28/06/2024

RT0805- Programmation répartie :
Rapport projet vidéothèque avec Jakarta EE

JULIEN BOUTREAUX

II)Introduction :

Le but de ce projet est de faire une vidéothèque en ligne en utilisant jakarta EE. Jakarta Entreprise Edition nous fournit un ensemble d'API qui facilitent le développement d'applications web. J'utilise également maven qui est un outil de gestion de projet qui me permet de faciliter le processus de construction du projet et de gérer les dépendances. J'utilise également payara qui est un serveur d'application open-source dérivé de GlassFish pour déployer mon application de manière efficace. Pour ce projet j'ai choisi de stocker mes différentes données sous forme de fichier xml.

III)Dépendances du projet :

Pour le projet il est nécessaire d'installer certaines dépendances importantes. J'utilise JAXB, JAXB permet de convertir des objets java en xml et inversement. J'utilise également JSP (JavaServer Pages). JSP me permet d'utiliser des pages web dynamiques basés sur java. Pour pouvoir manipuler les fichier JSP ou utiliser la bibliothèque JAXB pour manipuler plus facilement les fichiers xml. Il faut préciser les dépendances dans le fichier pom.xml avec les balises <dependency>, pour que les dépendances s'installe il faut ensuite utiliser le goal : ./mvnw clean install.

Dépendances pour JAXB et JSP :

```
<dependency>
    <groupId>jakarta.servlet.jsp.jstl</groupId>
    <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
    <version>2.0.0</version>
</dependency>
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
    <version>4.0.0</version>
</dependency>
<dependency>
    <groupId>com.sun.xml.bind</groupId>
    <artifactId>jaxb-impl</artifactId>
    <version>4.0.0</version>
    <scope>runtime</scope>
</dependency>
```

Stockage des données et classes :

J'ai choisi de stocker mes données sous forme de fichiers XML, je stocke les films dans le fichier films.xml, je stocke aussi les genres dans le fichier genres.xml et les utilisateurs dans le fichier utilisateurs.xml. J'utilise JAXB pour manipuler et créer les fichiers xml. Pour créer un fichier xml qui contient une liste de film avec JAXB, il faut d'abord que je crée une classe film.java et pour les attributs que je veux stocker dans le xml je mets @XmlElement pour qu'il soit pris en compte par JAXB. Il faut aussi crée la classe Liste_Films.java qui aura un seul attribut films qui est un ArrayList d'objet Film. Ainsi, je peux créer un JAXBContext avec l'instance Liste_Films.class. Et cela me permettre de transposer un objet Liste_Films en xml. C'est exactement le même fonctionnement pour utilisateurs.xml et genres.xml.

Exemple classe Film :

```java
@XmlRootElement
public class Film {
    @XmlElement
    private String title;
    @XmlElement
    private String release_date;
    @XmlElement
    private String moyenne;
    @XmlElement
    private String nb_vote;
    @XmlElement
    private String id;
    @XmlElement
    private String overview;
    @XmlElement
    private String poster_path;
    @XmlElement
    private String backdrop_path;

    @XmlElementWrapper(name = "genre_ids")
    @XmlElement(name = "genre_id")
    private List<Integer> genre_id;
```

Pour les listes de plusieurs éléments comme genre_id par exemple, on définit un @XmlElementWrapper qui va contenir la liste.

Résultat :

```
<film>
    <title>SW5zaWRlIE91dCAy</title>
    <release_date>2024-06-11</release_date>
    <moyenne>7</moyenne>
    <nb_vote>882</nb_vote>
    <id>1022789</id>
    <overview>RnJhaWNoZW1lbnQgZGlwbMO0bcOpZSwgUmlsZXkgZXN0IGTDqXNvcm
    <poster_path>/eHUWo4AiomQwG8EpWhvNNA1RMYz.jpg</poster_path>
    <backdrop_path>/xg27NrXi7VXCGUr7MG75UqLl6Vg.jpg</backdrop_path>
    <genre_ids>
        <genre_id>16</genre_id>
        <genre_id>10751</genre_id>
        <genre_id>12</genre_id>
        <genre_id>35</genre_id>
    </genre_ids>
</film>
```

