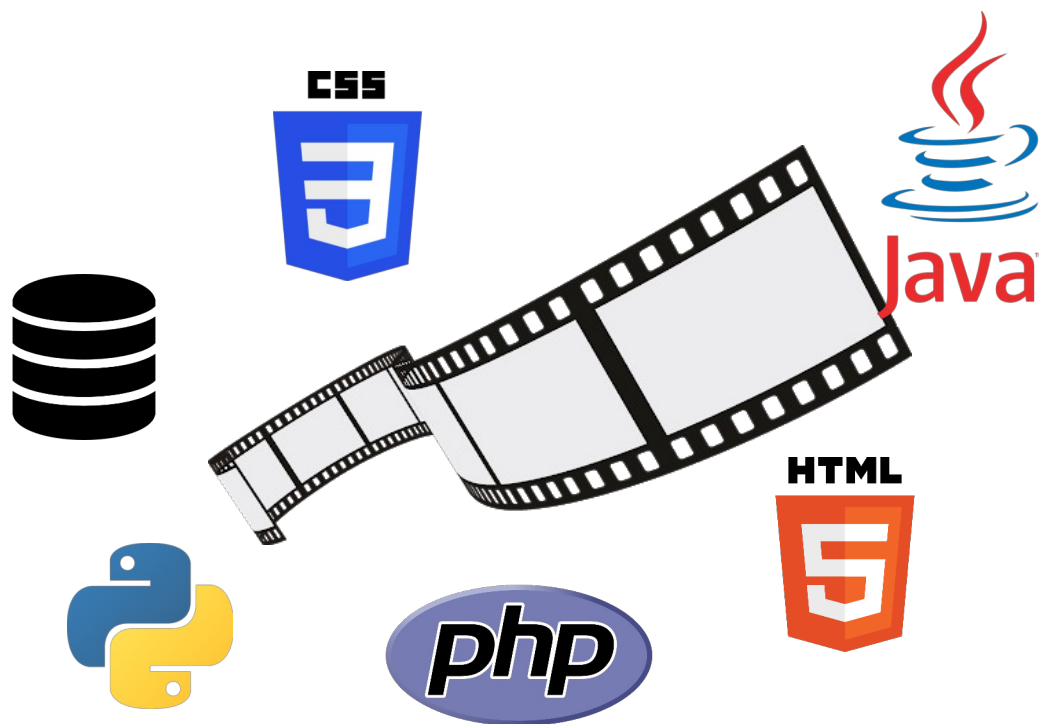


Compte rendu projet algorithmique IDU3 - 05/2022



Etudiants : Thomas NICOLAS, Vinent TAKAHASHI, Benjamin GUERiot, Arthur GONAY

Enseignant : Sorana CIMPAN, Ilham ALLOUI

Etablissement : Ecole Polytech Annecy-Chambéry

Section : Informatique Données Usage

Année : 2021/2022

Trello: <https://trello.com/b/Pj7xMKxc/projgoodfilms>

Github: https://github.com/Brocowlee/Proj_GoodFilms

Sommaire

Introduction.....	3
Gestion de projet.....	3
Cahier des charges.....	3
Liste des tâches (Trello).....	4
Planification des tâches (Gantt).....	4
Coordonner les activités du groupe projet.....	4
Fonctionnalités.....	5
Gestion de la base de données.....	5
Interface Java.....	9
Interface Web.....	19
Pour aller plus loin.....	28
Conclusion.....	28
Annexe.....	29
Table des acronymes.....	29
Table des outils.....	29

Introduction

Notre projet consiste en la réalisation d'un site répertoriant des films que l'on peut noter, commenter etc... avec un principe similaire à celui du site internet GoodReads, mais appliqué aux films. En plus de cela, une interface JAVA permet directement l'interaction avec la base de données.

Nous avons décidé de mener ce projet à 4 au lieu des 5-6 membres indiqués sur le sujet car peu de personnes de notre promo étaient intéressées par ce sujet. Cet effectif nous a imposé une importante rigueur quant à la gestion de projet, comme nous pourrons le voir dans la première partie de ce rapport.

Nous avons divisé le projet en 3 parties principales:

- La gestion et la récupération des données (Arthur)
- La création d'une interface java permettant l'interaction direct avec la base de données pour les administrateurs (Benjamin et Thomas)
- La création du site web ainsi que sa connexion avec la base de données (Vincent).

C'est aussi ce découpage que suivra la partie technique de ce rapport.

Gestion de projet

Cahier des charges

La première étape de ce projet a été la définition claire du cahier des charges, permettant de se répartir les tâches.

Objectif Principal	<ul style="list-style-type: none">- Réalisation d'une application web afin d'afficher différentes données- Réalisation d'un application java permettant de traiter les données- Création d'une base de données regroupant les informations nécessaires
Contraintes	<ul style="list-style-type: none">- Utiliser les langages JAVA / HTML / CSS / PHP / SQL / PYTHON- Mettre en relation plusieurs langages- Gestion de projet- Remplissage de la base de données à partir de données actuelles
Ressources allouées	<ul style="list-style-type: none">- Humaine : 4 personnes (sous effectif)- Matériel : Serveurs- Temporelle : ~1 mois
Fonctionnalités	<ul style="list-style-type: none">- Navigation dans un site web- Visualisation des films- Fonctions cercle d'amis- Gestion de commentaires- Gestion d'évaluation- Affichage de moyennes des notes (statistiques)- Consultation données utilisateurs
	<ul style="list-style-type: none">- Ajout de tables- Suppression de tables- Ajout de données- Suppression de données- Modification de données

Figure 1: Cahier des charges

Liste des tâches (Trello)

Un des outils que nous avons utilisés pour mener à bien ce projet a été Trello. Nous y avons répertorié toutes les subdivisions de nos tâches personnelles en y ajoutant la difficulté (étiquettes de couleur). Cet outil nous a permis de connaître à tout moment l'avancée de chacun.

Planification des tâches (Gantt)

Nous avons également mis en place un Gantt afin de planifier nos tâches dans le temps pour avoir une vue d'ensemble du projet et ainsi savoir ce qui doit être réalisé pour une date donnée.

Vous trouverez le Gantt dans le même dossier que ce rapport ainsi que sur reddit.

Coordonner les activités du groupe projet

Trello	Cela nous a permis de répertorier toutes les subdivisions de nos tâches personnelles. Cela a également permis à l'ensemble des membres de connaître l'avancée de tout un chacun à tout moment.
Gantt	Cela nous a permis de planifier nos tâches dans le temps afin d'avoir une vue d'ensemble sur ce qui doit être réalisé pour pouvoir mener ce projet à bien.
GitHub	GitHub nous a permis de gérer notre code, afin que chacun puisse travailler en simultané et apporter des modifications.
IDE (Eclipse, VS Code)	Ces IDE nous ont permis d'écrire notre code de manière simple et efficace grâce aux différentes aides ergonomiques présentes sur chacun d'eux.
Google Doc	Plateforme collaborative qui nous a permis de rédiger notre rapport de projet en simultané.
Google Slides	Plateforme collaborative qui nous a permis de mettre en place notre support de présentation orale en simultané.
Discord / Messenger	Cela nous a permis de communiquer entre nous afin de se partager diverses informations ainsi que l'avance de chacun. Nous avons également utilisé discord afin de faire des réunions de mise en commun et de présentations fonctionnelles de l'avancement du projet.

Figure 2: Outils de coordination

Fonctionnalités

Gestion de la base de données

Une des premières étapes après avoir fixé les objectifs que devait remplir le site a été la création de la base de données. Nous nous sommes ainsi mis d'accord sur les informations qui devaient y être présentes, et avons créé une première ébauche de ce qu'allait être notre structure de données. Avec la suite des étapes, celle-ci a beaucoup évolué notamment sur les types et tailles des différents attributs.

