



# Rapport de projet de C/SD

par

**Nathan BARLOY, Valentin CRÔNE, Johan TOMBRE**



1 AVENUE PAUL MULLER  
54600 VILLERS-LÈS-NANCY  
FRANCE

# Table des matières

<b>1 État de l'art</b>	<b>3</b>
1.1 Introduction . . . . .	3
1.1.1 Qu'est ce qu'un système de recommandation ? . . . . .	3
1.1.2 La place des systèmes de recommandation dans la recherche d'information . . . . .	4
1.1.3 Classer les différents systèmes de recommandation . . . . .	5
1.2 Les données . . . . .	5
1.2.1 L'approche réactive . . . . .	5
1.2.2 L'approche proactive . . . . .	6
1.2.3 Conclusion sur les données . . . . .	6
1.3 Méthode de pondération TF-IDF . . . . .	7
1.4 La matrice d'usage . . . . .	7
1.5 Réduction de matrice . . . . .	7
1.5.1 Clustering . . . . .	8
1.6 Calcul de similarités . . . . .	9
1.6.1 La distance Euclidienne . . . . .	9
1.6.2 Distance cosinus . . . . .	9
1.6.3 Corrélation de Pearson . . . . .	10
<b>2 Rapport</b>	<b>11</b>
2.1 Introduction . . . . .	11
2.2 Les objectifs . . . . .	11
2.3 Amélioration de la base de données . . . . .	12
2.3.1 Liens des bandes annonces . . . . .	12
2.3.2 Récupération des affiches des films . . . . .	12
2.3.3 Création d'une base de données utilisateurs . . . . .	12
2.4 Création d'une distance entre les films . . . . .	13
2.4.1 La distance de Jacard . . . . .	13
2.4.2 La distance entre les années . . . . .	14
2.4.3 Amélioration de la distance entre les genres . . . . .	14
2.4.4 Recommandation de contenu . . . . .	15
2.5 Système de recommandation collaboratif . . . . .	15
2.5.1 Création d'une distance entre 2 utilisateurs . . . . .	15
2.5.2 Estimation d'une note . . . . .	16
2.5.3 Tri des films . . . . .	16
2.6 Structures de données . . . . .	16
2.6.1 Structures de données statiques . . . . .	17
2.6.2 Structures de données dynamiques . . . . .	18
2.7 Fonctionnement de l'application produite . . . . .	24
2.7.1 Fonctionnement du serveur . . . . .	24
2.7.2 Fonctionnement du client CLI . . . . .	24
2.7.3 Fonctionnement du client GUI . . . . .	24
2.8 Protocole de communication entre client et serveur . . . . .	26
2.8.1 Table des codes de retour . . . . .	26

<b>3</b>	<b>Gestion de Projet</b>	<b>27</b>
3.1	matrice SWOT . . . . .	27
3.2	Estimation du temps passé sur le projet . . . . .	27
3.3	Comptes Rendus de Réunion . . . . .	29
3.4	Post-mortem . . . . .	37

# Chapitre 1

## État de l'art

### 1.1 Introduction

#### 1.1.1 Qu'est ce qu'un système de recommandation ?

Un système de recommandation a pour but de fournir des recommandation de résultats pertinentes à un utilisateur, en fonction de différents paramètres comme ses préférences, son historique, le temps passé sur le contenu, etc.., afin de lui proposer directement un contenu ciblé sans efforts de sa part.

Un algorithme de recommandation peut de ce fait être utilisé pour faciliter les recherches de contenus sur un thème donné, ce qui est bénéfique pour l'utilisateur car il aura plus d'informations pertinentes à regarder, et permettre aux gérants de la plateforme de fidéliser la clientèle.

Prenons un exemple, un site de e-commerce sur lequel on trouve facile des articles en rapport avec l'article que l'on est en train de consulter rendra plus facile la recherche, et donc aidera le client à trouver rapidement le produit adéquat, ce qui réduira le temps nécessaire à l'achat, et évitera la perte du client. Un autre exemple est une plateforme de visionnage de vidéos en ligne, qui génère des bénéfices grâce à la publicité, un tel système proposera du contenu ciblé intéressant pour l'utilisateur, qui consultera un plus grand nombre de vidéos sur le site, ce qui permet à la fois de satisfaire l'utilisateur et améliorer la génération de revenus liés à la publicité. Ces effets sont liés au fait que l'utilisateur reçoit des suggestions pertinentes de la part du système de recommandation auxquelles il n'aurait pas forcément pu prêter attention autrement.

