|  |  |
| --- | --- |
|  | **2011** |
|  | Leonhard hermle  Julien Cellier  Xavier Mourgues |

|  |
| --- |
| **[Rapport Projet tuteurè]** |
| Rapport du projet tuteuré XML exprimant les besoins, les objectifs et les moyens de mise en œuvre du projet ainsi que les problèmes rencontrés et le résultat atteint en fin de projet. |

Université Toulouse I – IUT de Rodez

Année 2010/2011

TABLE DES MATIERES

[REMERCIEMENTS 4](#_Toc287426511)

[Introduction 5](#_Toc287426512)

[1. Présentation du sujet 6](#_Toc287426513)

[1.1. Les besoins 7](#_Toc287426514)

[1.2. Les contraintes 7](#_Toc287426515)

[1.3. L’univers de travail 7](#_Toc287426516)

[1.4. Répartition des tâches 8](#_Toc287426517)

[2. Java 9](#_Toc287426518)

[2.1. Tidy 9](#_Toc287426519)

[2.2. Cibler l’information 9](#_Toc287426520)

[2.3. Flexibilité 9](#_Toc287426521)

[2.4. Principe de fonctionnement 10](#_Toc287426522)

[2.5. Problemes rencontrés 10](#_Toc287426528)

[3. Base de données 12](#_Toc287426529)

[3.1. Modèle Conceptuel des données 12](#_Toc287426530)

[3.2. Schéma XSD du document XML 12](#_Toc287426531)

[3.3. XMLDB 13](#_Toc287426532)

[3.4. Le type de données XMLType 13](#_Toc287426533)

[3.4.1. Exemple d’une insertion dans la table film : 13](#_Toc287426534)

[3.4.2. Exemple de récupération simple d’une valeur contenu dans le XML 14](#_Toc287426535)

[3.5. Trigger 14](#_Toc287426536)

[3.6. Vue 15](#_Toc287426537)

[3.7. Problèmes rencontrés 15](#_Toc287426538)

[4. Restitution des données 16](#_Toc287426539)

[4.1. Objectifs 16](#_Toc287426540)

[4.2. Aspect graphique 17](#_Toc287426541)

[4.2.1. Police de caractère 17](#_Toc287426542)

[4.2.2. Les couleurs 17](#_Toc287426543)

[4.2.3. Les liens 17](#_Toc287426544)

[4.2.4. La taille des éléments 18](#_Toc287426545)

[4.2.5. Le menu 18](#_Toc287426546)

[4.2.6. Le champ de recherche 18](#_Toc287426547)

[4.2.7. La pagination 18](#_Toc287426548)

[4.2.8. La recherche par alphabet 19](#_Toc287426549)

[4.3. Aspect technique et structure de l’interface 19](#_Toc287426550)

[4.3.1. Le modèle MVC 19](#_Toc287426551)

[4.3.2. Schéma de navigation 20](#_Toc287426552)

[4.3.3. Le fichier conf.ini 20](#_Toc287426553)

[4.3.4. La pagination 22](#_Toc287426554)

[4.3.5. L’Ajax 23](#_Toc287426555)

[4.3.6. Extraction XML 24](#_Toc287426556)

[4.3.7. Exécution d’une application java via PHP. 24](#_Toc287426557)

[5. Conclusion 26](#_Toc287426558)

REMERCIEMENTS

Nous souhaitons tout d’abord adresser nos remerciements à ceux qui nous ont permis de réaliser ce projet tuteuré, à ceux qui nous ont encadrés et qui nous ont rendus cette expérience agréable et enrichissante.

Nous remercions PATRICK SABI, notre professeur qui nous à encadré tout au long de notre projet et de nous avoir apporté des conseils.

Merci également à BRUNO BELIERE d’avoir mis à disposition deux bases de données Oracle qui nous ont permis de mener à bien notre projet.

Un grand merci à ALLOCINE qui autorise le parcours des pages HTML de son site afin de permettre la récupération de données textuelles.

Introduction

Dans le cadre de nos projets tuteurés de l’année 2010/2011, notre groupe à choisi de se lancer dans la réalisation d’une application permettant d’exploiter du contenu de pages HTML utilisant du format XML dans une base de données afin de pouvoir les stocker et les restituer pour que l’utilisateur puisse jouir des différentes informations dans une même interface graphique. Ainsi nous avons dû faire face à certains choix plus ou moins difficiles à débattre au sein du groupe.

Certaines questions évidentes se posent tel que, quels ont été ces choix ? Pourquoi ces choix ?

Nous établirons une brève présentation de ce projet mettant en avant les choix effectués, nos besoins, les contraintes découlant des décisions prises pour aboutir sur les problèmes rencontrés lors de la période de projet. Enfin, nous conclurons pour faire le bilan de ce dernier.

1. Présentation du sujet

Après mûres réflexions nous nous sommes orientés vers l’exploitation de sites cinématographiques tels qu’Allociné ou IMDB afin de réutiliser des données concernant les films, tel que les avis ou notes attribués par les internautes. D’autres choix auraient été possibles comme par exemple l’exploitation de sites météorologiques (restitution de statistiques pour une période donnée). Ce dernier choix figurant déjà dans un projet des années précédentes, nous avons préféré partir sur de nouvelles bases en choisissant un nouveau thème, le cinéma.

L’objectif du projet est donc de restituer à partir de sites web existants des données de films (critiques, commentaires, notes, bande annonce, informations diverses…). Ainsi, les sites exploités seront ceux comportant des informations diverses pour un même élément recherché (un film en particulier). Les commentaires et les notes d’utilisateurs et des sites seront rassemblés permettant une vue d’ensemble sur le film recherché, tout ceci pouvant servir à comparer et fusionner les avis d’un film donné.

Plus simplement, il sera possible de regrouper les commentaires et notes restitués via une interface graphique développé en langage web, notamment en PHP. Cette partie du projet sera expliquée plus en détails dans le chapitre « Restitution de données ». Le parcours (parsing) des pages des sites web se fera en java.

Le projet se présente donc en 3 étapes :

* La **récupération de données** à travers divers sites web évolutif et dynamique (plus en détail dans le chapitre « JAVA »).
* Le **stockage de ces données** dans une base de données Oracle v10.2 (grâce à JAVA et plus en détail dans le chapitre «Base de données »).
* La **restitution visuelle des données** pour les utilisateurs (plus en détail dans le chapitre «Restitution de données »).