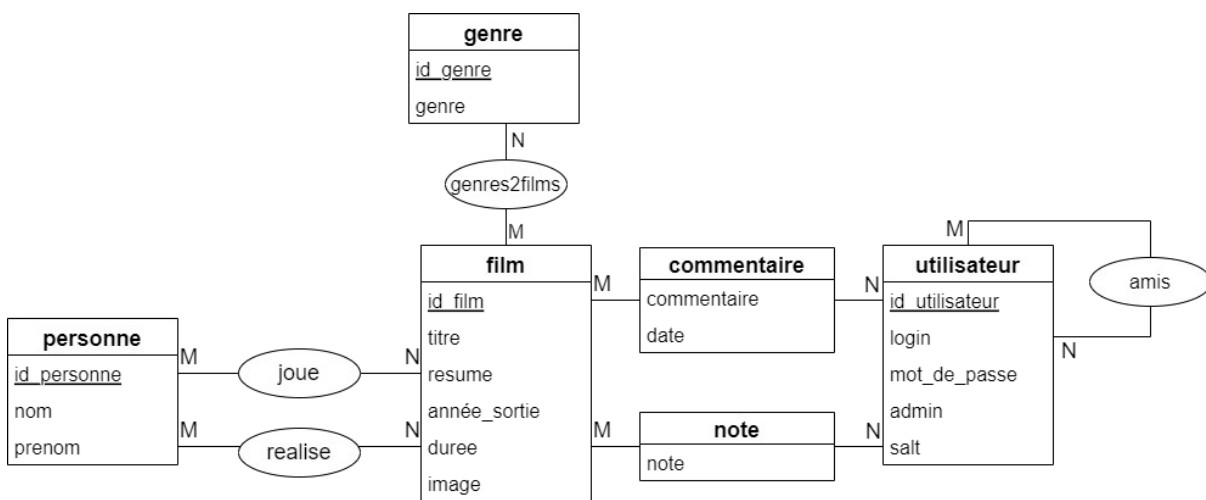


Figure 3: Modèle entité/association

Ensuite, nous nous sommes posé la question du remplissage de cette base de données. Dans un premier temps nous avons pensé utiliser une base de données CSV déjà existante. Cependant toutes celles que nous avons trouvé ne contenaient pas toutes les informations que nous avons défini dans notre modèle. Après avoir réfléchi à plusieurs solutions, c'est celle du web scraping que nous avons retenue, faisant ainsi un lien avec le module de projet data science de ce semestre. Le principal avantage de cette solution était qu'elle permettait une importante flexibilité sur la récolte des données. En effet, il existe de très nombreux sites internet répertoriant un grand nombre de films, et même si leur BDD n'est pas public, un script permet d'en récupérer une partie de ces données.

La première étape a été la sélection du site dont nous allions récupérer les données. Après en avoir examiné plusieurs, notre dévolu s'est porté vers *JustWatch*, un site permettant aux utilisateurs de connaître sur quelles plateformes de streaming certains contenus multimédia peuvent être visualisés.

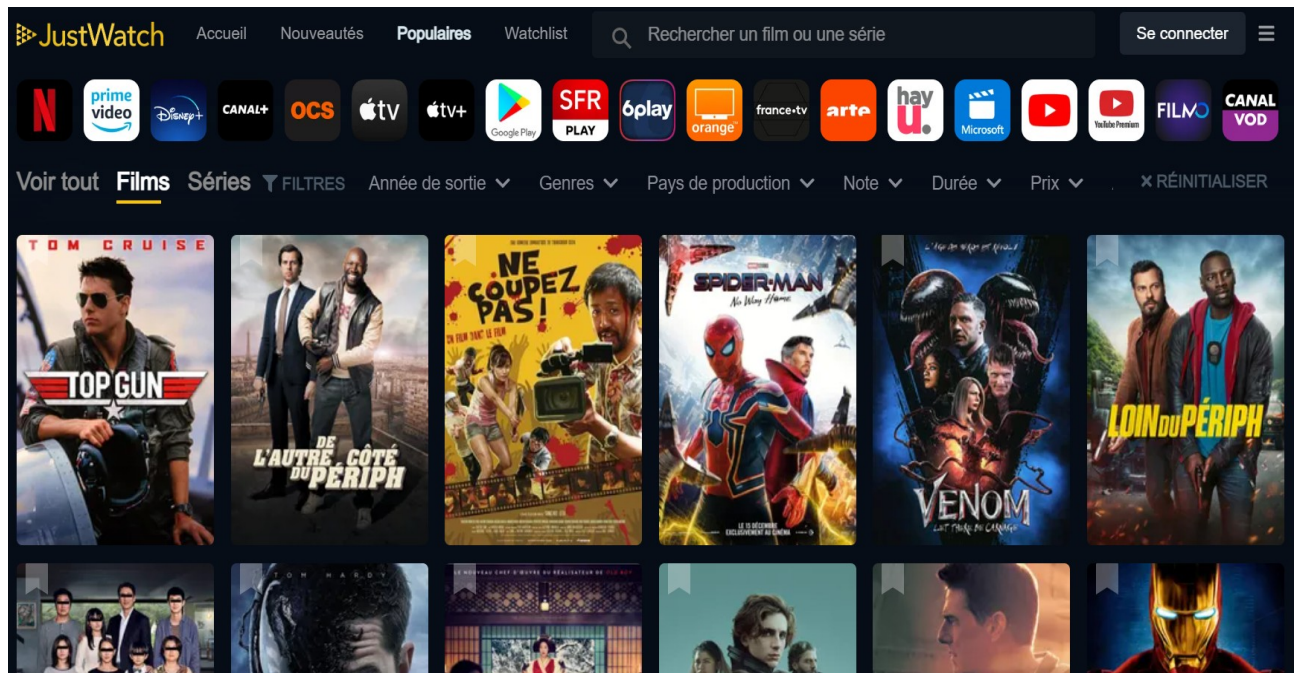


Figure 4: Site de JustWatch

Il a d'abord fallu récolter les liens des films les uns après les autres. Un problème que nous avons rencontré a été que la page charge les films au fur et à mesure et que par défaut, seulement 30 films sont chargés. Pour contourner ce problème, la première solution que nous avons explorée a été l'utilisation de sélénium pour charger dynamiquement la page. Cependant cette solution était compliquée à mettre en place et nous avons préféré contourner le problème en appliquant des filtres sur l'année de sortie des films. Ainsi 30 films étaient chargés chaque année, résolvant le problème dans le cadre de ce projet.

Pour le web scraping nous avons utilisé la bibliothèque BeautifulSoup sur Python. A partir de là, il fallait écrire les fonctions de récupération de chaque élément nous intéressant sur les pages (titre, date de sortie, image etc...).

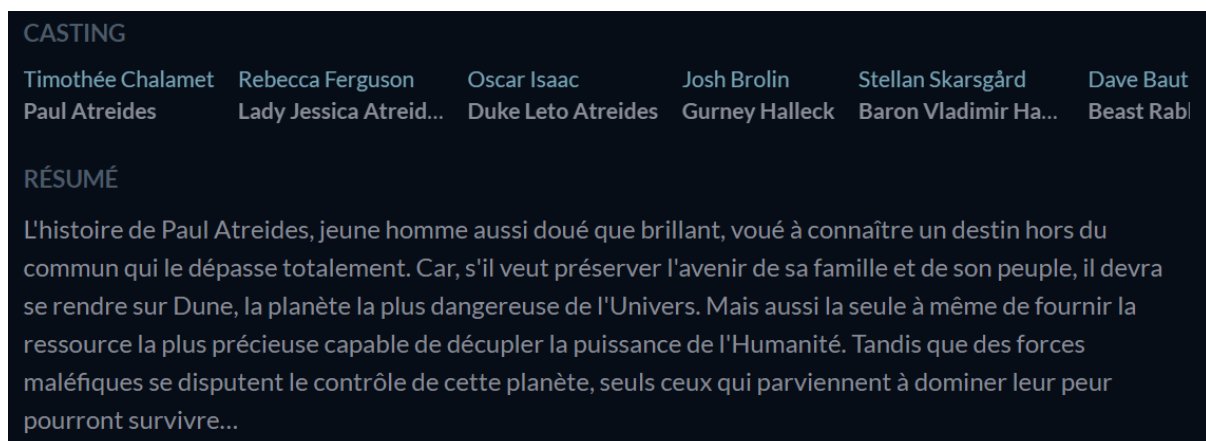
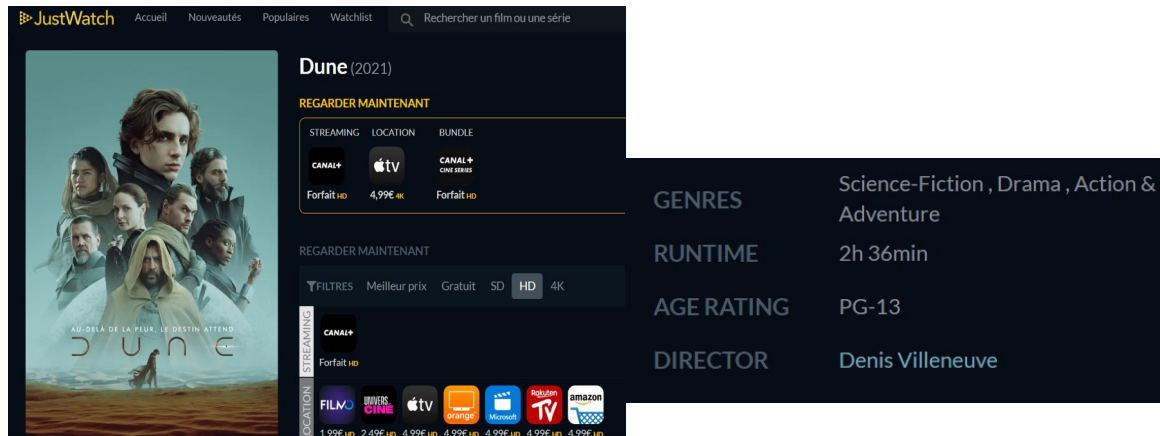


Figure 5: Informations d'un film sur JustWatch

Exemple de fonction de récupération :

```
#fonction de récupération des titres (string)
def getTitre(soup):
    return soup.find("h1").text[1:-1] #contient des espaces au début et à la fin

#fonction de récupération des années de sortie (string)
def getAnnee(soup):
    return soup.find("span", {"class": "text-muted"}).text[2:-2]

#fonction de récupération des durées des films
def getDuree(soup):
    return soup.find_all("div", {"class": "detail-infos__value"})[2].text
```

Figure 6: Fonction de récupération

La récupération des personnes et des genres était un peu plus complexe étant donné qu'il fallait faire attention à ne les créer qu'une seule fois. Pour gérer cela, des listes contenant les éléments déjà créés se mettaient à jour au fur et à mesure.