IV)Récupération des données de l'API TMDB :

Mon fichier api_TMDB.java qu'on lance avec le main.java permet de récupérer des films en utilisant l'api de TheMovieDatabase avec la fonction getFilm(). Pour que l'api me retourne des films on utilise l'url suivante :

https://api.themoviedb.org/3/discover/movie?api_key=

J'ajoute ensuite à cette URL ma clé api, le langage voulu et j'ai aussi préciser que je voulais des films qui ont au minimum 7 de moyenne et 100 votes minimum pour remplir ma base de données avec des films assez populaires.

L'api nous envoi les films sous forme de json et il faut ensuite crée les objets films avec les données et utilise JAXB pour convertir la liste de film en xml.

J'ai eu quelques problèmes de conversion au fichier xml, pour le titre et le résumé des films, ces chaînes de caractères n'étaient pas au format utf-8, ce qui provoque des erreurs lors de la création du fichier xml, ma solution fût de convertir le titre et le résumé en base64 pour régler le problème (pour l'affichage sur le site je reconvertis les données).

La fonction get_genre_film() fonctionne de la même manière pour récupérer la liste des genres.

V)Explication générale de l'application

1)Index.html :

C'est la première page du site, on peut accéder à la page login et à la page register à partir de cette page.

2)Register.html et RegisterServlet :

Une page dédiée à l'inscription des utilisateurs, le formulaire envoi les données en post à la servlet RegisterServlet. La servlet consulte d'abord le fichier utilisateurs.xml pour vérifier que le pseudo n'a pas déjà été pris par un autre utilisateur (le pseudo est unique pour chaque utilisateur car je m'en sers pour supprimer les utilisateurs). Si le pseudo est déjà pris alors l'utilisateur est renvoyé sur la page register.html et dois à nouveau remplir le formulaire. Sinon l'utilisateur est ajouté au fichier utilisateurs.xml et il est renvoyé à la page login.html pour se connecter.

3)Login.html, LoginServlet et FilmsPopulaireServlet :

L'utilisateur doit entrer son pseudo et son mot de passe pour se connecter. Les données sont envoyées en post à LoginServlet. La servlet va vérifier si le pseudo et le mot de passe correspondent bien à un utilisateur. S'il n'y a personne qui correspond l'utilisateur est renvoyé sur la page index.html. Si c'est le cas alors la servlet crée une nouvelle session, la session a deux attribut username et isAdmin. Si le pseudo de l'utilisateur est « julien » et le mot de passe « juju02140 » alors isAdmin est égal à true, pour les autres je mets false. Ensuite elle va appeler la servlet FilmsPopulairesServlet qui s'occupe de récupérer une liste de 20 films dans films.xml et de rediriger l'utilisateur sur la page home.jsp.

4)Base_site.jsp :

Je me sers de cette page comme un template qui sert de base pour d'autres pages du site comme home.jsp, recherche.jsp par exemple. Il contient une barre de navigation, cette barre est enssentiel, c'est elle qui offre toutes les fonctionnalités du site. Elle permet d'aller sur la page home, la vidéothèque, de se déconnecter, de rechercher un film par genre ou de rechercher un film par mot. Si l'utilisateur est un admin alors un autre bouton Admin apparaîtra, ce qui lui permet d'accéder à la page de l'admin.

5)Home.jsp :

Elle affiche la liste des 20 films que filmsPopulaireServlet lui envoie.

6)Description_film.jsp et DescriptionFilmServlet :

Si on clique sur les films affichés sur le site la servlet DescriptionFilmServlet sera appelé elle récupère en get l'id du film qui a été cliqué et la recherche dans films.xml. Ensuite

elle envoie les données du film à description_film.jsp. La page va afficher le titre, la date de sortie, la moyenne, le nombre de vote et le synopsis du film. L'utilisateur pourra choisir d'ajouter ce film à sa vidéothèque ou non.