A l’origine, les sites web que nous avons sélectionnés étaient Allociné et IMDB. Malheureusement, se posait un souci d’autorisation car le site en question bloque les robots qui parcourent ses pages. Ce problème est mis en avant dans le chapitre des problèmes dans la rubrique « java ». Par conséquent, le choix d’un second site autre qu’IMDB contenant une quantité importante de données de films, non négligeables. IMDB et l’une plus grande base de données de films mondiale rassemblant des informations sur tous les films disponibles, c’est pourquoi l’impossibilité d’exploitation de leur site internet est très regrettable étant donné que la comparaison de données entre un site de référence comme IMDB et un second peut montrer des différences importantes.

Nous aurons une première phase de découverte visant à étudier les différentes techniques de mise en œuvre d’un tel projet.

Plus tard, dans un souci d’évolution, plus d’information pourront être récupéré et d’autres sites dit «source d’information» pourront être rajoutés. En effet, dans la mesure du possible un site pourra être ajouté afin de pouvoir être exploité. La démarche à suivre sera indiquée à l’utilisateur. Cette option étant assez complexe à réaliser, d’autant plus que les délais de finalisation du projet sont serrés, ne pourra peut-être pas être créée.

## Les besoins

Nous avons besoin de récupérer des données de sites web qui ne sont pas structuré en XML mais en HTML ce qui nous posera problème lorsque voudront traiter ces données. En effet, un site étant évolutif il est possible que sa structure HTML change. Pour parer cet effet, nous devrons réussir à soutirer uniquement les informations nécessaires et non les informations inutiles (publicités …). Pour cela l’utilisation de la technologie DOM sera indispensable.

Dans un deuxième temps, nous aurions besoin d’une machine avec un accès à une base de données Oracle ainsi que qu’un accès libre a Internet (sans proxy bloquant les sites tel qu’Allocine).

## Les contraintes

L’une des premières contraintes que nous avons est l’utilisation d’Oracle v10.2.

D’autre part l’extraction des données se fera par des primitives XSLT mais la structure de données sera en HTML ce qui nous forcera à analyser syntaxiquement ces informations afin d’avoir ce qui se rapproche le plus d’une structure XML.

## L’univers de travail

Nous avons décidé que pour la récupération des données : nous utiliserons un univers java afin de profiter de sa souplesse d’utilisation de ressources tiers tel que le SGBD Oracle via JDBC, le logiciel JTIDY et les primitives XSLT.

Le stockage des données sera partiellement inséré sous format XML et stocké dans une base Oracle.

Pour finir la restitution des données se fera via une PUI en PHP/HTML.

## Répartition des tâches

Nous avons décidé de séparer le projet en 3 parties au sein du groupe.

* M. Leonhard Hermle prendra à sa charge la PUI en PHP/HTML
* M. Cellier Julien s’occupera du stockage en base de données et XML avec interfaçage pour le PHP et le Java.
* M. Mourgues Xavier traitera la récupération des données en Java.

1. Java

La partie Java, ou « extraction des données » traitent de tout ce qui concerne les techniques de prise d’information sur les sites web pour ensuite le stocker en base de données. Cette technique, appelé le « scrapping » est confronté a plusieurs problèmes : d’une part le fait que toutes pages HTML ne sont pas syntaxiquement correcte pour être considérées comme des pages XML, et d’autre part que le web étant dynamique et évolutif, la construction d’une page peut changer du tout au tout dans le temps.

## Tidy

L’utilisation du logiciel tidy, ou de sa librairie jtidy pour java permet de nettoyer un fichier HTML syntaxiquement pour pouvoir le considérer comme un fichier XML. De plus, une fois le fichier transformé en XML, cette librairie permet de naviguer dans l’arbre d’un fichier XML avec beaucoup de méthodes facilitant cette navigation.

## Cibler l’information

Lors de la récupération d’une page web, beaucoup de choses peuvent changer (tel que les pubs, la disposition des menu, des rubriques nouvelles ou disparaissant) c’est pour cela que nous devons donc cibler l’information.

Grace a un algorithme de recherche dans l’arbre XML, il nous a été possible de trouver une information telle que, par exemple,  la durée d’un film. En effet grâce à un mot clé que nous savons unique dans la page (« Durée ») nous pouvons parcourir l’arbre pour y trouver le nœud qui contient ce mot clé. Et ainsi parser l’information de durée qui nous intéresse.

## Flexibilité

Dans une première phase du projet, nous avons extrait les informations du site allociné de manière fixe. C'est-à-dire en analysant plusieurs pages de détails de film et en repérant les mots clés et éléments uniques qui nous permettaient d’extraire les informations voulues. Cependant, il faudrait donner la possibilité à l’utilisateur d’ajouter un site qu’il trouve intéressant sans pour autant qu’il ait à toucher a la programmation. C’est pour cela qu’une certaine flexibilité est à rajouter quand à l’identification des éléments importants sur une page quelconque.

## Principe de fonctionnement

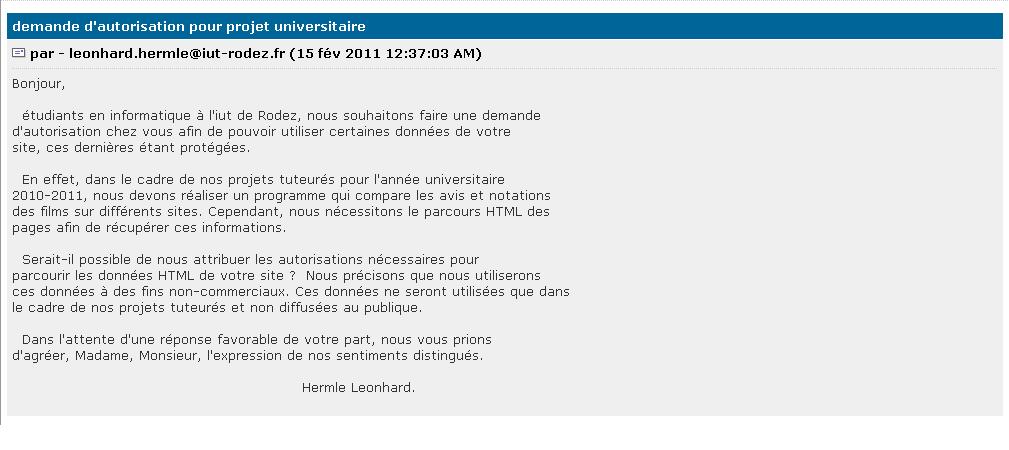
Actuellement, le programme Java fonctionne de la manière suivante :