Il fallait ensuite s'occuper de la partie écriture des requêtes. Pour cela nous écrivons dans un fichier texte l'ensemble des requêtes, complétées par les informations récupérées sur le site via les méthodes. Il est important de noter que les tables étaient alors déjà créées et le seul objectif du code python était leur remplissage et la création du lien entre les différents éléments. Ci-dessous une partie du code permettant l'écriture des requêtes.

```
def initTxt(link):
    global id_f
    result=requests.get(link)
    soup = BeautifulSoup(result.text, 'html.parser')
    with open('films.txt', 'a', encoding="utf-8") as f:
        try:
            f.writelines('\nINSERT INTO Film(titre,resume,annee_sortie,duree,image) VALUES ("'+getTitle(soup)+'","'+getResume(soup)+'\n')
            lstFilms.append(link)
            idP=isAlreadyAddPers(soup, getReal(soup))
            if idP==None:
                lstPers.append(getReal(soup))
                prenom=getReal(soup).split(" ")[1]
                nom=getReal(soup).split(" ")[2]
                f.writelines('INSERT IGNORE INTO Personne(nom,prenom) VALUES ("'+nom+'","'+prenom+'");\n')
                f.writelines("INSERT INTO realise(id_Film, id_Personne) VALUES ("'+str(len(lstFilms))+"','"+str(len(lstPers))+"');\n")
            else:
                f.writelines("INSERT INTO realise(id_Film, id_Personne) VALUES ("'+str(len(lstFilms))+"','"+str(idP)+"');\n")
```

Figure 7: Exemple d'écriture de requêtes

Un `try` englobe la création des requêtes car pour de nombreuses raisons certaines erreurs peuvent apparaître. Pour éviter que cela pose problème, le `try` empêche la création des requêtes en cas d'erreur, sans arrêter le programme.

Cette gestion des erreurs a d'abord posé problème car dans les premières versions du code, les insertions n'étaient pas en auto-increment afin de récupérer les ids des films et ainsi associer les éléments entre eux. Cela avait pour effet de sauter certains id et de rendre plus complexe la mise en auto-increment pour la suite de l'utilisation de la base de données.

Nous avons donc changé de méthode, et avons opté pour une version où une liste contenant les films ajoutés est mise à jour afin de récupérer l'id et faire le liens avec les autres tables personnes et genres.

Beaucoup de modifications ont dû être apportées afin que les prêts de 6000 requêtes SQL s'exécutent toutes correctement. En effet, certaines tailles de `VARCHAR` était trop faible ou certains caractères posaient des erreurs d'encodages.

Interface Java

Le but de cette partie est de créer une interface graphique qui permettra à l'administrateur d'agir sur la base de données sans passer par un tiers comme PhpMyAdmin.

Nous avons ici pour but d'implémenter des fonctionnalités qui permettront à l'utilisateur de notre site de :

- ajouter / supprimer des tables
- ajouter/ supprimer / modifier les données d'une table

Nous allons par la suite expliquer la partie fonctionnelle de l'interface Java, pour plus d'informations et de détails sur le code, une JavaDoc est disponible sur le dépôt GitHub.

Au lancement de l'application nous avons besoin d'une fenêtre de connexion afin de vérifier si la personne utilisant cette application est bien un administrateur.


A screenshot of a login panel. It features a light gray background. On the left, the labels 'Identifiant :' and 'Mot de passe :' are displayed in a dark blue font. To the right of each label is a white rectangular text input field with a thin blue border. Below these two input fields is a single, wider button with a blue gradient and the text 'Connexion' in white.

Figure 11: Panel de connexion

Cette page est un JFrame contenant l'affichage du menu de connexion qui est un JPanel. Cela forme notre GUI. JFrame et JPanel sont des classes qui sont directement intégrées dans Java et qui servent à gérer l'affichage graphique d'une application. JPanel possède également des attributs permettant de gérer chaque éléments affichés sur un panel comme par exemple un JTextField permettant d'écrire un texte dans une case visible ci-dessus. Cette page JPanel est reliée aux autres pages grâce à un listener afin qu'à tout moment le bon panel soit affiché dans le GUI général.

Une petite précision quant à la connexion. Nous avons décidé de crypter les mots de passes dans la base de données lors de l'inscription via le site.

Nous avons donc utilisé l'API Bcrypt qui permet d'utiliser une fonction de hashage du même nom. Le principe est simple, le mot de passe va être hasher puis nous allons concaténer une chaîne de caractère générée aléatoirement avec une taille aléatoire également (Cette chaîne est appelée Salt). Cela permet de réduire les attaques basées sur les dictionnaires.

Nous obtenons alors un mot de passe crypté. Voici un exemple du résultat obtenu dans la base de données :

id_utilisateur	login	mot_de_passe	admin	salt
<input type="checkbox"/> 2	test	\$2a\$10\$8IYJu9EHPiYUhAsSwSjNo7mvBNBa9pMuJt3tzHya39ELE7juA.	1	\$2a\$10\$8IYJu9EHPiYUhAsSwSjNyYgQ
<input type="checkbox"/> 4	george	\$2a\$10\$8labAAFEf9c55/FUx.VCiuzxBC2yUliDr.XumPRV1e..01tM7shg\$	0	\$2a\$10\$8labAAFEf9c55/FUx.VCi5EA

Figure 12: Affichage de la table utilisateur

Les utilisateurs George et test ont tous deux le même mot de passe. Cependant, nous observons qu'il est crypté différemment.

Il est désormais temps de parler des données que nous récupérons dans notre BDD.

Nous allons d'abord initialiser nos tables déjà existantes avec les colonnes qu'elles contiennent.

Nous créons donc une nouvelle instance de TableObject qui est une classe regroupant toutes les informations d'une table ainsi que les méthodes qui concernent celle-ci.

```
public static void initTable() {  
    liste_tables.clear();  
  
    try {  
        DatabaseMetaData md = database.connexion.getMetaData();  
        String[] types = {"TABLE"};  
        ResultSet rs = md.getTables(null, null, "%", types);  
  
        while (rs.next()) {  
            if(!rs.getString("TABLE_NAME").contentEquals("sys_config")) {  
                TableObject table = new TableObject(rs.getString("TABLE_NAME")) ;  
                initColumn(table);  
            }  
        }  
  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Figure 13: Fonction d'initialisation des tables

Voici un exemple de méthode que nous pouvons retrouver dans la classe TableObject :

```
public ArrayList<List<String>> selectAll() {  
    ArrayList<List<String>> total_list = new ArrayList<>();  
  
    String QUERY = "SELECT * FROM " + this.name;  
  
    try {  
        ResultSet rs = Main.database.querySQL(QUERY);  
  
        while(rs.next()){  
            ArrayList<String> sublist = new ArrayList<>();  
  
            for(String name : this.informations.keySet()) {  
                sublist.add(getValueOf(rs, name));  
            }  
            total_list.add(sublist);  
        }  
    } catch (ClassNotFoundException | SQLException e) {  
        e.printStackTrace();  
    }  
  