7)RechercheFilmGenreServlet :

Dans la barre de navigation, on a le bouton genre qui est une liste déroulante avec tous les genres disponibles, en cliquant sur l'un des genres cela appellera la servlet RechercheFilmGenreServlet qui récupère l'id du genre en get. Elle va ensuite parcourir films.xml et récupérer tous les films qui ont l'id du genre recherché. Elle va ensuite transmettre la liste des films à la page recherche.jsp et redirigé l'utilisateur sur celle-ci.

8)RechercheFilmMotservlet :

Même fonctionnement que pour la recherche de genre, dans la barre de navigation, on peut entrer un mot pour rechercher le titre d'un film. Si l'utilisateur appuie sur le bouton alors le formulaire envoie le mot en get à la servlet. La servlet va ensuite rechercher dans films.xml les films dont le titre commence par le mot spécifié. Elle transmet ensuite la liste des films trouvés à recherche.jsp.

9)VideothequeServlet.java :

Cette servlet est appelée quand l'utilisateur clique sur le bouton vidéothèque dans le menu. Elle va se servir de la session pour avoir le nom de l'utilisateur concerné et le chercher dans utilisateurs.xml pour récupérer la liste de film qu'il possède et envoyer ses films à la page videotheque.jsp

10)AjoutVideothequeServlet.java :

Lorsque l'utilisateur est sur la page description_film.jsp. Il peut choisir d'ajouter le film à sa vidéothèque. Lorsqu'il clique sur le bouton ajouter, cela appelle la servlet AjoutVideothequeServlet.java. La servlet va utiliser la session pour retrouver l'utilisateur dans utilisateurs.xml et ainsi récupérer sa liste de film. Elle récupère en get, l'id du film que l'utilisateur veut ajouter. Elle va retrouver le film concerné et l'ajouter à a liste des films de l'utilisateur. Et ensuite on appelle la méthode registrer de l'utilisateur qui va réécrire les nouvelles données.

11)SupprimerFilmVideothequeServlet :

Sur la page videothèque.jsp, l'utilisateur peut supprimer les films de sa vidéothèque en appuyant sur le bouton supprimer. Cela va appellé la servlet SupprimerFilmVideothequeServlet, elle récupère l'id du film en get, elle utilise la session pour retrouver le nom d'utilisateur et va supprimer le film de la liste des films de l'utilisateur, on appelle encore la méthode register pour réécrire l'utilisateur avec les nouvelles données.

12)AdminServlet.java :

Elle est appelée quand on appuie sur le bouton admin (qui n'est visible que par l'admin), elle vérifie avec la session si celui qui essaye d'appelé la servlet est bien un admin. Si ce n'est pas le cas, on revoit l'utilisateur vers la page index. Sinon la servlet récupère la liste des utilisateurs dans le fichier utilisateurs.xml puis envoi la liste des utilisateurs à la page_admin.jsp.

13)Page_admin.jsp :

Elle affiche la liste des utilisateurs inscrit au site, l'admin peut supprimer les utilisateurs (fonctionnel). Il y a également une option pour rechercher des films sur TMDB pour en ajouter à notre films.xml (pas fonctionnel).

14)SupprimerUtilisateurServlet.java :

Elle est appelée quand l'admin clique sur le bouton supprimer associer à un utilisateur. Elle vérifie d'abord avec la session si l'utilisateur qui appelle est un admin, puis récupère l'id de l'utilisateur que l'admin veut supprimer en get. Puis trouve l'utilisateur dans le fichier utilisateurs.xml et le supprime du fichier xml.

15)DeconnexionServlet.java :

L'utilisateur peut se déconnecter, il y a un bouton compte qui est une liste déroulante, et il y a un bouton déconnecter. Si on appuie dessus, ça appelle la DeconnexionServlet.java. Cette servlet va supprimer la session et rediriger l'utilisateur vers l'index.