### **1.1.2 La place des systèmes de recommandation dans la recherche d'information**

La recherche d'information sur internet est fondée sur un principe d'indexation des données. Afin de pouvoir répondre aux requêtes des utilisateurs dans un temps raisonnable, on indexe les données dans une base de données, et on les filtre en fonction de la requête des utilisateurs. Les requêtes sont entrées à partir de mots clés (requêtes ad hoc), et le moteur de recherche va alors retourner les résultats qui correspondent. Le nombre conséquent de résultats obligera le moteur de recherche à limiter la quantité d'informations retournée en limitant un certain nombre de résultats par page. Ce système fonctionne plutôt bien, mais nécessite que l'utilisateur traite lui-même un grand nombre de résultats, ce qui peut se limiter à la ou les premières pages car il est impossible de tout traiter. On cherche donc à rendre les résultats de la première page plus pertinents. On fait alors immédiatement le lien avec les systèmes de recommandation, car au lieu de filtrer par mots clés, on va pouvoir rechercher des contenus à thème proche, date proche, et personnaliser les résultats en fonction des habitudes de l'utilisateur.

Ces exemples montrent un certain nombre de cas où l'utilisation d'un système de recommandation s'avère utile. Cependant il existe de nombreux autres, car leur champ d'application est immense, c'est pourquoi il est intéressant de s'intéresser aux systèmes existants pour tenter de les reproduire et les améliorer.

### 1.1.3 Classer les différents systèmes de recommandation

Il existe différents types de systèmes de recommandation, en fonction du type de données à classer, de la puissance disponible, de activités utilisateurs, des besoins de l'entreprise, et de nombreux autres facteurs. Les facteurs principaux à retenir sont :

- La connaissance des préférences utilisateur.
- La similarité entre les utilisateurs, les uns par rapport aux autres. (Métrique)
- La connaissance d'informations sur des données à recommander
- La connaissance d'informations de regroupement sur des données à recommander

A partir de ces facteurs on produit différents types de recommandations. Les systèmes les plus utilisés sont le filtrage basé sur le contenu, et le filtrage collaboratif.

#### Le filtrage basé sur le contenu

Le filtrage basé sur le contenu va essayer d'associer un utilisateur à un ensemble de données proche de ses préférences. On va alors rechercher des données qui pourraient l'intéresser en tenant compte uniquement de lui-même, et de ce qu'il a consulté. Pour cela, il faut dresser des liens entre les différents éléments de la base de donnée, pour pouvoir déterminer si un élément est proche ou non de ce qu'a consulté l'utilisateur. L'avantage de ce filtrage est qu'il ne nécessite pas d'avoir d'autres utilisateurs pour fonctionner, puisqu'on ne compare pas les utilisateurs entre eux. Cela est donc très pratique quand la base de donnée des utilisateurs est petite.

#### Le filtrage collaboratif

Le filtrage collaboratif va mesurer une certaine distance entre un utilisateur donné, et tous les autres utilisateurs pour sélectionner les utilisateurs les plus proches uniquement. On utilisera alors le profil de ces utilisateurs proches pour regarder les données qui les ont intéressés pour les recommander à notre utilisateur initial. L'avantage de ce filtrage est qu'il ne nécessite pas de créer un algorithme qui détermine si deux données sont proches ou non. Cependant, si la base de donnée des utilisateurs n'est pas assez fournie, ce filtrage ne sera que très peu efficace.

Évidemment, les filtrages les plus efficaces sont ceux qui mélangeant les méthodes décrites précédemment, de manière à en tirer les avantages.

## 1.2 Les données

Pour pouvoir lier des produits avec les utilisateurs, il est indispensable de mettre au point un système de notation afin de déterminer si un utilisateur apprécie ou non un produit. On distingue deux approches différentes :

- l'approche « réactive » : où le système de recommandation demande à l'utilisateur des informations / avis sur certains produits afin d'affiner ses recommandations. Cette approche présente cependant l'inconvénient de solliciter l'utilisateur.
- l'approche « proactive », qui anticipe les goûts de l'utilisateur.

### 1.2.1 L'approche réactive

Cette approche consiste à déterminer les goûts de l'utilisateur à travers un processus conversationnel. Le système demande régulièrement des informations à l'utilisateur sur ses préférences, ses goûts... puis va proposer des recommandations que l'utilisateur va accepter ou critiquer les résultats afin d'affiner le système. L'avantage de ce système de critique est qu'il est simple à appliquer et ne requiert pas de connaissance de l'utilisateur dans ce domaine. Toutefois, l'inconvénient majeur est qu'il demande directement à l'utilisateur de faire un effort pour exprimer son avis et donner un retour.

### 1.2.2 L'approche proactive

L'approche proactive ce base davantage sur la déduction des appréciations pour fournir ensuite une recommandation. Ainsi l'utilisateur n'a plus à guider le système avec des retours sur les recommandations proposées mais va observer les interactions de l'utilisateur pour déterminer les goûts de celui-ci. Ces observations peuvent être directes ou indirectes.