    return total_list;  
}
```

Figure 14: Fonction qui sélectionne toutes les lignes d'une table

Cette méthode va récupérer l'intégralité des lignes présentes dans une table. C'est une opération similaire à `select *` en sql.
A noter que les clés de la HashMap *informations* correspondent aux noms des colonnes.
Beaucoup d'autres fonctions sont présentes dans cette classe mais il est plus simple de se référer à la JavaDoc.

Retournons sur la partie interface graphique.

Une fois l'administrateur connecté, le JPanel contenant l'affichage des tables est affiché. Celui-ci est composé de la liste des tables qui est quant à elle composée de boutons cliquables créés dans une classe à part entière. Ces boutons permettent l'affichage de ces dites tables. Ce Panel est également composé de plusieurs boutons d'options ainsi que d'un menu.

id_film	titre
1	Loin du périph
2	Choose or Die
3	The Northman
4	Bubble
5	The Contractor
6	Gold
7	365 Jours : Au lendemain
8	Roudram Ranam Rudhiram (RRR)
9	Adam à travers le temps
10	La Ruse
11	Le Couteau par la lame
12	Senior Year
13	X
14	Silverton Siege
15	En corps
16	Metal Lords
17	Le Mystère Marilyn Monroe : Conversations inédites
18	Eaux Profondes
19	Fresh
20	En attendant Bojangles
21	Crush
22	Spider-Man: No Way Home
23	Venom: Let There Be Carnage
24	Dune
25	Barbaque
26	Matrix Resurrections
27	Les Éternels
28	Free Guy

Figure 15: Affichage de la table film

Il n'était pas chose aisée de bien aligner les colonnes entre elles. Pour ce faire, nous avons réalisé un algorithme un peu « rustre » qui consistait à ajouter des espaces selon la taille maximale d'une donnée présente dans la même colonne.

Ensuite, nous avons fait de même pour les titres des colonnes sauf que nous avons ajouter la taille maximale / 2 à l'espacement afin de pouvoir placer le titre au milieu de la colonne.

Ceci n'a été réalisé que pour l'esthétique de l'interface mais nous voulions souligner la difficulté que c'était pour le mettre en place notamment dû au fait que les longueurs des chaînes de caractères devaient être calculées en pixels.

Abordons désormais les boutons d'options. Nous pouvons retrouver des boutons supprimer et ajouter permettant respectivement de supprimer toutes les lignes sélectionnées grâce aux checkbox de chaque ligne ou d'ajouter une nouvelle ligne à cette table.

Voici un exemple de la méthode de la classe TableObject appelée dans un de ces cas :

```
public void deleteline(int index) {
    try {
        Main.database.updateSQL("DELETE FROM " + this.name + " WHERE " + this.informations.keySet().toArray()[0] + " = " + index + ";");
    } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    }
}
```

Figure 16: Fonction qui supprime une ligne

Il existe également pour chaque ligne un bouton *éditer* représenté par un icon de crayon permettant la modification des données d'une ligne.

```
public void updateInfo(int id, List<Integer> indexes, ArrayList<String> value) {
    String request = "UPDATE " + this.name + " SET";
    for(int i = 0; i < indexes.size(); i++) {
        int index = indexes.get(i);
        String column_name = (String) this.informations.keySet().toArray()[index];
        if(!(indexes.get(indexes.size() - 1) == index)) {
            request += " " + column_name + " = '" + value.get(i) + "',";
        } else {
            request += " " + column_name + " = '" + value.get(i) + "' WHERE " + this.informations.keySet().toArray()[0] + " = " + id + ";";
        }
    }
    try {
        Main.database.updateSQL(request);
    } catch (ClassNotFoundException | SQLException e) {
        e.printStackTrace();
    }
}
```

Figure 17: Fonction qui met à jour une donnée

En outre, le menu permet d'ouvrir une nouvelle fenêtre ce qui donne la possibilité de travailler sur deux tables en simultanée, créer une toute nouvelle table, supprimer la table actuellement affichée, ou bien de se rendre sur le GitHub ou le Trello.

Pour l'ajout d'une table, un nouveau JPanel s'affiche dans le GUI général, cet affichage permet d'ajouter un nombre infini de colonnes à cette table ainsi que de lui donner un nom.

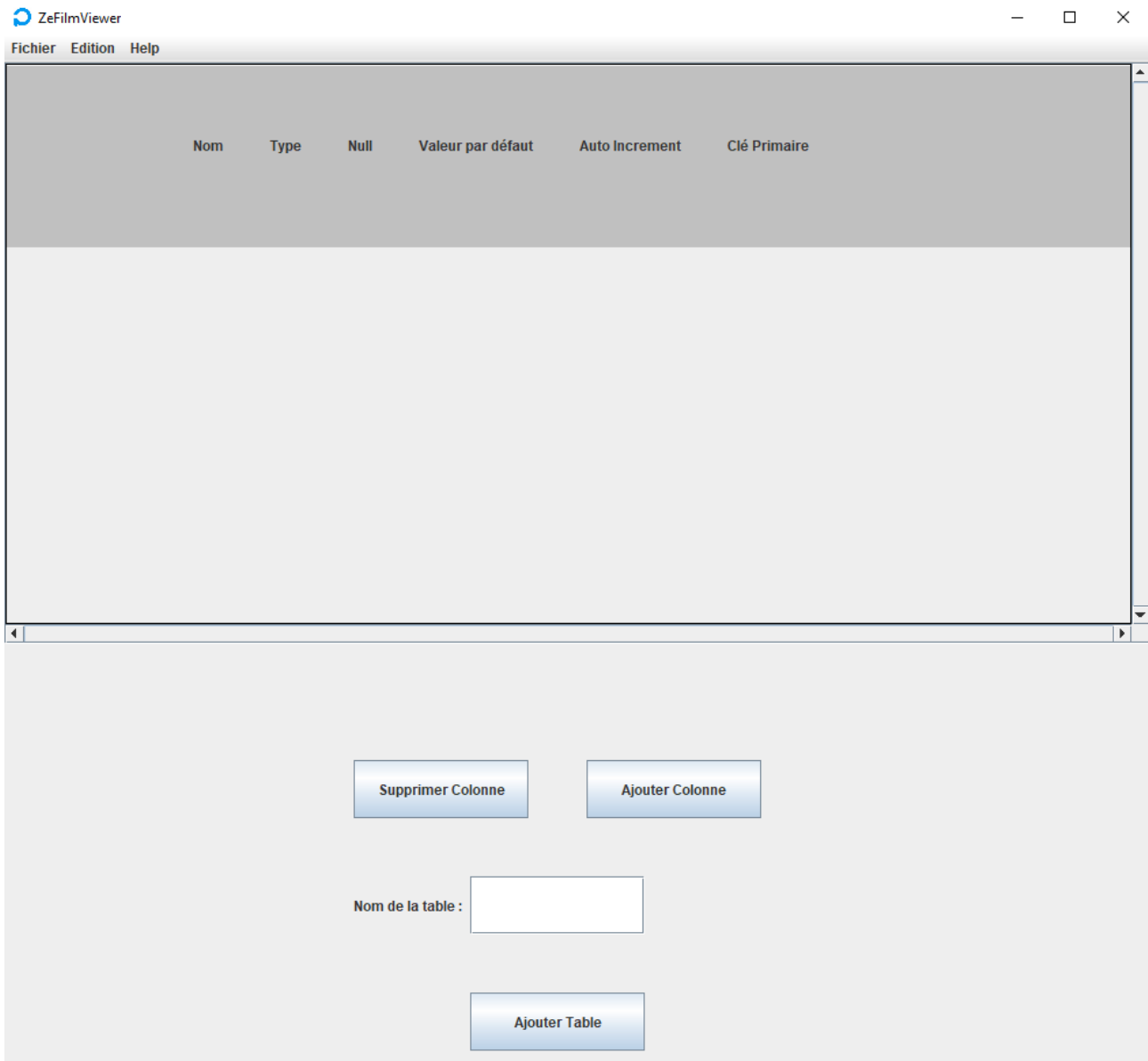
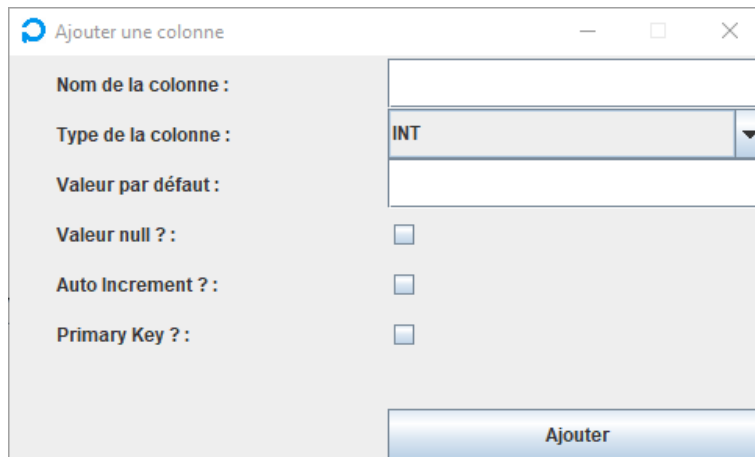


Figure 18: Fenêtre de création de tables

Chaque colonne est définie par plusieurs paramètres qui sont eux-mêmes définis par l'utilisateur dans un JFrame s'ouvrant lors de l'appui du bouton *Ajouter colonne*.



Ajouter une colonne

Nom de la colonne :

Type de la colonne : **INT** ▼

Valeur par défaut :

Valeur null ? : ☐

Auto Increment ? : ☐

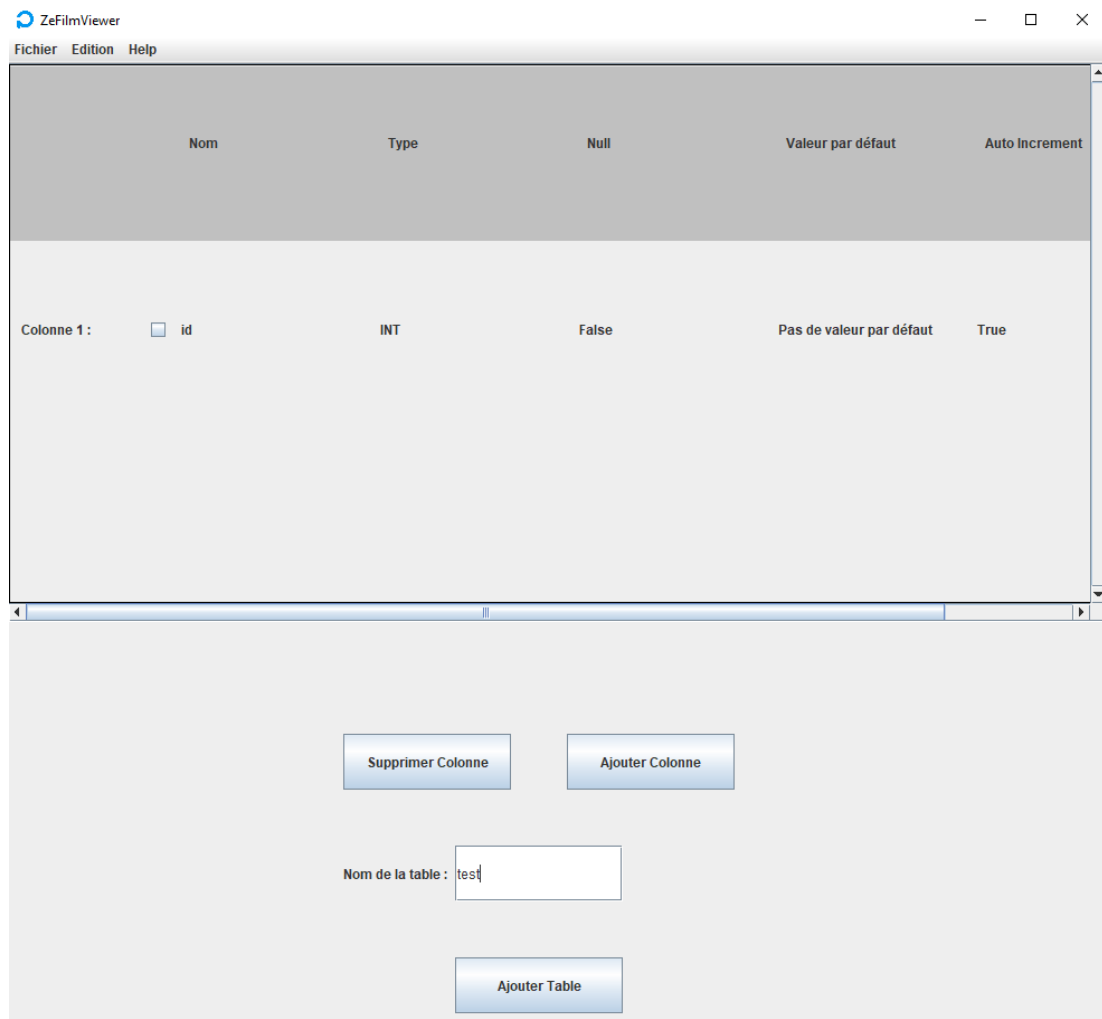
Primary Key ? : ☐

Ajouter

Figure 19: Fenêtre de création de colonnes

Ce JFrame permet de choisir le nom, le type, la valeur par défaut si nécessaire, ainsi que d'autres paramètres.

Une fois ajoutée, la colonne s'affiche dans la création de la table :



ZeFilmViewer

Fichier Edition Help

	Nom	Type	Null	Valeur par défaut	Auto Increment
Colonne 1 :	<input type="checkbox"/> id	INT	False	Pas de valeur par défaut	True

Supprimer Colonne **Ajouter Colonne**

Nom de la table :

Ajouter Table

Figure 20: Fenêtre de création de tables contenant une colonne

Une fois qu'au moins une colonne à été ajoutée, la table peut être insérée dans la base de données et sera donc affichée sur le JPanel principal.

Pour l'ajout des tables, nous avons créé une classe TableBuilder qui va « transformer » les attributs renseignés via le JPanel en requête pour pouvoir ajouter cette nouvelle table dans la BDD.

