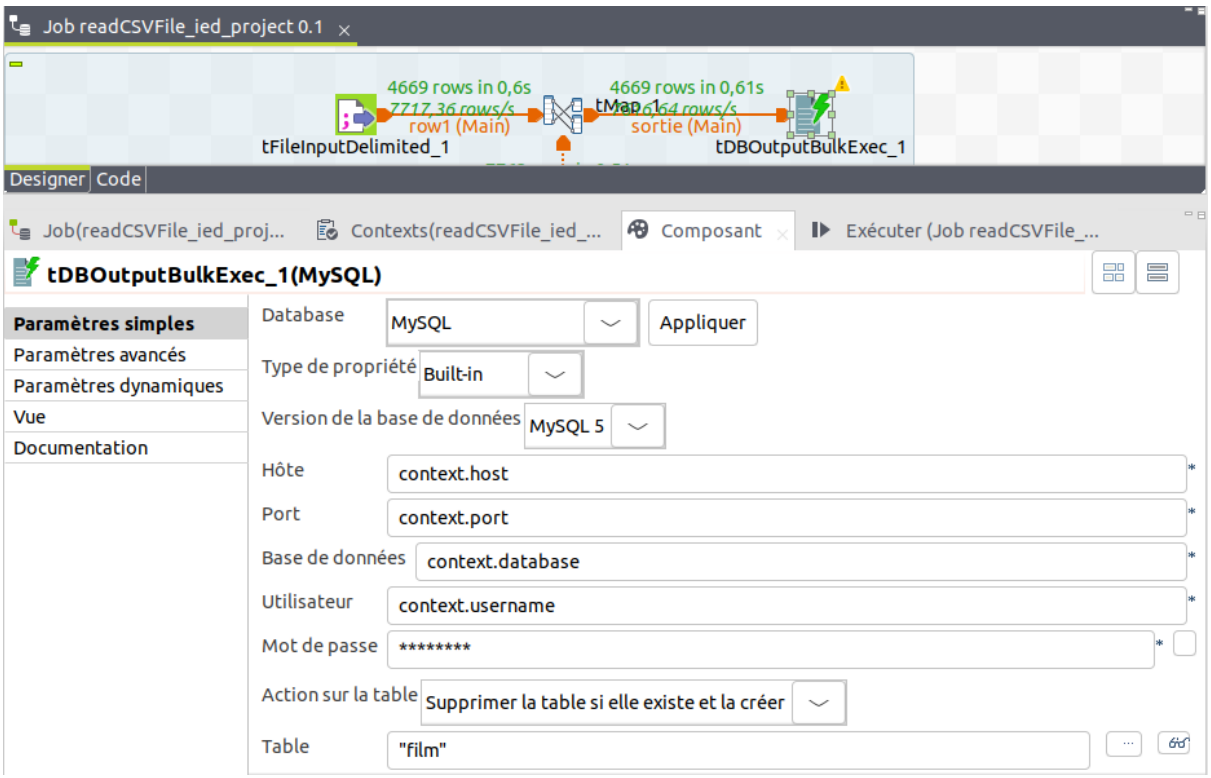


Figure 8 : Génération de table *film* dans la base de données sous Talend

Le job ci-dessous illustre toutes les étapes décrites précédemment :