16)Xmlfonctions.java :

Ce fichier regroupe la plupart des fonctions que les servlets utilisent comme lire_film_xml() qui retourne la liste de tous les films du fichiers films.xml, lire_user_xml() retourne tous les utilisateurs du fichiers utilisateurs.xml, trouverFilm(String film_id) qui cherche un film en fonction de son id et le retourne, trouverFilmMot qui retourne la liste des films qui commencent par le mot. Et enfin decoder_base_64(String mot64) qui reconvertit la chaîne de caractère encodé en base64 (le titre et le synospsis des films) en chaîne ce caractère lisible.

VI)Fonctionnement d'un fichier jsp :

Je fais d'abord les inclusions nécessaires pour que le fichier accepte le format UTF-8

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ include file="base_site.jsp" %>
```

Le taglib me permet d'utiliser d'importer une bibliothèque de balise, ici c'est pour importer la bibliothèque JSTL Core qui me permet d'utiliser les balises comme <c :forEach> par exemple. J'inclus également mon template base_site.jsp contenant le menu du site.

Exemple avec FilmPopulairesServlet.java et home.jsp :

```
request.setAttribute(name:"filmsPopulaires", filmsPopulaires);

// Forwarder la requête à home.jsp
RequestDispatcher dispatcher = request.getRequestDispatcher(path:"../home.jsp");
dispatcher.forward(request, response);
```

Après avoir sélectionner les 20 premiers films contenu dans films.xml. La servlet va ajouter un attribut filmsPopulaires à la requête qui est une liste d'objet Film, ensuite le dispatcher va transferer la requête à la page home.jsp

Contenu page home.jsp :

```
<p>Bienvenue: ${username}</p>
<p>Les films populaires du moment</p>

<ul class="film-list">
    <!-- Boucle sur la liste des films populaires -->
    <c:forEach var="film" items="${filmsPopulaires}">
        <li class="film-list-item" data-bs-toggle="tooltip" data-bs-placement="top" title="${film.title}">
            <h5>${film.title}</h5>
            <a href="/rest/description_film?film_id=${film.id}">
                <img src="https://image.tmdb.org/t/p/w342/${film.posterPath}" alt="${film.title} Poster">
            </a>
        </li>
    </c:forEach>
</ul>
```

Maintenant la page home.jsp peut exploiter la liste des films, elle crée une liste à puce qui contiendra tous les films présents dans la liste, elle affichera le titre et son poster.

La méthode est le même pour les autres pages jsp.

VII)Fonctionnement d'une servlet :

Exemple avec RechercherFilmServlet :

```java
public class RechercherFilmGenreServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // Log the search term
        String genre = request.getParameter(name:"genre");
        System.out.println("Id du genre recherche : " + genre);

        List<Film> films = XmlFonctions.trouverFilmGenre(genre);

        for (Film film: films){
            String title64 = film.getTitle();
            String title_decode = XmlFonctions.decoder_base_64(title64);
            System.out.println("titre decode" + title_decode);
            film.setTitre(title_decode);
        }

        request.setAttribute(name:"films", films);

        RequestDispatcher dispatcher = request.getRequestDispatcher(path:"../recherche.jsp");
        dispatcher.forward(request, response);
```

Une servlet hérite de la classe HttpServlet, si elle récupère les données en get, on utilie la méthode doGet, si c'est en post doPost. Ensuite elle récupère les données de la requête qu'elle reçoit en utilisant request.getParameter(). Ensuite elle traite les données, dans notre exemple la servlet trouve la liste des films appartiennent au même genre que celui recherché. Ensuite elle transmet la liste des films à la page recherche.jsp.

Pour que la servlet soit prise en compte il faut la spécifier dans le web.xml

```xml
<servlet>
    <servlet-name>RechercherFilmGenreServlet</servlet-name>
    <servlet-class>org.RechercherFilmGenreServlet</servlet-class>
</servlet>
```