```
public String writeRequest(List<Column> list_columns) {
    ArrayList<String> primary_keys = new ArrayList<>();
    String primary = "";
    String request = "CREATE TABLE IF NOT EXISTS `" + this.name + "` ( ";

    for(Column column : list_columns) {

        request += "`" + column.getName() + "` " + column.getType();

        if(column.hasDefaultValue()) {
            request += " DEFAULT '" + column.getDefaultValue() + "'";
        }
        if(column.isNotNull()) {
            request += " NOT NULL";
        }
        if(column.isAutoIncrement()) {
            request += " AUTO_INCREMENT";
        }
        if(column.isPrimary()) {
            primary_keys.add(column.getName());
        }
        if(!(list_columns.get(list_columns.size() - 1) == column)) {
            request += ", ";
        } else {
            request += " ";
        }
    }

    for(String prim : primary_keys) {

        if(!(primary_keys.get(primary_keys.size() - 1) == prim)) {
            primary += "`" + prim + "` , ";
        } else {
            primary += "`" + prim + "`";
        }
    }

    String end_request = "";
    if(primary.isBlank()) {
        end_request = ") ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;" ;
    } else {
        end_request = ", PRIMARY KEY (" + primary + ") ) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;" ;
    }

    String final_request = request += end_request;

    return final_request;
}
```

Figure 21: Fonction de transformation d'attributs en requête

Ainsi qu'une fonction build qui va envoyer la requête au serveur :

```
public void build() {  
    try {  
        Main.database.updateSQL(writeRequest(list_columns));  
    } catch (ClassNotFoundException | SQLException e) {  
        e.printStackTrace();  
    }  
}
```

Figure 22: Fonction pour insérer une nouvelle table dans la BDD

Afin de prévenir l'utilisateur en cas d'erreur, quelque soit l'erreur, nous avons créé un fenêtré spéciale afin d'afficher le message d'erreur.

Ici, le nom de table n'est pas renseigné au moment de l'ajout ainsi cette erreur s'affiche :

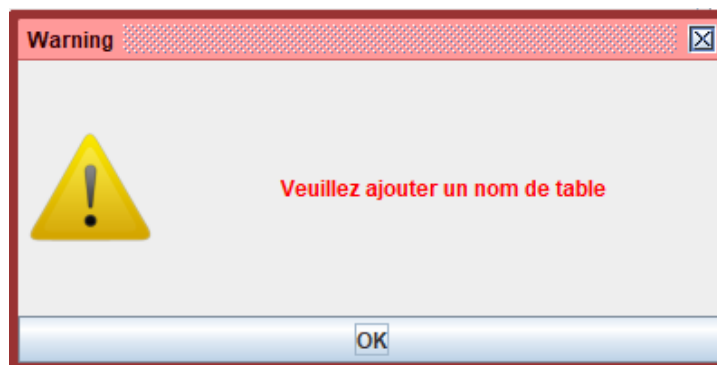


Figure 23: Fenêtré d'erreur

Interface Web

Le but de cette partie est de créer une interface qui permettra à l'utilisateur de noter et commenter ses films préférés à la manière du site GoodReads.com. La base de données des films étant déjà construite, nous nous concentrerons ici sur la gestion des utilisateurs et sur l'affichage des informations à l'écran.

Nous avons ici pour but d'implémenter des fonctionnalités qui permettront à l'utilisateur de notre site de :

- s'inscrire
- se connecter
- rechercher des films dans la base
- regarder les films qu'il a ajouté à sa liste
- noter des films
- commenter des films
- suivre d'autres utilisateurs
- regarder les listes de films et les commentaires d'autres utilisateurs

Architecture

Viennent maintenant les contraintes techniques. Nous avons, pour la partie web, utilisé exclusivement du CSS/HTML et du PHP. Nous avons choisi d'utiliser une architecture Modèle-Vue-Contrôleur qui est très utilisée dans le développement web. Cette structure nous permet de séparer le code HTML du PHP et des requêtes à la base de données, ce qui rend toute modification du code plus simple par la suite.

Cette architecture se présente comme qui suit :

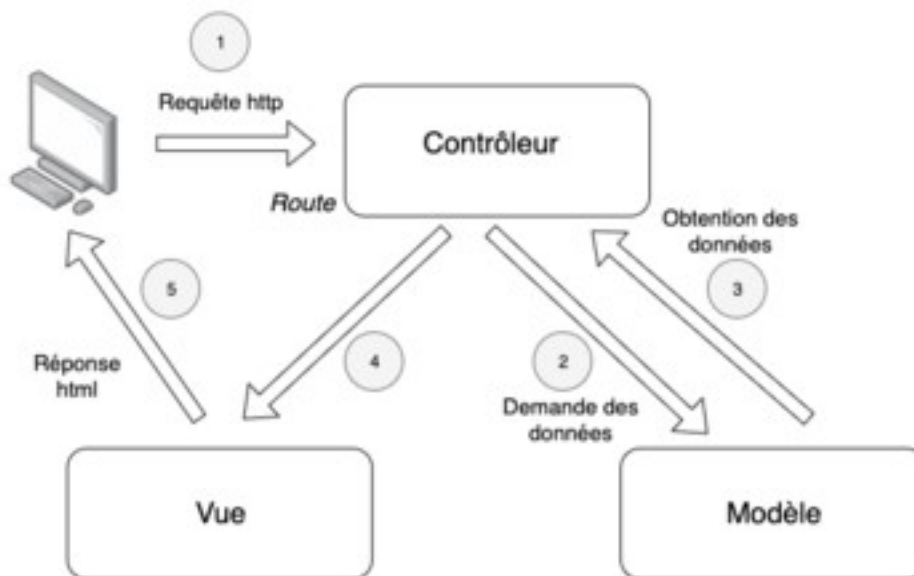


Figure 24: Schéma du modèle MVC

Le code est séparé en trois parties, le modèle, la vue et le contrôleur qui ont chacun un rôle spécifique.

Le modèle va gérer toutes les requêtes à la base de données, il va se composer de fonctions qui, sans aucune logique, vont retourner telle ou telle information.

La vue est le code HTML, chaque page du site aura sa vue et elle sera composée du code html de l'affichage de la page.

Le contrôleur fait le lien entre la requête brute à la base de donnée du modèle et à l'affichage de la page avec la vue. Sa fonction commence lorsque l'utilisateur effectue une requête (appui sur un bouton, un lien...). Le contrôleur va alors récupérer les informations du formulaire (s' il y en a un), appeler des fonctions du modèle pour récupérer des informations et afficher le tout dans la vue.

Prenons un exemple :

Je souhaite afficher dans ma page d'accueil la liste des films les plus récents.

Je crée donc dans mon modèle la fonction qui va effectuer la requête SQL à la base de données et renvoyer le résultat souhaité.