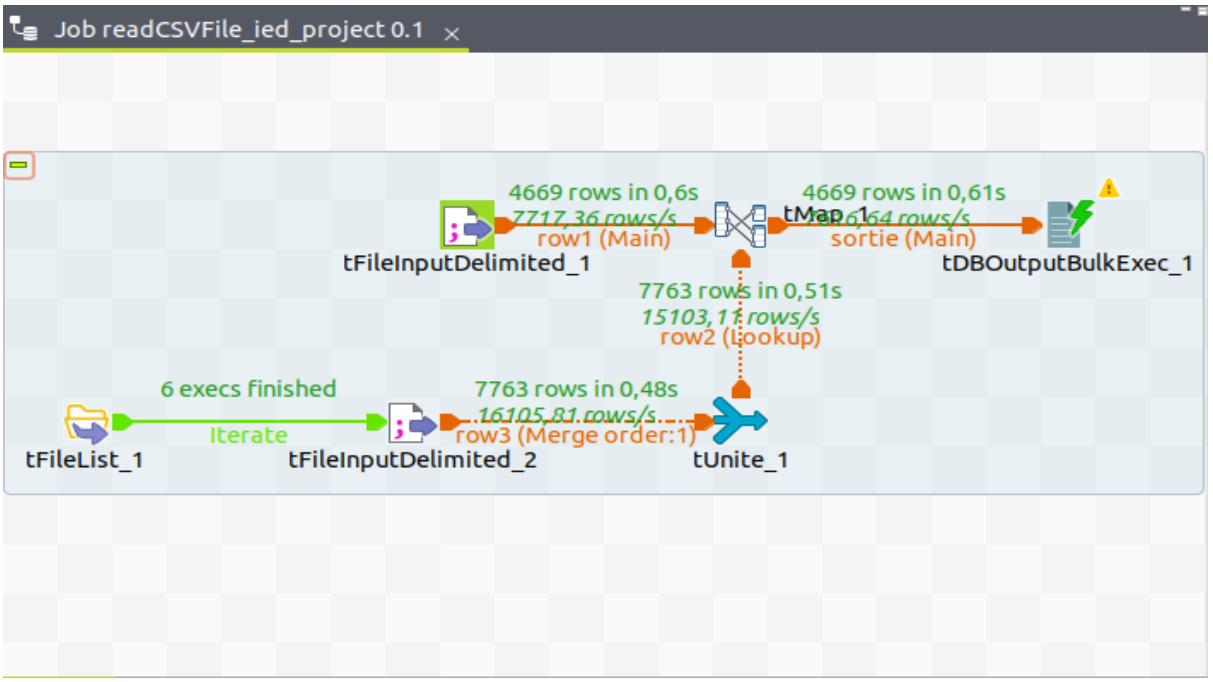


Figure 9 : Illustration du Job d’alimentation de la base de données sous Talend

A la fin de l'exécution, on aura la table *film* alimentée dans notre base de données.

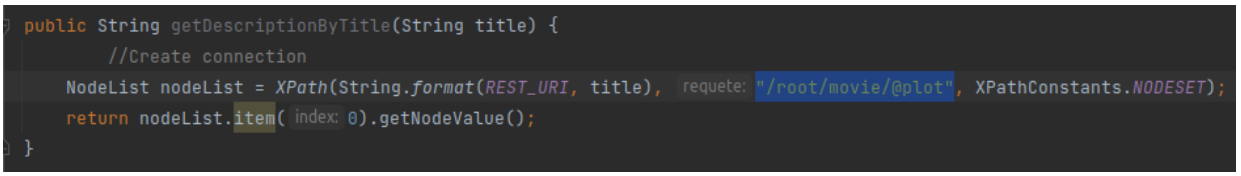
```
MariaDB [projet_ied]> select * from film;
+---------+---------------------+-------------------------------------------------------------------------------------------------+-----------------+
| movieID | movieYear           | movieName                                                                                       | ProductionBudget |
 DomesticGross | WorldwideGross | TheatricalDistributor    | Gender               |
+---------+---------------------+-------------------------------------------------------------------------------------------------+-----------------+
|       1 | 2009-12-18 00:00:00 | Avatar                                                                                          | $425,000,000     |
 $760,507,648  | $2,783,919,005 | 20th Century Fox         | Action               |
|       2 | 2007-05-24 00:00:00 | Pirates of the Caribbean: At World's End                                                        | $300,000,000     |
 $309,420,425  | $960,996,492   |                          |                      |
|       3 | 2012-07-20 00:00:00 | The Dark Knight Rises                                                                           | $275,000,000     |
 $448,139,104  | $1,079,343,948 | Warner Bros.             | Action               |
|       4 | 2013-07-02 00:00:00 | The Lone Ranger                                                                                 | $275,000,000     |
 $89,289,912   | $259,989,912   |                          |                      |
|       5 | 2012-03-09 00:00:00 | John Carter                                                                                     | $275,000,000     |
 $73,058,680   | $282,778,101   | Walt Disney              | Adventure            |
|       6 | 2010-11-24 00:00:00 | Tangled                                                                                         | $260,000,000     |
 $200,821,936  | $586,581,936   |                          |                      |
|       7 | 2007-05-04 00:00:00 | Spider-Man 3                                                                                    | $258,000,000     |
 $336,530,304  | $890,875,304   | Sony Pictures            | Adventure            |
|       8 | 2012-12-14 00:00:00 | The Hobbit: An Unexpected Journey                                                               | $250,000,000     |
 $303,003,584  | $1,014,703,584 |                          |                      |
|       9 | 2009-07-15 00:00:00 | Harry Potter and the Half-Blood Prince                                                          | $250,000,000     |
 $301,959,200  | $935,083,689   |                          |                      |
|      10 | 2013-12-13 00:00:00 | The Hobbit: The Desolation of Smaug                                                             | $250,000,000     |
 $258,366,855  | $950,466,855   |                          |                      |
```