#### Observation directe

Ce sont toutes les informations que l'utilisateur donne de manière explicite en notant, postant un commentaire sur un produit par exemple ou encore les informations du profil de l'utilisateur comme son âge, son sexe, sa localisation. La notation des produits est un élément centrale dans un système de recommandations. Ces notes sont principalement numériques et peuvent avoir différentes formes qui délivrent plus ou moins d'informations.

Il y tout d'abord la notation unaire : cette notation ne délivre qu'une seule information, si l'utilisateur a aimé un produit. On peut donner par l'exemple de réseau social Instagram qui donne la possibilité aux utilisateurs de « liker » une photo uniquement, il est impossible de mettre un avis négatif.

Il y a ensuite la notation binaire qui prend en compte l'avis négatif de l'utilisateur. L'information transmise est donc binaire : « J'aime » / « Je n'aime pas ». C'est par exemple ce qu'utilise Facebook ou Youtube. L'avantage de cette notation est facilite le choix de l'utilisateur.

Enfin, la notation ordonnée est celle qu'on retrouve dans la majorité des site de e-commerce (Amazon, C Discount et autres). L'utilisateur donne une note à un produit suivant une échelle de notation comprise entre deux valeurs (entre 1 et 5 par exemple). Plus ce chiffre est élevé, plus l'utilisateur a aimé le produit. Cette notation permet plus de nuances sur l'appréciation des produits et améliore donc les recommandations qui en découlent. Cependant, le coût en calcul en est impacté car l'information n'est plus binaire. De plus cette notation est sensible à la manière de noter de chaque utilisateur : une note de 3/5 sera considéré comme une mauvaise note par certains utilisateurs alors que pour d'autres, ce sera un note neutre / moyenne. Cela peut donc impacter la qualité des recommandations.

Une autre ressource importante dans les systèmes de recommandations est l'analyse des commentaires, descriptions, noms des produits... afin de produire des tags spécifiques à l'utilisateur. Diverses méthodes existent pour cela et nous aborderons plus en détail la méthode TF-IDF plus loin dans le rapport. Grâce aux tags produits par ce système, il est possible de faire des rapprochements entre utilisateurs et/ou produits.

#### Observation indirecte

Ce sont l'ensemble des informations implicites données par l'utilisateur ou déduites par le système. Il peut s'agir par exemple d'actions réalisées sur une page web comme une recherche, un ajout dans un panier d'achat ou tout simplement de la fréquence de consultation d'une page. Il est ensuite possible, à partir de ces informations, d'estimer les préférences de l'utilisateur. Par exemple, si on utilisateur ajoute un produit dans son panier ou tout simplement le consulte fréquemment, on peut conclure que ce type de produit l'intéresse et intégrer cette information à notre système.

### 1.2.3 Conclusion sur les données

Nous avons vu différentes manières de collecter des informations relatives à l'utilisateur dans le but d'alimenter notre algorithme de recommandation. Que ce soit par une approche réactive ou proactive, ces techniques peuvent soulever un débat sur la collecte de données. Il faut donc mener une réflexion pour trouver un bon compromis entre efficacité du système et collecte de données « abusive ».

### 1.3 Méthode de pondération TF-IDF

La TF-IDF ,correspondant à "Term Frequency - Inverse Document Frequency", est une méthode de mesure très utilisée pour l'analyse du contenu d'un texte. Il permet d'évaluer l'importance d'un mot dans un texte qu'on pourra ensuite considérer comme tag.

Afin de mieux comprendre son fonctionnement, définissons quelques éléments :

- $x$  est un mot
- $N$  est le nombre total de documents
- $y$  est un de ces documents
- $tf_{x,y}$  est la fréquence d'apparition de  $x$  dans  $y$
- $df_x$  est le nombre de documents contenant  $x$
- $w_{x,y}$  est le poids de  $x$  dans le document  $y$  (le score obtenu)

On peut alors établir la formule suivante :

$$w_{x,y} = tf_{x,y} \cdot \log\left(\frac{N}{df_x}\right)$$

L'analyse de cette formule permet de bien comprendre son fonctionnement. On comprend facilement l'intérêt de  $tf_{x,y}$  : plus il y a d'occurrences d'un mot dans un texte, plus il est important. Cependant, si on se contente de ce rapport, les mots fréquemment utilisés tels que les déterminants auront un poids fort. Pour pallier à ce problème, on introduit  $\log(\frac{N}{df_x})$ .