```
function getAllRecentFilms(){
    $db=$this->getDatabaseConnection();
    $sql="SELECT titre, annee_sortie, image FROM film ORDER BY annee_sortie DESC LIMIT 21";
    $result=mysqli_query($db, $sql);

    return $result;
}
```

Figure 25: Fonction permettant d'obtenir tous les films les plus récents

Dans mon contrôleur, j'appelle cette fonction pour récupérer la liste de films, et j'appelle la vue où je pourrais afficher ces films.

```
function displayAccueil(){
    $liste_films = $this->filmModel->getAllRecentFilms();
    require("Views/Accueil.php");
}
```

Figure 26: Fonction qui permet d'afficher l'accueil

Dans la vue, on utilisera une boucle pour afficher toutes les données. Le reste est composé simplement de l'affichage de la page avec du code HTML.

```

<div id="content">
  <h1>Vous nous avez manqué <?= $_SESSION["login"] ?></h1>
  </br>

  <h2>Voici les derniers films sortis :</h2>

  <div id="liste_films">
    <?php while($donnees = $liste_films->fetch_array()){ ?>

      <div id="un_film">
        <form id="form_film">
          <input type="hidden" name="film" value="<?php echo $donnees['titre'] ?>">
          <button id="bouton_film" type="submit" name="action" value="un_film">
            
          </button>
        </form>
      </div>
    <?php } ?>
  </div>

```

Figure 27: Vue de la page d'accueil

Et voilà, vous avez affiché la liste des films les plus récents de votre base de données (ici les images).

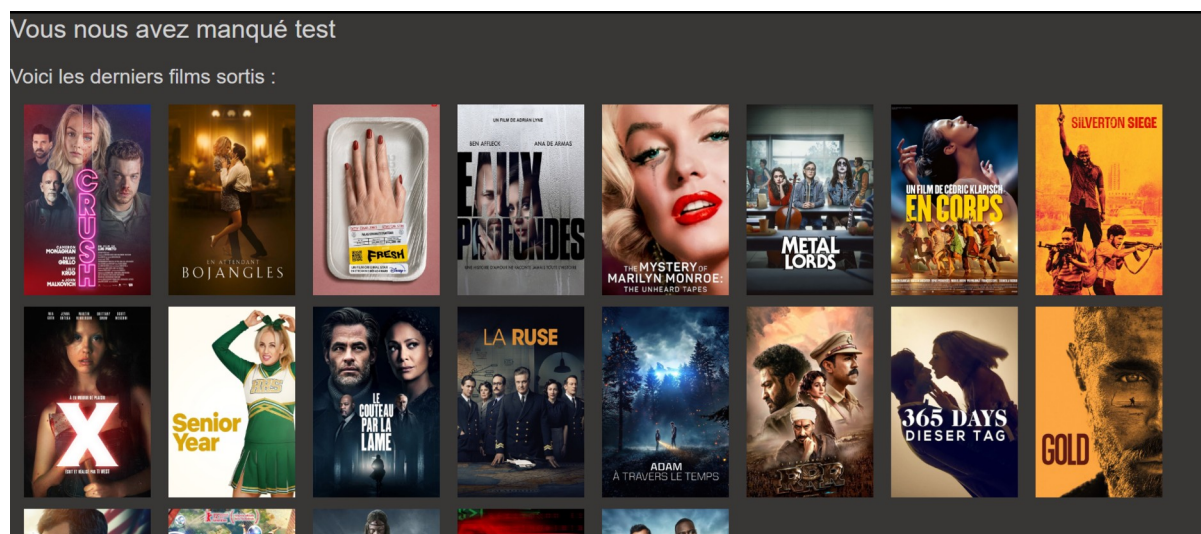


Figure 28: Page d'accueil

La couleur du fond, des titres et la disposition des images se fera avec du CSS.

Structure du site

Maintenant que nous avons vu l'architecture du code qui va générer le site et la manière de générer et d'afficher des requêtes SQL sur le site, nous allons voir sa structure et comment nous avons implémenté toutes les fonctionnalités décrites plus tôt.

La première page que vous allez voir en arrivant sur le site sera la page de connexion.

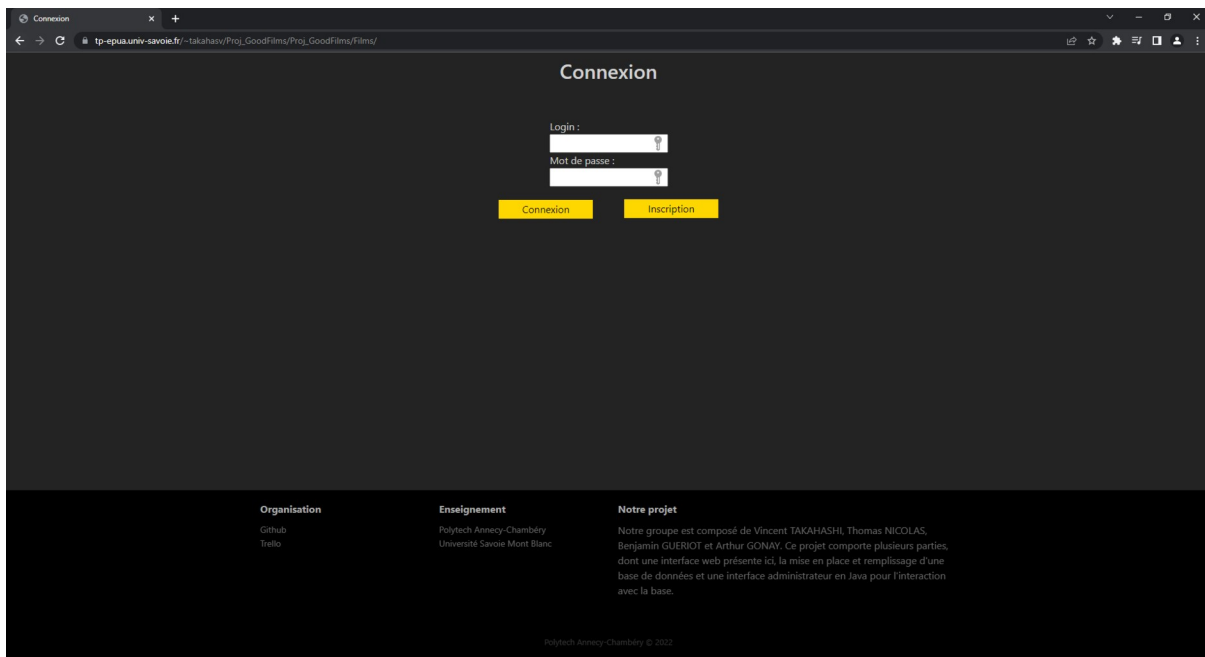


Figure 29: Page de connexion

Cette page permet à l'utilisateur de se connecter avec son login et son mot de passe présent dans la base. Comme vu précédemment, le mot de passe est crypté dans la base de données, il a donc fallu le décrypter avant de le comparer au mot de passe entré par l'utilisateur. Si le mot de passe est le même que celui présent dans la base, alors l'entrée est autorisée. La vérification du mot de passe s'effectue dans le contrôleur, celui-ci affichera la page d'accueil si le mot de passe et le login sont valides.