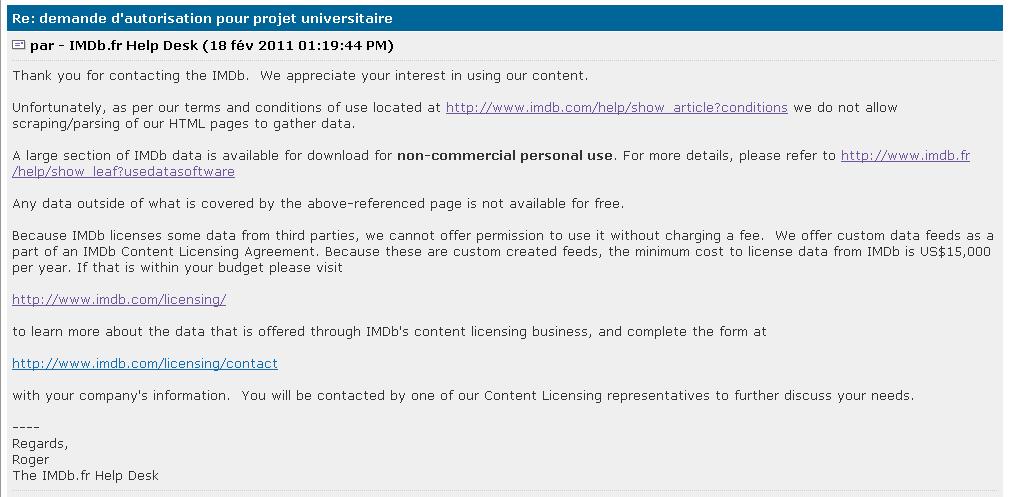
* L’utilisateur le lance avec une url d’un film qui l’intéresse (Allociné uniquement).
* Le programme télécharge alors toute la page.
* La librairie Jtidy rend la page accessible en tant que format XML
* On cible alors l’information nécessaire (tel que le titre du film, la durée, les acteurs, les commentaires, etc…)
* Une fois toutes les informations récupérées on les stock dans une base de données oracle.



## Problemes rencontrés

**IMDB** : Le site web IMDB refuse strictement le parcours des pages par un robot grâce à la protection par proxy. Cela pose problème pour notre projet car ce site contient l’une des plus grandes bases de données de films, très complet, fiable mais aussi détenteur de nombreux commentaires et critiques de films qui nous auraient été très utiles. Par conséquent, nous avons décidé de prendre contact avec cette société en expliquant les faits, afin de demander une autorisation de parcours des données. Malheureusement, nous avons reçus une réponse négative. En effet, la charte stipule l’interdiction de parsing / scrapping de données. Cependant, une sélection de données peut être téléchargée gratuitement où encore une licence permettant l’accès à la totalité des données peut être acheté. Dans les deux cas, cette solution va à l’encontre de notre projet qui consiste à parcourir des pages spécifiques dans le but d’en extraire les données. Voici le message envoyé à IMDB et leur réponse vis-à-vis de ce dernier :

Demande d’autorisation d’exploitation des données auprès de IMDB

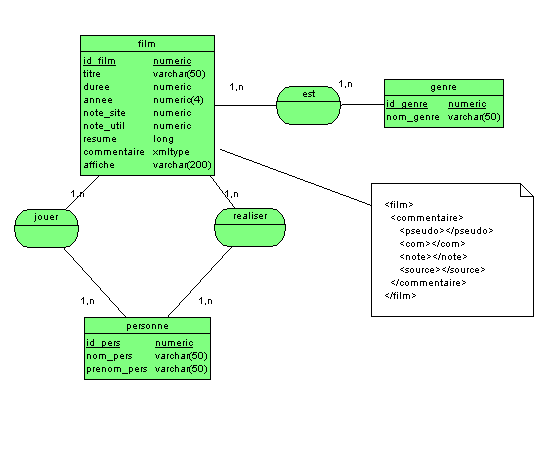
Réponse de IMDB suite à notre demande

1. Base de données

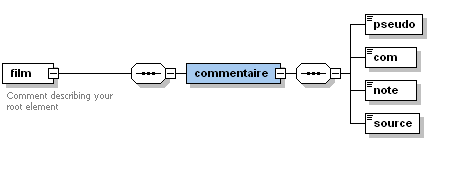
Au début de notre projet s’est posé le problème du choix du type de base de données que nous allions utiliser. En effet, il nous est mis à disposition deux bases de données Oracle XMLP1 et XMLP2. Il fallait donc choisir entre une base de données relationnelle qui prévoit l’utilisation de SQL2 ou une base de données objet utilisant du SQL3.

Pour une utilisation de ce genre il s’est révélé plus simple d’utiliser une base de données relationnelle.

## Modèle Conceptuel des données



## Schéma XSD du document XML



## XMLDB

Pour stocker des documents XML dans la base de données Oracle nous avons utilisé le format XMLType notamment grâce à Oracle XML DB.

Oracle XML DB est un module de gestion XML intégré au Système de Gestion de Base de Données (Dans les technologies de l'information (TI), une donnée est une description élémentaire, souvent codée, d'une chose, d'une transaction d'affaire, d'un événement, etc.) Oracle.

Cette technologie (Le mot technologie possède deux acceptions de fait :) étend les possibilités de la base de données relationnelle d'Oracle en proposant toutes les fonctionnalités d'une base de données XML. Ce module offre une structure indépendante de stockage et de gestion de données XML. Il propose les fonctionnalités suivantes :

* Un type de données XML qui est utilisé pour stocker et gérer du contenu XML.
* Une collection de méthodes et des opérateurs SQL permettant d'agir sur du XML.
* La possibilité d'enregistrer un schéma conforme au W3C dans la base de données Oracle.

## Le type de données XMLType

Le type de données XMLType a fait son apparition dans la version 9 d'Oracle. Il permet de créer, extraire et indexer des données XML stockées dans le SGBD Oracle. En tant que nouveau type de données, XMLType peut être utilisé comme type pour colonnes dans des tables et des vues ainsi que dans des procédures stockées PL/SQL. Avec le type XMLType et ses possibilités, on dispose de toutes les possibilités d'une base de données relationnelle tout en travaillant avec des données sous format XML.