```xml
<servlet-mapping>
    <servlet-name>RechercherFilmGenreServlet</servlet-name>
    <url-pattern>/rest/recherche_genre</url-pattern>
</servlet-mapping>
```

La balise <servlet> sert à spécifier le nom de la servlet et son package et <servlet-mapping> sert à spécifier le chemin de l'url associé à la servlet.

VIII)Fonctionnalités manquantes :

Sur la page description_film.jsp, on peut voir qu'il y a une option pour noter le film, je n'ai pas eu le temps de l'implémenter. L'admin devait aussi avoir l'option de pouvoir rechercher directement des films sur l'api TMDB pour les ajouter au fichier film.xml mais ça n'a pas été implémenté.

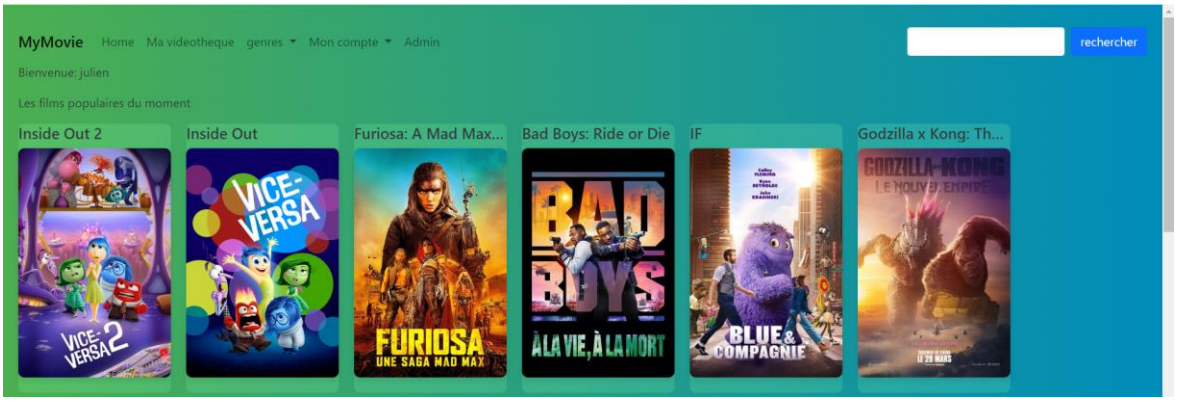
IX)Conclusion