```
function displayConnexion(){
    if(isset($_POST["login"]) && isset($_POST["password"])){
        if($this->userModel->verifyUser($_POST["login"], $_POST["password"])){
            $_SESSION["connexion"]="ok";
            $_SESSION["login"]=$_POST["login"];
            $_SESSION["mdp"]=$_POST["password"];
            $val=$this->userModel->getUserId();
            $_SESSION["user_id"] = $val->fetch_assoc();
            $liste_films = $this->filmModel->getAllFilmsTitles();
            require("Views/Accueil.php");
        }
        else {
            $_SESSION["connexion"]="error";
            require("Views/Connexion.php");
        }
    }
    else {
        $_SESSION["connexion"]="error";
        require("Views/Connexion.php");
    }
}
}
```

Figure 30: Fonction permettant d'afficher la connexion

L'utilisateur a aussi la possibilité de s'inscrire afin d'ajouter son login et son mot de passe à la base de données, seulement si son login n'est pas déjà présent dans la base.

Figure 31: Page d'inscription

La page d'accueil vue précédemment est la première page sur laquelle atterrit l'utilisateur lorsqu'il se connecte sur la site.

En haut de la page d'accueil, nous pouvons observer un header qui va rester tout au long de la navigation sur le site et dicter son organisation.

- Le titre du site "Good2Watch" est un lien vers l'accueil
- Cliquer sur un film permet de voir ses informations
- Mon profil amène au profil de l'utilisateur
- La recherche par genre amène vers une page avec une liste des genres
- La recherche par titre permet de rechercher des titres de films
- La déconnexion permet de revenir à la page de connexion en effaçant les données de l'utilisateur



Figure 32: Header

Lorsque l'utilisateur clique sur l'image d'un film, une action est enregistrée. Dans l'index de notre site, si l'action correspond à la requête d'un utilisateur qui est de visionner un film en particulier, on appelle la fonction du contrôleur correspondante.

Pour afficher les informations d'un film, nous voulons le titre du film, l'année de sortie, le résumé, ce sont des informations présentes dans la table des films, donc récupérables facilement., mais aussi d'autres informations comme la note moyenne des utilisateurs ou leurs commentaires.

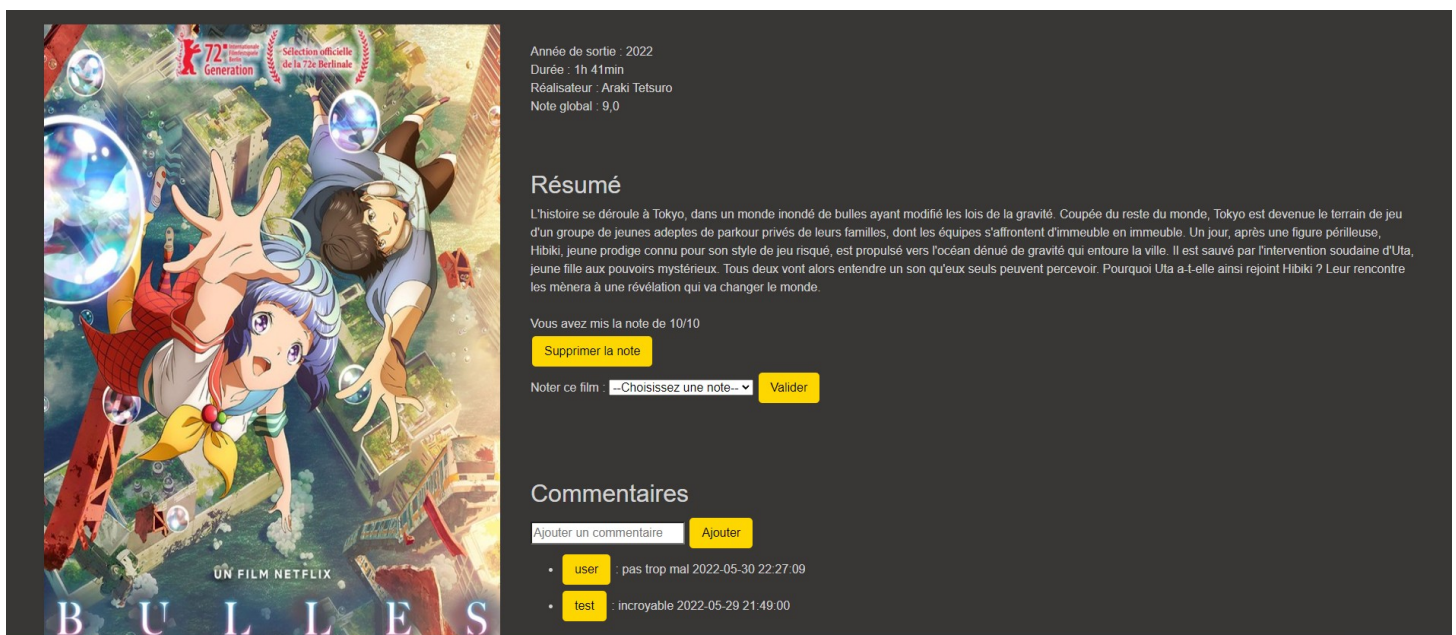


Figure 33: Page d'informations sur un film

Vous pouvez voir que l'utilisateur courant a mis la note de 10/10 au film *Bubble* mais que la note globale est de 9/10 (un autre utilisateur a mis la note de 8/10).

Les personnes suivies, la liste des films regardés, ainsi que les commentaires les plus récents apparaîtront dans la page de profil de l'utilisateur.

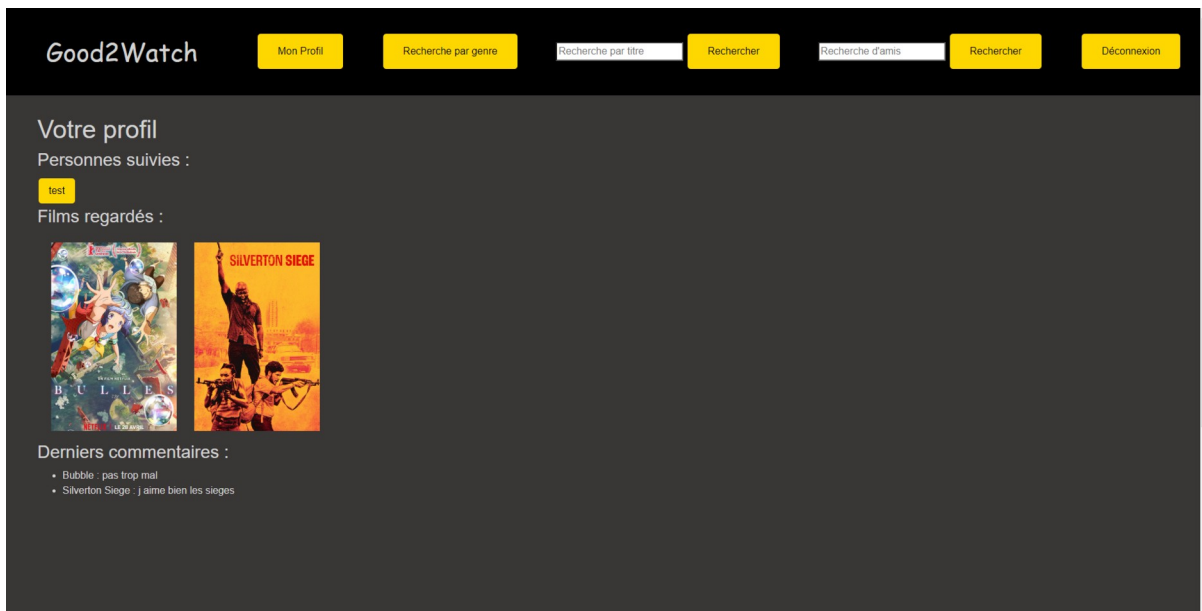


Figure 34: Page de profil de l'utilisateur connecté

Pour chaque affichage d'une page utilisateur, on vérifie si l'utilisateur que l'on veut afficher est le même que celui qui est connecté. Si non, on vérifie si l'utilisateur connecté suit l'utilisateur que l'on souhaite afficher.

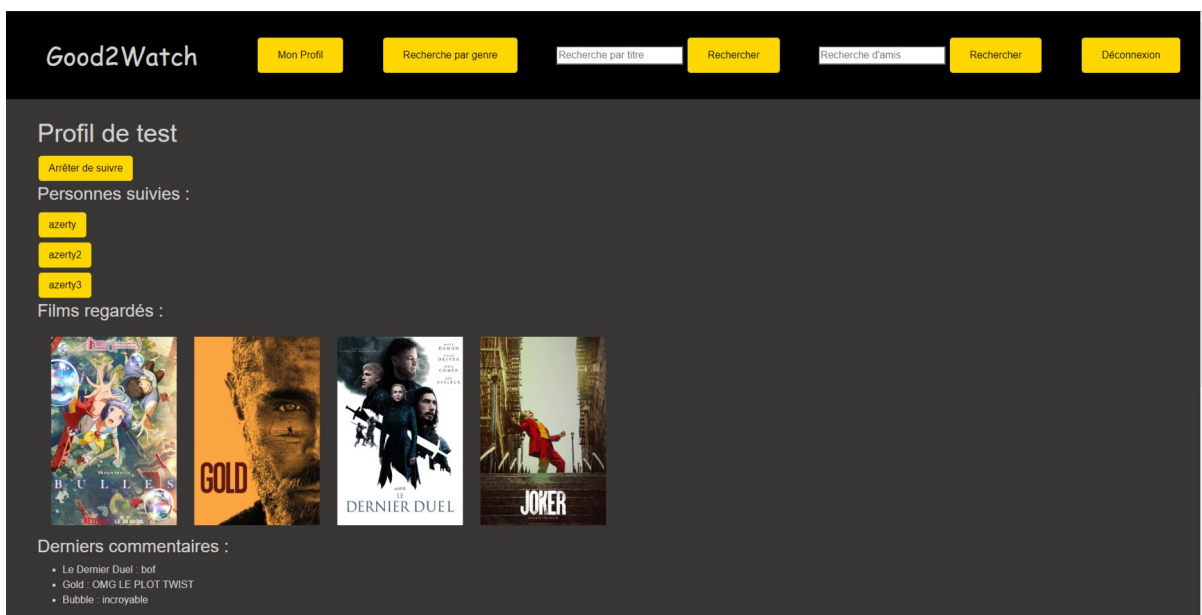


Figure 35: Page de profil d'un autre utilisateur

Ici, l'utilisateur qui est connecté (son login est "user") affiche le profil d'un autre utilisateur ("test"), qu'il suit. On peut voir que le bouton a comme option "arrêter de suivre". Nous pouvons aussi avoir accès à la liste des personnes suivies par test (en l'occurrence "azerty", "azerty2" et "azerty3"), la liste des films qu'il a regardé et ses commentaires les plus récents.

Durant toute la durée d'utilisation du site, on stocke son login dans une variable `$_SESSION["login"]`, qui va rester la même durant toute la durée de la connexion. Pour savoir si on est sur la page de l'utilisateur actuellement connecté ou pas, on teste l'égalité entre la variable de session et la variable du formulaire du bouton sur lequel l'utilisateur a cliqué.