### Exemple d’une insertion dans la table film :

*INSERT INTO film*

*VALUES ( 6, 'gran torino', 138, 2010, 9 ,9.5 ,'clint a la retraite...',XMLType('*

*<film>*

*<commentaire>*

*<pseudo>ford\_48</pseudo>*

*<com>trés bon film avec M. Eastwood</com>*

*<note>10</note>*

*<source>Allocine</source>*

*</commentaire>*

*<commentaire>*

*<pseudo>jojodu12</pseudo>*

*<com>Clint encore une fois énorme !!</com>*

*<note>8</note>*

*<source>Allocine</source>*

*</commentaire>*

*</film>'),'' )*

### Exemple de récupération simple d’une valeur contenu dans le XML

*SELECT f.commentaire.extract('/film/commentaire/com/text()')*

*FROM film f;*

Extract : Cette fonction permet d'extraire l'argument d'un XMLType dans le chemin spécifié et retourne la valeur du nœud.

Exemple : <com>Ce film est trop bien</com>

text() : Cette fonction permet de récupérer juste ce qui est contenu entre les balises.

Exemple : Ce film est trop bien

Afin de fusionner les commentaires sur les différents sites nous avons créé un trigger calculant la moyenne des notes des utilisateurs contenu dans le document XML et l’insérant dans un champ de la table.

Pour ceci nous avons utilisé le langage PL/SQL associé au XPATH (XML Path Langage) qui permet de récupérer les parties d’un document XML et d’exprimer une requête qui en extraira tout ou partie du document.

## Trigger

*CREATE OR REPLACE TRIGGER moyenne*

*AFTER INSERT OR UPDATE ON film*

*FOR EACH ROW*

*BEGIN*

*NEW.notes\_util := (SELECT AVG(extract(value(note),'note/text()').getNumberVal())*

*FROM film f, table(xmlsequence(extract(f.commentaire,'/film/commentaire/note'))) note*

*WHERE id\_film = NEW.id\_film)*

*END;*

*END moyenne;*

Table() :Cette fonction permet de créer un tableau de valeur.

Xmlsequence() : Cette fonction permet de traiter l'entrée spécifiée et retourne la tête du nœud correspondant.

Exemple de résultat : Avant : Après :



## Vue

La création d’une vue aurait dû cacher l’existence du XML lors de la restitution des données mais malheureusement l’idée d’utiliser celle-ci n’est apparue que plus tard au cour de ce projet sur conseil de notre professeur tuteur. De ce fait l’extraction du XML s’effectue via PHP grâce à l’utilisation de XPATH dans les requêtes SQL.

## Problèmes rencontrés

Beaucoup de difficultés sont apparut lors de la création du trigger et de la compréhension du XPath.

1. Restitution des données

La restitution des données consiste en la création d’une interface graphique permettant à son utilisateur de naviguer aisément pour trouver les avis et commentaires concernant un film donné. En effet, il s’agit de faciliter l’accessibilité aux données dans le but de cacher les étapes techniques complexes permettant d’aboutir à la recherche. Un utilisateur peu donc être ignorant des problèmes auxquels sont confrontés les développeurs de l’application car aucune manipulation compliquée n’est à exécuter.

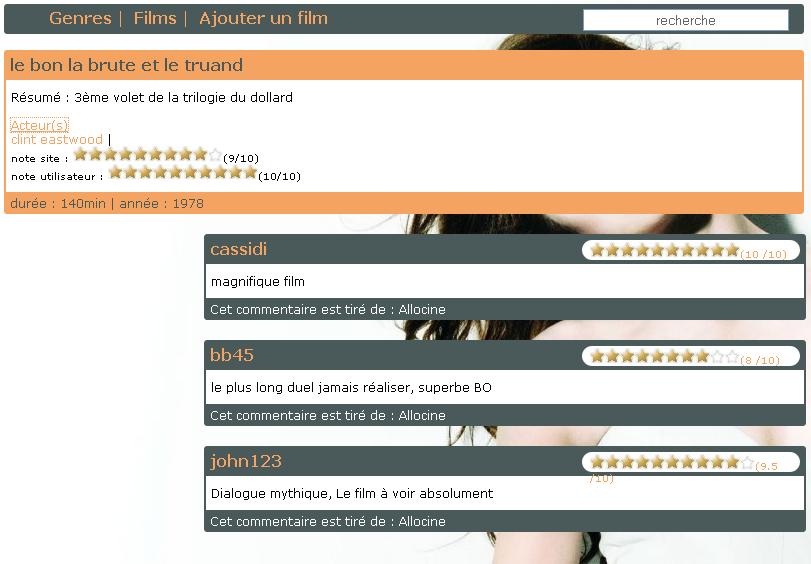
La restitution s’effectue alors principalement sous le langage de programmation PHP5 orienté objet. Bien sûr on retrouve aussi du HTML et du CSS permettant la mise en forme dynamique des pages de l’interface.

L’interface doit être ergonomique et homogène afin de faciliter la navigation mais aussi sobre tout en respectant un choix de couleur épuré. Pour ce faire, Nous avons établis des règles permettant d’atteindre facilement l’interface recherchée. Nous avons aussi établis des aspects fonctionnels et techniques permettant une navigation améliorée sur l’interface.

## Objectifs

Cette interface donne à son utilisateur la possibilité de rechercher un film, d’afficher les détails d’un film (réalisateur, acteurs, année de production, résumé…) mais aussi les commentaires et les notes des internautes ainsi que des sites.

L’utilisateur peut rechercher grâce à la sélection de différents critères (par genre, par recherche simple, par alphabet, acteurs et réalisateurs).



Page principale d’un film

## Aspect graphique

### Police de caractère

Nous avons choisis la police de caractères « verdana » sans serif où les empattements des caractères sont inexistants et donnent une impression de modernité. Le nombre de police est limité à une seule car trop de diversité donne un aspect lourd à l’interface qui devient désagréable à regarder. Cela porterait atteinte au confort de navigation de l’utilisateur. Aussi l’utilisation d’un nombre de polices de caractères trop élevé sera limité et nous n’utiliserons pas de types de police gras afin de remédier à cet effet.

### Les couleurs

Une interface optimale ne contient pas plus de trois couleurs. Ainsi, nous nous basons sur deux couleurs principales qui sont le Orange et le Kaki.

### Les liens

Afin d’optimiser la souplesse de l’interface nous utilisons au maximum des liens notamment dans la barre de menu qui est le moyen le plus efficace de recherche sur les interfaces.