En conclusion, la plupart des fonctionnalités attendus ont été implémentés :

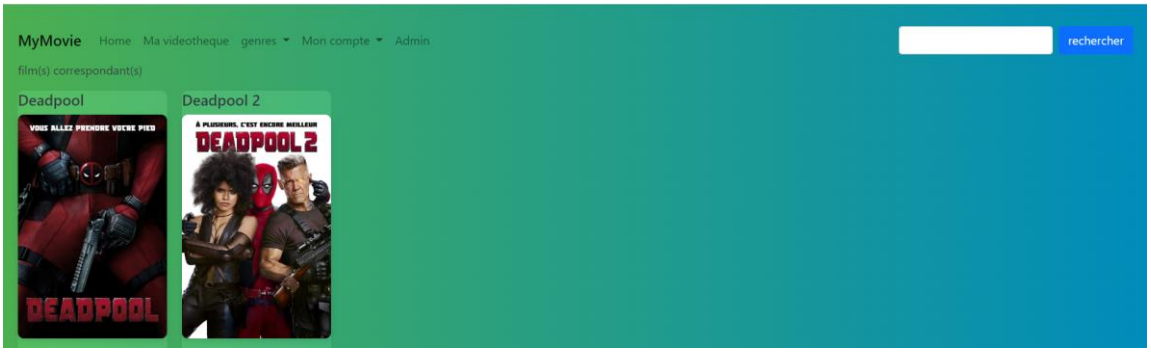
L'inscription, la connexion, la mise en place d'une session qui gère utilisateurs normaux et l'admin, la recherche de film par genre, la recherche de film par mot, un utilisateur peut ajouter et supprimer des films sur sa vidéothèque, l'admin est le seul qui peut consulter la page admin et choisir par exemple de supprimer des utilisateurs.

X) Annexes

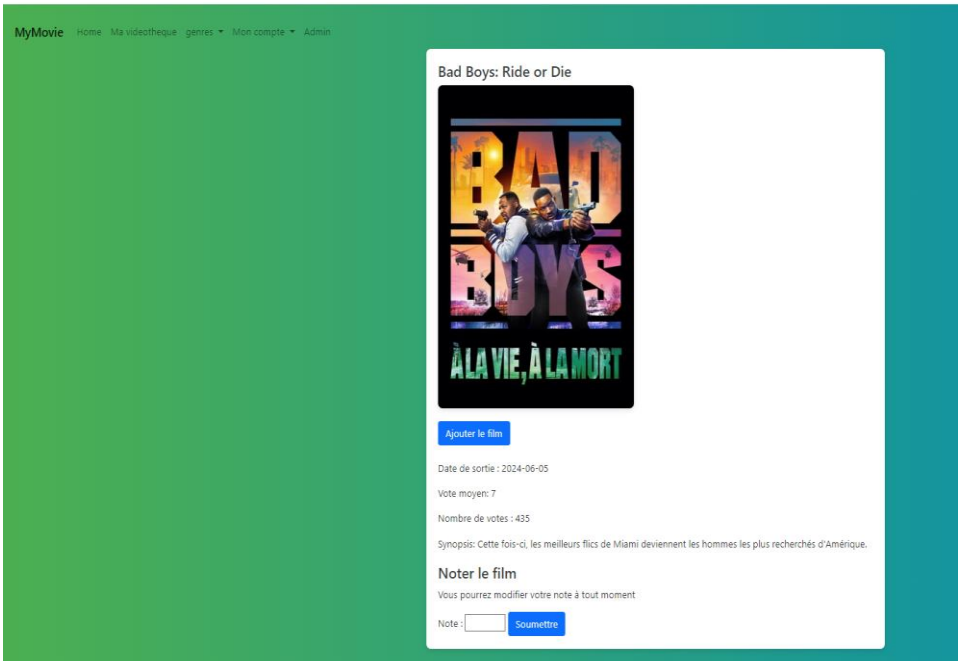
1)Affichage page home :



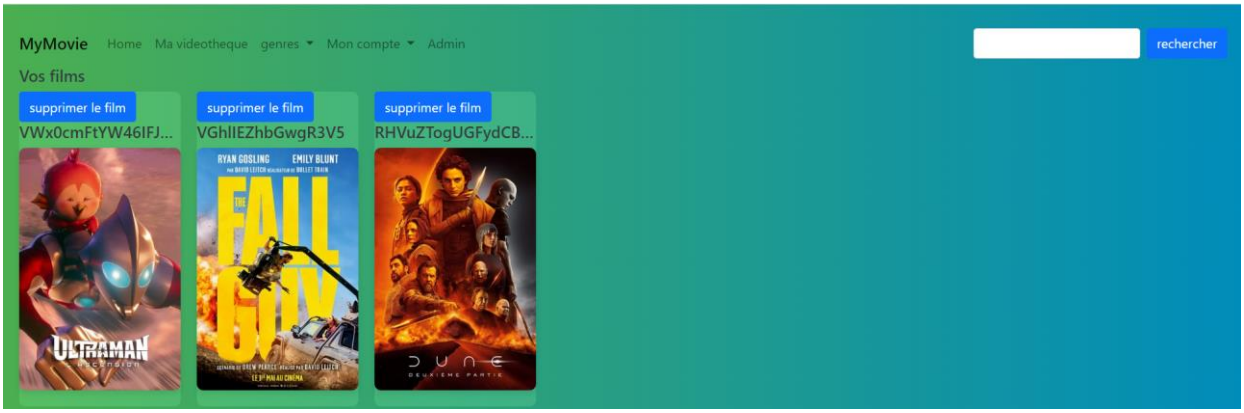
2) Affichage page résultat avec come mot recherché dead :



3) Page description :



4) Page vidéothèque :



5) Page Admin :