Figure 10 : Affichage de la table *film* de notre base de données locale

3.2: Client Rest

Cette partie consiste à faire appel au service REST par un appel HTTP GET avec l'URI obtenue après une demande de Key sur le site http://www.omdbapi.com/:

Étant donné le titre d'un film, on retrouve sa description. La requête XPath qui permet d'obtenir cette information est la suivante :

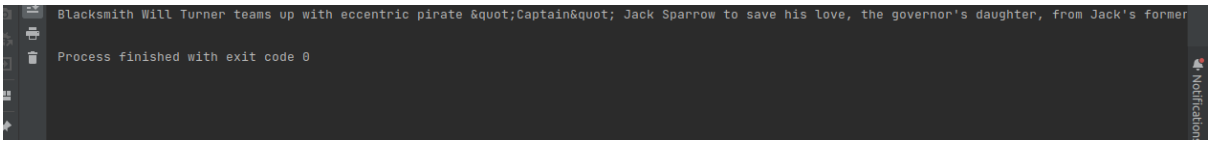


```
public String getDescriptionByTitle(String title) {
        //Create connection
    NodeList nodeList = XPath(String.format(REST_URI, title),  requete: "/root/movie/@plot", XPathConstants.NODESET);
    return nodeList.item( index: 0).getNodeValue();
}
```

Figure 11 : Requête XPath qui récupère l'attribut description de la balise film

Les résultats sont retournés sous forme d'une chaîne de caractère contenant la valeur du premier film retourné.

Le résultat de la requête est illustré dans l'image suivante:



```
Blacksmith Will Turner teams up with eccentric pirate &quot;Captain&quot; Jack Sparrow to save his love, the governor's daughter, from Jack's former

Process finished with exit code 0
```

Figure 12 : Résultat de la requête XPath

3.3: Client SPARQL

Cette partie de notre système peut interroger le point d'accès *SPARQL* de *DbPedia* et faire deux types de requêtes:

- Étant donné le titre d'un film, retrouver son réalisateur, son producteur ainsi que la liste des acteurs qui ont joué dans ce film. La requête qui permet de trouver ces informations est la suivante :

```
SELECT ?filmName ?directorName  ?actorName ?producerName WHERE {\n"    +
"                  ?film a dbo:Film ;\n"                                +
"                     dbp:name \"%s\"@en ;\n"                           +
"                     dbp:name ?filmName;\n"                            +
"                     dbo:director ?director ;\n"                       +
"                     dbo:producer ?producer;\n"                        +
"                     dbo:starring ?actor.\n"                           +
"                  ?director rdfs:label ?directorName.\n"               +
"                  ?producer rdfs:label ?producerName.\n"               +
"                  ?actor rdfs:label ?actorName.\n"                     +
"                  FILTER (langMatches(lang(?directorName),\"en\")).\n" +
"                  FILTER (langMatches(lang(?producerName),\"en\")).\n" +
"                  FILTER (langMatches(lang(?actorName),\"en\")).\n"    +
"                                                                     }
```

Figure 13 : Requête SPARQL qui permet de chercher un filme par son titre

Les résultats sont retournés sous forme d'un produits cartésien entre les producteurs, les réalisateurs et les acteurs et pour remédier à la redondance des informations, nous avons construits pour attribut chaque un Set contenant une liste d'éléments uniques et stocker le tout dans une HashMap ayant comme clés les nom d'attributs et comme valeur le set en question.