```
if($login != $utilisateur_target){  
    $soi = false;  
    if($this->userModel->isAlreadyFriendWith($login, $utilisateur_target)){  
        $friend = true;  
    }  
    else{  
        $friend = false;  
    }  
}  
else {  
    $soi = true;  
}
```

Figure 36: Contrôle de l'utilisateur

Ici on obtient deux booléens `$soi` et `$friend` qui vont servir dans la vue pour savoir quoi afficher en fonction du rapport entre l'utilisateur connecté et l'utilisateur target (celui dont on cherche à voir le profil).

```

if ($soi){
    echo'<h1>Votre profil</h1>';
    $utilisateur_target = $login;
}

else{
    if(isset($_GET["suivre"])){
        $utilisateur_target = $_GET["suivre"];
    }
    else if(isset($_GET["fuir"])){
        $utilisateur_target = $_GET["fuir"];
    }

    echo'<h1>Profil de '.$utilisateur_target.'</h1>';

    if($friend){
        ?>
        <form id="form_fuir">
            <input type="hidden" name="fuir" value="<?= $utilisateur_target ?>">
            <button id="bouton_ami" type="submit" name="action" value="fuir">Arrêter de suivre</button>
        </form>

        <?php } else { ?>

        <form id="form_suivre">
            <input type="hidden" name="suivre" value="<?= $utilisateur_target ?>">
            <button id="bouton_ami" type="submit" name="action" value="suivre">Suivre</button>
        </form>

        <?php }
    }
}

```

Figure 37: Vue du profil

Nous n'avons plus qu'à tester dans la vue les deux booléens et afficher ce qui convient.

Pour les recherches d'amis et de films, on récupère l'input de l'utilisateur et on recherche dans la base de données les éléments qui ressemblent à cet input.

```

function getResearchFilm(){
    $db=$this->getDatabaseConnection();
    $sql="SELECT titre, image FROM film WHERE film.titre LIKE '%"$_GET['recherche']."%' ORDER BY titre DESC";
    $result=mysqli_query($db, $sql);

    return $result;
}

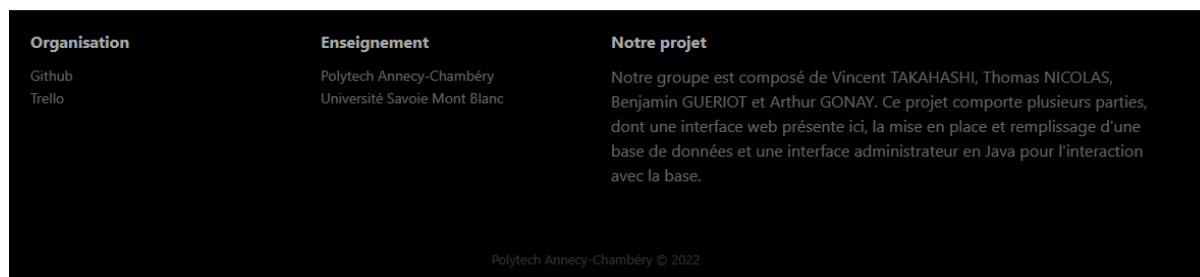
```

Figure 38: Fonction qui donne le film recherché

Du côté du style...

Pour le côté style, nous avons choisi un fond sombre avec des boutons jaunes qui font contraste et qui seront donc bien visibles.

Nous avons placé un footer avec différents liens intéressants vis-à-vis de notre projet.



Organisation	Enseignement	Notre projet
Github Trello	Polytech Annecy-Chambéry Université Savoie Mont Blanc	Notre groupe est composé de Vincent TAKAHASHI, Thomas NICOLAS, Benjamin GUERLOT et Arthur GONAY. Ce projet comporte plusieurs parties, dont une interface web présente ici, la mise en place et remplissage d'une base de données et une interface administrateur en Java pour l'interaction avec la base.
Polytech Annecy-Chambéry © 2022		

Figure 39: Footer

Pour aller plus loin..

Nous sommes assez satisfaits de notre projet même si nous aurions pu explorer d'autres pistes que nous n'avons pas pu traiter par manque de temps.

Nous pouvons penser à :

- Une gestion des acteurs
- Des pages dédiées pour les acteurs/réalisateurs
- Un classement des films en fonction des notes données par la communauté
- Héberger le site web sur un serveur (non local)
- Réaliser plus de statistiques sur les utilisateurs

Conclusion

Ce projet a représenté un véritable challenge pour nous tous car il est le plus ambitieux que nous avons mené jusqu'à présent. En effet, en l'espace d'un mois nous estimons avoir passé plus de 120h cumulées à la réalisation de ce projet. Nous sommes particulièrement fiers de notre gestion de projet qui a été une priorité dès les débuts du projet. En utilisant les outils qui nous ont été enseignés, les objectifs étaient clairs pour chaque membre de l'équipe ce qui a permis d'en arriver à son terme.

Sa multidisciplinarité a été une véritable aubaine pour approfondir les modules dispensés lors de ce semestre. Que ce soit pour la gestion et la récolte de données, la création d'une interface JAVA et sa connexion à une base de données, ou bien la création complète d'un site web ; nous ressortons de nombreux enseignements techniques de cette expérience et nous sommes heureux d'avoir choisi ce sujet.

Annexe

Table des acronymes

API : Application Programming Interface

JDBC : Java Database Connectivity

IDE : Environnement de Développement Intégré

BDD : Base de Données

SQL : Structured Query Language

HTML: HyperText Markup Language

PHP : HyperText PreProcessor

CSS : Cascading Style Sheets

CSV : Comma-Separated Values

GUI : Graphical User Interface

MVC : Modèle-Vue-Contrôleur

Table des outils

Trello: <https://trello.com/b/Pj7xMKxc/projgoodfilms>

Github: https://github.com/Brocowlee/Proj_GoodFilms