### La taille des éléments

Les pages seront assez courtes et le nombre d’éléments affichés sont limités. Cela nécessite le développement d’un système de pagination en PHP permettant de répartir les films sur un nombre de pages nécessaires. Aussi, toute liste de film n’affiche que les informations capitales, utile pour sa recherche. En cliquant sur ce dernier, la fiche entière du film sera affichée, permettant ainsi de voir tous les détails relatifs au film, tels que nom d’auteur, année de production, note attribuée, commentaires, etc.…

### Le menu

Le menu comporte les principaux liens nécessaires à une recherche rapide de données. En effet, ici on retrouve des liens vers différentes pages tels que la recherche par alphabet ou par genres. Aussi on y retrouve le formulaire de recherche simplifié.

Ici nous avons à faire à un menu de type classique tel qu’on les retrouves souvent. L’avantage est la pertinence et l’efficacité de recherche car l’utilisation d’un menu est universelle pour toute application. L’utilisateur ne sera donc pas perturbé et trouvera rapidement les données de film recherchés.

Z:\Projet tut\projettuteurexml\screenshots\menu.JPG

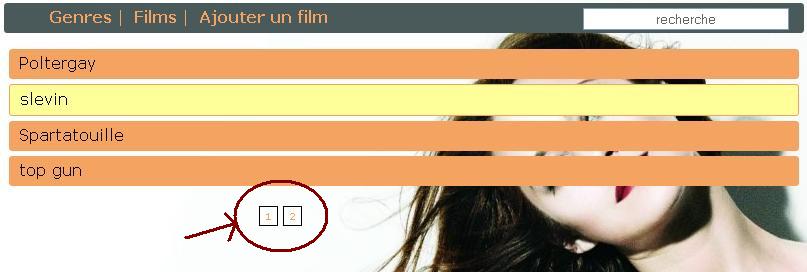
### Le champ de recherche

Il permet une recherche rapide d’un fim donné sous condition de connaitre un minimum d’information sur ce dernier (nom…). C’est une recherche approximative et plusieurs résultats peuvent être trouvés et affichés.

Le formulaire sera accompagné d’un script d’**auto-complétion** en ajax. Démuni d’un bouton de validation de la recherche, pour chaque caractère entré dans le champs, les résultats affichés seront réactualisés. Cette méthode apporte un confort conséquent à l’utilisateur mais peut cependant ralentir le système étant donné que la requête est ré-executée pour chaque nouveau caractère ajouté. Dans notre cas, les requêtes utilisées pour la recherche de données dans Oracle sont simples, n’utilisant que peu de jointures empêchant un ralentissement trop important de la recherche. Aussi, le nombre de films ne dépassant pas les quelques milliers, le parcours de la base Oracle se fera assez rapidement.

### La pagination

L’interface de restitution des données doit afficher un nombre de données assez important en l’occurrence d’une liste de films assez nombreuse. En effet, ils existent trop de films pour pouvoir les afficher sur une seule page. De ce fait, nous utilisons la solution de pagination pour les films et commentaires dès que le nombre d’éléments donnés est d’ordre trop important. Cette solution consiste en la répartition du nombre de film sur un nombre de pages données.



### La recherche par alphabet

L’utilisateur peut effectuer une recherche par alphabet en cliquant sur l’onglet menu « Films »



Code correspondant :

<?php

echo "<fieldset><center><a href='index.php?page=films&initiale=09'>0-9 </a>";

for ($i=ord("A");$i<ord("Z")+1;$i++) { //boucle pour affichage des lettres

echo "<a href='index.php?page=films&initiale=".chr($i)."'>".chr($i)."</a>";

}

echo "</center></fieldset>";

?>

## Aspect technique et structure de l’interface

### Le modèle MVC

Pour bien structurer le site de restitution nous nous baserons sur un modèle MVC qui consiste en la séparation du code produit en 3 parties distinctes que voici :

* Le **modèle** qui représente le comportement de l'application : traitements des données, interactions avec la base de données, etc. Il décrit les données manipulées par l'application et définit les méthodes d'accès.
* La **vue** correspond à l'interface avec laquelle l'utilisateur interagit. Elle est conçue en html, CSS et JavaScript. La vue n'effectue aucun traitement, elle se contente d'afficher les résultats des traitements effectués par le modèle, et de permettre à l'utilisateur d'interagir avec elles.
* Le **contrôleur** synchronise la gestion des évènements pour mettre à jour la vue ou le modèle. Ainsi il sélectionne le modèle et la vue adéquate selon les souhaits de l’utilisateur par interaction avec l’interface graphique.

Ainsi, cette méthode de conception permet de bien ordonner les différents éléments afin de faciliter le travail du développeur qui aura une facilité de navigation au sein de son code. Par conséquent, la mise à jour, elle aussi, par la même personne ou un autre développeur est plus simple et plus rapide à effectuer.

### Schéma de navigation

Page principale (affiche tous les films par défaut).

Menu

Ajouter un film (permet la recherche d’un film grâce à un formulaire via l’application java)

Page recherche simple (fonction Ajax qui fait une recherche approximative d’un film par nom)

Page genres (affichage des genres disponibles dans la base de données Oracle)

Page films (propose un tri par alphabet)

Affichage des films disponibles sur les sites. Permet de choisir le bon résultat.

Page liste film (liste les films disponibles selon les options sélectionnées. C’est un affichage simple)

Affichage détaillé d’un film

### Le fichier conf.ini

Afin de pouvoir configurer facilement l’espace graphique et la connexion à la base de données Oracle, nous utiliserons un fichier texte « conf.ini » qui comporte les éléments nécessaires à la configuration. Tout utilisateur pourra alors modifier ce fichier selon ses besoins en ouvrant ce dernier grâce à un éditeur simple.

Contenu du fichier :

* Les identifiants de connexion à la base de données, c'est-à-dire utilisateur, mot de passe et chaine de connexion.
* Le nombre de films par page pour la pagination.
* Le chemin d’accès à l’application java qui sera exécutée à partir du code PHP.

Structure du fichier :

*# Paramètres de connexion à la base de données Oracle*

*utilisateur=xmlp2*

*mdp=mlp42*

*path=iutdb*

*# pagination*

*# Nombre de films par page*

*nb\_pagination=4*

*#chemin d accès de l'application java*

*java\_path=chemin d’accès*

Méthode d’exploitation :

Le parcours du fichier se fait par le biais de la fonction « file() » qui permet l’ouverture du fichier puis par « foreach » utilisé pour le parcours des lignes. Nous utilisons la fonction « explode() » qui coupe une chaine de caractère afin de retourner uniquement l’élément de la chaine recherchée.