Le résultat de la requête est illustré dans l'image suivante:

SPARQL | HTML5 table

filmName	directorName	actorName	producerName
"Pirates of the Caribbean"@en	"Gore Verbinski"@en	"Geoffrey Rush"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean"@en	"Joachim Rønning"@en	"Geoffrey Rush"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean"@en	"Espen Sandberg"@en	"Geoffrey Rush"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean"@en	"Rob Marshall"@en	"Geoffrey Rush"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean"@en	"Rob Marshall"@en	"Keira Knightley"@en	"Jerry Bruckheimer"@en

Figure 14 : Résultat de la requête

- Etant donné un acteur, retrouver la liste des films dont il a joué:

La requête SPARQL suivante permet de répondre à cette question:

```
"SELECT ?film ?filmName ?directorName ?producerName WHERE {\n" +
" ?film a dbo:Film ;\n"                                        +
"   rdfs:label ?filmName;\n"                                   +
"   dbo:director ?director ;\n"                                +
"   dbo:producer ?producer;\n"                                 +
"   dbo:starring ?a .\n"                                       +
"?a foaf:name \"%s\"@en .\n"                                   +
" ?director rdfs:label ?directorName.\n"                       +
" ?producer rdfs:label ?producerName.\n"                       +
" FILTER (langMatches(lang(?filmName),\"en\")).\n"             +
" FILTER (langMatches(lang(?directorName),\"en\")).\n"         +
" FILTER (langMatches(lang(?producerName),\"en\")).\n"         +
"                                                            }"
```

Figure 15 : Requête SPARQL qui permet de chercher les filmes par acteur

Le résultat de la requête est illustré dans l'image suivante:

filmName	directorName	producerName
"De-Lovely"@en	"Irwin Winkler"@en	"Charles Winkler"@en
"Conspiracy (2001 film)"@en	"Frank Pierson"@en	"Frank Pierson"@en
"Pirates of the Caribbean: Dead Men Tell No Tales"@en	"Espen Sandberg"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean: At World's End"@en	"Gore Verbinski"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean: Dead Man's Chest"@en	"Gore Verbinski"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean (film series)"@en	"Joachim Rønning"@en	"Jerry Bruckheimer"@en
"Pirates of the Caribbean: On Stranger Tides"@en	"Rob Marshall"@en	"Jerry Bruckheimer"@en

Figure 16 : Résultat de la requête

Le même problème de redondance était présent pour cette requête aussi mais cette fois-ci, ça a nécessité une implémentation algorithmique plus maline: Nous avons groupé les producteurs et les réalisateurs par film. Nous avons utilisé une Hashmap qui a pour clé, les titres des films trouvés et comme valeur une autre HashMap identique à celle construite pour la requête précédente. Il a fallu qu'on vérifie à chaque fois si un film était présent dans la map pour lui ajouter les nouvelles informations trouvées, sinon on lui crée une entrée dans la map.

Le résultat est illustré par l'objet json suivant :

```
films: {
    The Berlin Affair:{
        producteurs: [Menahem Golan, Yoram Globus]
        realisateurs: [Liliana Cavani]
    }
    Pirates of the Caribbean: Dead Men Tell No Tales:{
        producteurs: [Jerry Bruckheimer]
        realisateurs: [Joachim Rønning, Espen Sandberg]
    }
}
```

Figure 17 : Résultat de la requête sur un acteur

4. Conception du médiateur

Le médiateur permet répondre aux deux types de requêtes: recherche par titre du film ou par acteur.

Pour le premier type on introduit un titre d'un film puis on interroge notre base de données locale pour récupérer l'ensemble de ses informations qui sont disponibles. Ensuite on appelle le notre client Rest pour faire une requête HTTP pour trouver sa description depuis l'api open movie. On fait aussi appel au client Sparql pour compléter ces informations. A la fin on combine les trois résultats et on les affiche.

Le deuxième type prend un acteur comme paramètre qui sera passé au client Sparql pour récupérer la liste des films dont il a joué, ensuite on interroge notre base de données pour récupérer les informations qui concernent ces films. Enfin on appelle notre client Rest pour chercher leurs descriptions et compléter les informations de ces films.

5. Conclusion

Ce travail nous a permis de concrétiser de manière pratique tout ce que nous avons acquis dans le module IED.