EXEMPLE :

*public static function configuration($element){*

*/\*-------------test de lecture du fichier ini----------------\*/*

*$fichier = 'conf.ini';*

*//ouverture du fichier*

*if(file\_exists($fichier) && $fichier\_lecture=file($fichier)){*

*//parcours des lignes pour trouver l’élément recherché*

*foreach($fichier\_lecture as $ligne){*

*//récupération de l’élément*

*$var = explode("=", $ligne);*

*if ($var[0] == $element){*

*return $var[1];*

*}*

*}*

*} else { //après erreur d’ouverture du fichier*

*echo 'Erreur d\'ouverture et/ou de lecture du fichier conf.ini<br/>';*

*}*

*/\*-------------------------------------------------------------\*/*

*}*

Appel de la fonction :

*include\_once 'Mutilitaire.php'; //inclus la fonction*

*Mutilitaire::configuration('utilisateur'); //utilisateur est l’élément recherché*

Remarque 1: le parcours d’un fichier peut être inadapté si ce dernier est trop long nuisant aux performances de l’application. Il faut donc réfléchir à son utilisation. Dans notre cas, c’est une solution adapté car le fichier en question est assez court.

Remarque 2 : La publication des identifiants dans un fichier ini peut causer des soucis de sécurité une fois mis en ligne. En effet, on peut accéder au fichier en entrant son URL dans la barre d’adresse. Par conséquent il est important de prendre des mesures afin de protéger le fichier. L’utilisation d’un fichier HTaccess semble donc judicieuse pour empêcher ce genre de problèmes car il restreint les droits d’accès du fichier à protéger. Par manque de temps nous n’établirons pas cette protection mais il serait très envisageable de la mettre en place dans le futur car cette faille est non-négligeable.

Autre solution : Il est aussi possible d’utiliser la méthode « fgets ()» pour la lecture des lignes d’un fichier. Dans ce cas, il faut utiliser la fonction « fopen » et non pas « file() » pour ouvrir le fichier.

### La pagination

La pagination est un élément principal de l’application car elle garantie un meilleur confort de navigation pour son utilisateur d’autant plus que le nombre de films est très élevé.

Le principe est très simple. Après exécution de la requête le résultat (tableau) comportant tous les films est stocké dans une variable session. Le nombre de films autorisé par page est cherché dans le fichier « conf.ini ». Le numéro de la page recherchée est passé en paramètre dans l’URL. Ensuite ne sont sélectionnés que les films de cette page dans le tableau.

Exemple : (nombre total d’élément à afficher / nombre de films par pages)

=> Résultat arrondi à l’unité supérieure

*$elements\_totaux = 9 ;*

*$nombre\_par\_page = 2 ;*

*//ceil() permet d’arrondir à l’unité supérieur ;*

*//ne pas utiliser round() ;*

*$nb\_pages = ceil($elements\_totaux/$nombre\_par\_page) ;*

*//affichage des pages*

*if ($nbpages > 1 ){ for ($i=1; $i<=$nb\_pages; $i++){ echo $i ;*

*}*

*}*

Inconvénients : Cette méthode nécessite la récupération de tous les résultats, non seulement ceux de la page sélectionnée. La meilleure solution serait de faire une requête pour chaque nouvelle page en utilisant la contrainte SQL « LIMIT » où l’on limiterait le nombre de résultats.

Point positifs : Une fois le chargement fait, l’affichage est instantané puisqu’aucune autre requête n’est à effectuer

### L’Ajax

Une partie de l’application est réalisée en Ajax afin de garantir plus de fluidité à la navigation. En effet, le champ de recherche muni d’un script Ajax permet d’afficher et de mettre à jour continuellement la liste des films après chaque nouvelle lettre entrée dans le champ. Ainsi, l’utilisateur n’est pas obligé de valider le formulaire par un bouton « valider » et voit les résultats s’afficher en direct.

Le principe de ce script est simple, après chaque nouvelle entrée ce dernier appelle une page PHP à exécuter. Dans notre cas, il effectue une requête et en affiche le résultat grâce à l’insertion d’un modèle et d’une vue adéquate.

Champ de recherche simple

Script AJAX

(S’exécute à chaque nouvelle entrée)

Page PHP

-inclus modèle

-inclus vue

### Extraction XML

Certaines données étant au format XML grâce au type XMLtype dans Oracle, il est nécessaire d’utiliser du Xpath afin de pouvoir extraire chaque commentaire de film stocké dans une hiérarchie prévue à cet effet.

Nous utilisons l’extraction dans une requête qui est la suivante :

*$stmt = ociparse($connect,"*

*SELECT f.TITRE, f.DUREE, f.ANNEE, f.RESUME, f.NOTES\_SITE, f.NOTES\_UTIL, //utilisation du XPATH pour extraire les commentaires stockés sous XML*

*f.COMMENTAIRE.extract('/film/commentaire/com') as COMMENTAIRE, f.COMMENTAIRE.extract('/film/commentaire/pseudo') as PSEUDO, f.COMMENTAIRE.extract('/film/commentaire/note') as NOTE, f.COMMENTAIRE.extract('/film/commentaire/source') as SOURCE FROM FILM f WHERE f.ID\_FILM='".$id."'"); ociexecute($stmt,OCI\_DEFAULT);*

### Exécution d’une application java via PHP.

L’application de restitution est liée à l’application JAVA mais est invisible aux yeux de l’utilisateur. En effet, l’ajout d’un film se fait par le biais d’un formulaire simple dans lequel est entré le nom du film à rechercher. Après validation, le programme java est lancé et effectue la récupération des données.

L’exécution de l’application Java se fait avec une fonction PHP déjà existante « exec() »

Voici le code PHP permettant d’exécuter le programme java :

*<?php*

*class Java {*

*public static function resultats ($data){*

*//récupération du chemin dans le fichier conf.ini*

*$path = Mutilitaire::configuration('java\_path');*

*echo "<div style='width:400px;'>";*

*//teste si java et installé*

*if (shell\_exec(‘java’)){*

*//vérifie si le formulaire est vide*

*if (empty($data)){*

*echo "< strong>Votre entrée n'est pas valide !!!</strong>";*

*} else { //sinon*

*//exécution de l’application java*

*if ($output = shell\_exec($path.'/'.$data )){*

*echo "<strong>Commande exécutée : ".$data."</strong><br/>";*

*echo $output."<br/><br/><br/><br/></div>";*

*} else {*

*echo "< strong>Erreur, vérifiez le path dans le fichier conf.ini</strong>";*

*}*

*}*

*} else { //Si java n’est pas installé…*

*echo "<strong>Votre ordinateur nécéssite l'installation de l'environnement Java</strong><br/>*

*<span style='font-size:10px;'>(Le téléchargement est disponible <a href='http://www.java.com/fr/download/index.jsp'>>>ici<<</a>)</span>";*

*}*

*}*

*}*

*?>*

Ajout d’une film via Java

1. Conclusion

Pour conclure, le projet s’est déroulé sous une bonne entente de l’équipe et son enthousiasme. La répartition du travail a été bénéfique à l’évolution du projet.

Lors de ces deux semaines de projet nous avons rencontrés des problèmes qui nous ont bloqués à certains moments mais qui ne sont pas tous rester irrésolus. Malgré tout, nous avons réussi à créer une application plus ou moins fonctionnelle et qui serait un bon départ pour être améliorée.

L’utilisateur pourra jouir d’une ergonomie, d’une souplesse et d’une fiabilité de l’application.

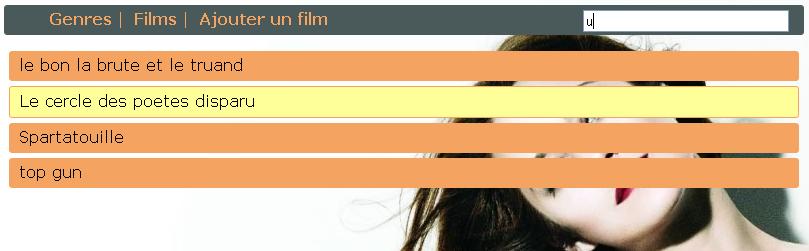
Dû à des soucis de délais, tous les objectifs n’ont pas pu être atteints mais le projet peut être repris à des fins d’améliorations.

Plus tard, l’utilisation de vues et de triggers pourraient améliorer le fonctionnement de cette application. Ainsi que la possibilité de rajouter des sites à comparer pour avoir encore plus d’informations afin d’alimenter la base de données.

ANNEXES



Page genre (liste des genres disponibles)



Liste des films après sélection d’un critère



Détails d’un film



Commentaire d’un film



Notation par système d’étoiles

Code source java :

/\*

\* Personne.java 14 févr. 2011

\* Projet\_XML\_recuperation

\*/

**package** lpd2i.recuperation;

**import** java.util.ArrayList;

/\*\*

\* Classe contenant les informations d'une personne

\* **@author** Xavier Mourgues

\* **@version** 0.1

\*/

**public** **class** Personne {

/\*\*

\* prenom de la personne

\* **@uml.property** name="prenom"

\*/

**private** String prenom;

/\*\*

\* nom de la personne

\* **@uml.property** name="nom"

\*/

**private** String nom;

/\*\*

\* liste des films realise

\* **@uml.property** name="realise"

\*/

**private** ArrayList<Film> realise;

/\*\*

\* liste des films dont la personne est actrice

\* **@uml.property** name="joue"

\*/

**private** ArrayList<Film> joue;

/\*\*

\* **TODO** Commenter le constructeur

\* **@param** string

\* **@param** string2

\*/

**public** Personne(String prenom, String nom) {

**this**.prenom = prenom;

**this**.nom = nom;

}

/\*\*

\* **TODO** Commenter le constructeur

\* **@param** nodeValue

\*/

**public** Personne(String prenomnom) {

String text[] = prenomnom.split(" ");

**this**.prenom = text[0];

**this**.nom = text[1];

**for**(**int** i = 2; i<text.length; i++){

**this**.nom = **this**.nom + " " + text[i];

}

}

/\*\*

\* méthode de stockage d'un acteur ou réalisateur dans la base

\* **TODO**

\*/

**public** **void** save(){

// verifier que la personne n'y soit pas déja

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*/

@Override

**public** String toString() {

**return** "Personne [prenom=" + prenom + ", nom=" + nom +"]";

}

}

/\*

\* Film.java 14 févr. 2011

\* Projet\_XML\_recuperation

\*/

**package** lpd2i.recuperation;

**import** java.util.ArrayList;

/\*\*

\* Classe gérant les informations d'un film

\*

\* **@author** Xavier Mourgues

\* **@version** 0.1

\*/

**public** **class** Film {

/\*\*

\* Titre du film

\* **@uml.property** name="titre"

\*/

**private** String titre;

/\*\*

\* Durée du film

\* **@uml.property** name="duree"

\*/

**private** **int** duree;

/\*\*

\* Année de sortie du film

\* **@uml.property** name="annee"

\*/

**private** **int** annee;

/\*\*

\* Note moyenne donnée par les site

\* **@uml.property** name="noteSite"

\*/

**private** **float** noteSite;

/\*\*

\* Résumé du film

\* **@uml.property** name="resume"

\*/

**private** String resume;

/\*\*

\* Genres du film

\* **@uml.property** name="genres"

\* **@uml.associationEnd** multiplicity="(0 -1)" elementType="java.lang.String"

\*/

**private** ArrayList<String> genres;

/\*\*

\* Commentaires du film

\* **@uml.property** name="commentaires"

\*/

**private** ArrayList<Commentaire> commentaires;

/\*\*

\* Liste des acteurs

\* **@uml.property** name="acteurs"

\*/

**private** ArrayList<Personne> acteurs;

/\*\*

\* Liste des réalisateurs

\* **@uml.property** name="realisateurs"

\* **@uml.associationEnd** multiplicity="(0 -1)" elementType="lpd2i.recuperation.Personne"

\*/

**private** ArrayList<Personne> realisateurs;

**public** Film(){

**this**.acteurs = **new** ArrayList<Personne>();

**this**.realisateurs = **new** ArrayList<Personne>();

**this**.genres = **new** ArrayList<String>();

**this**.commentaires = **new** ArrayList<Commentaire>();

}

/\*\*

\* **@return** the titre

\* **@uml.property** name="titre"

\*/

**public** String getTitre() {

**return** titre;

}

/\*\*

\* **@param** titre the titre to set

\* **@uml.property** name="titre"

\*/

**public** **void** setTitre(String titre) {

**this**.titre = titre;

}

/\*\*

\* **@return** the duree

\* **@uml.property** name="duree"

\*/

**public** **int** getDuree() {

**return** duree;

}

/\*\*

\* **@param** duree the duree to set

\* **@uml.property** name="duree"

\*/

**public** **void** setDuree(**int** duree) {

**this**.duree = duree;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** parseInt

\* **@uml.property** name="annee"

\*/

**public** **void** setAnnee(**int** annee) {

**this**.annee = annee;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** f

\* **@uml.property** name="noteSite"

\*/

**public** **void** setNoteSite(**float** note) {

**this**.noteSite = note;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** nodeValue

\* **@uml.property** name="resume"

\*/

**public** **void** setResume(String resume) {

**this**.resume = resume;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** nodeValue

\*/

**public** **void** addGenre(String genre) {

genres.add(genre);

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*/

@Override

**public** String toString() {

StringBuffer str = **new** StringBuffer();

str.append("Film [titre=" + titre + ", duree=" + duree + ", annee=" + annee

+ ", noteSite=" + noteSite + ", resume=" + resume + ", genres=");

**for**(String genre : genres){

str.append("," +genre);

}

str.append(", commentaires=");

**for**(Commentaire com : commentaires){

str.append(com);

}

str.append(", acteurs=");

**for**(Personne acteur : acteurs){

str.append(acteur);

}

str.append(", realisateurs=");

**for**(Personne realisateur : realisateurs){

str.append(realisateur);

}

str.append("]");

**return** str.toString();

}

/\*\*

\* méthode stockant les informations du film dans la base de donnée

\* **TODO**

\*/

**public** **void** save(){

// verifier que le film n'existe pas déjà

// stocker les information de base (titre, durée, etc..)

// stocker les acteurs, realisateur et genre

**for**(Personne e : acteurs){

e.save();

}

**for**(Personne e : realisateurs){

e.save();

}

// stocker les commentaires

// stocker les liens (acteur/film, realisateur/film, genre/film)

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** personne

\*/

**public** **void** addRealisateur(Personne personne) {

**this**.realisateurs.add(personne);

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** parseActorsContent

\*/

**public** **void** setActeurs(ArrayList<Personne> acteurs) {

**this**.acteurs = acteurs;

}

}

/\*

\* Film.java 14 févr. 2011

\* Projet\_XML\_recuperation

\*/

**package** lpd2i.recuperation;

**import** java.util.ArrayList;

/\*\*

\* Classe gérant les informations d'un film

\*

\* **@author** Xavier Mourgues

\* **@version** 0.1

\*/

**public** **class** Film {

/\*\*

\* Titre du film

\* **@uml.property** name="titre"

\*/

**private** String titre;

/\*\*

\* Durée du film

\* **@uml.property** name="duree"

\*/

**private** **int** duree;

/\*\*

\* Année de sortie du film

\* **@uml.property** name="annee"

\*/

**private** **int** annee;

/\*\*

\* Note moyenne donnée par les site

\* **@uml.property** name="noteSite"

\*/

**private** **float** noteSite;

/\*\*

\* Résumé du film

\* **@uml.property** name="resume"

\*/

**private** String resume;

/\*\*

\* Genres du film

\* **@uml.property** name="genres"

\* **@uml.associationEnd** multiplicity="(0 -1)" elementType="java.lang.String"

\*/

**private** ArrayList<String> genres;

/\*\*

\* Commentaires du film

\* **@uml.property** name="commentaires"

\*/

**private** ArrayList<Commentaire> commentaires;

/\*\*

\* Liste des acteurs

\* **@uml.property** name="acteurs"

\*/

**private** ArrayList<Personne> acteurs;

/\*\*

\* Liste des réalisateurs

\* **@uml.property** name="realisateurs"

\* **@uml.associationEnd** multiplicity="(0 -1)" elementType="lpd2i.recuperation.Personne"

\*/

**private** ArrayList<Personne> realisateurs;

**public** Film(){

**this**.acteurs = **new** ArrayList<Personne>();

**this**.realisateurs = **new** ArrayList<Personne>();

**this**.genres = **new** ArrayList<String>();

**this**.commentaires = **new** ArrayList<Commentaire>();

}

/\*\*

\* **@return** the titre

\* **@uml.property** name="titre"

\*/

**public** String getTitre() {

**return** titre;

}

/\*\*

\* **@param** titre the titre to set

\* **@uml.property** name="titre"

\*/

**public** **void** setTitre(String titre) {

**this**.titre = titre;

}

/\*\*

\* **@return** the duree

\* **@uml.property** name="duree"

\*/

**public** **int** getDuree() {

**return** duree;

}

/\*\*

\* **@param** duree the duree to set

\* **@uml.property** name="duree"

\*/

**public** **void** setDuree(**int** duree) {

**this**.duree = duree;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** parseInt

\* **@uml.property** name="annee"

\*/

**public** **void** setAnnee(**int** annee) {

**this**.annee = annee;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** f

\* **@uml.property** name="noteSite"

\*/

**public** **void** setNoteSite(**float** note) {

**this**.noteSite = note;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** nodeValue

\* **@uml.property** name="resume"

\*/

**public** **void** setResume(String resume) {

**this**.resume = resume;

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** nodeValue

\*/

**public** **void** addGenre(String genre) {

genres.add(genre);

}

/\* (non-Javadoc)

\* @see java.lang.Object#toString()

\*/

@Override

**public** String toString() {

StringBuffer str = **new** StringBuffer();

str.append("Film [titre=" + titre + ", duree=" + duree + ", annee=" + annee

+ ", noteSite=" + noteSite + ", resume=" + resume + ", genres=");

**for**(String genre : genres){

str.append("," +genre);

}

str.append(", commentaires=");

**for**(Commentaire com : commentaires){

str.append(com);

}

str.append(", acteurs=");

**for**(Personne acteur : acteurs){

str.append(acteur);

}

str.append(", realisateurs=");

**for**(Personne realisateur : realisateurs){

str.append(realisateur);

}

str.append("]");

**return** str.toString();

}

/\*\*

\* méthode stockant les informations du film dans la base de donnée

\* **TODO**

\*/

**public** **void** save(){

// verifier que le film n'existe pas déjà

// stocker les information de base (titre, durée, etc..)

// stocker les acteurs, realisateur et genre

**for**(Personne e : acteurs){

e.save();

}

**for**(Personne e : realisateurs){

e.save();

}

// stocker les commentaires

// stocker les liens (acteur/film, realisateur/film, genre/film)

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** personne

\*/

**public** **void** addRealisateur(Personne personne) {

**this**.realisateurs.add(personne);

}

/\*\*

\* **TODO** Commenter cette méthode

\* **@param** parseActorsContent

\*/

**public** **void** setActeurs(ArrayList<Personne> acteurs) {

**this**.acteurs = acteurs;

}

}