En effet, si le mot  $x$  n'apparaît que dans un document, alors  $tf_{x,y}$  sera multiplié par  $\log(N)$  et gonflera ainsi  $w_{x,y}$ . À l'inverse, si  $x$  apparaît dans les  $N$  documents, alors  $tf_{x,y}$  sera multiplié par  $\log(1)$  et on obtiendra alors un poids de 0.

La pertinence d'un mot dans un texte se mesure donc grâce à la TF-IDF qui fait le lien entre la rareté de ce mot dans un ensemble de documents et sa fréquence d'apparition dans ce texte.

### 1.4 La matrice d'usage

Également appelé le modèle utilisateur, la matrice d'usage contient des données collectées sur les utilisateurs associés aux produits. On le représente généralement sous la forme d'une matrice avec les utilisateurs sur les lignes et les produits en colonne. Les scores attribués peuvent être une note, un nombre de consultation, un durée, un like... Cette matrice comporte généralement de nombreuses valeurs manquantes qu'il va falloir chercher à déterminer grâce à notre système de recommandation. Il est également intéressant de remarquer que les goûts des utilisateurs évoluent au cours du temps. Il est donc important de réajuster les données de cette matrice régulièrement.

### 1.5 Réduction de matrice

Reprendons la matrice d'usage tout juste présentée. Elle est de dimension  $m.n$  avec  $m$  le nombre d'utilisateurs et  $n$  le nombre de produits. Dans la pratique, cela représente habituellement des milliers / millions d'utilisateurs pour autant voire plus de produits. On obtient donc une matrice possédant des milliards de valeurs. Il serait donc très long d'effectuer des calculs dessus pour en déduire des recommandations. La solution est donc de réduire la dimension de la matrice.

Plusieurs approches sont possibles. On peut, par exemple, ne pas prendre en compte les utilisateurs dont on ne possède que très peu d'informations ou de la même manière retirer les produits peu populaires et donc ayant moins de chance d'intéresser les utilisateurs. Cependant, cette méthode a comme gros inconvénient de réduire le champ des recommandations proposées. En effet, comment proposer à un utilisateur un produit qui lui conviendrait parfaitement s'il est retiré de la liste car peu connu ? Cela restreint plus ou moins les produits recommandables à ceux qui sont populaires et donc déjà connus par la plupart.

Une autre approche consiste à ne conserver qu'une gamme de produits répondant à un critère spécifié. Par exemple, il est possible de supprimer les films de genre romantique si l'utilisateur ne manifeste aucun intérêt pour ce genre. Enfin, une dernière méthode consiste à ne prendre en compte que les utilisateurs jugés comme étant proche de l'utilisateur cible, c'est ce qu'on appelle le clustering (détailé dans la section suivante).

L'avantage de cette méthode est donc de réduire le temps de calcul pour déterminer une recommandation. Cependant, la qualité de la recommandation s'en retrouve impactée. En effet cela réduit le nombre de personnes analysées par l'algorithme dans le cas d'une réduction du nombre d'utilisateur, et une réduction du nombre de produits réduira le champs des recommandations à un certains type.

### 1.5.1 Clustering

L'idée est de décarter les items (resp. les utilisateurs) jugés non représentatifs pour un utilisateur (resp. item) donné. On réduit ainsi la dimension de la matrice tout en minimisant la perte de données avant d'effectuer les calculs de recommandation. Pour ce faire, on utilise des méthodes de partitionnement qui consistent à regrouper les éléments similaires. La méthode la plus connue est k-means.

Un cluster est un regroupement d'éléments qui sont similaires entre eux et dissimilaires aux autres éléments appartenant aux autres clusters. Le but de l'algorithme k-means est donc de créer  $k$  clusters à partir d'un ensemble d'utilisateurs.

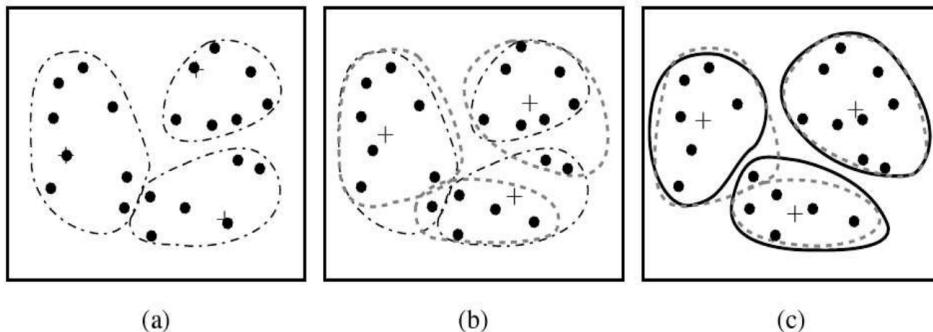


FIGURE 1.1 – Illustration de l'algorithme k-means pour  $k=3$

Le fonctionnement de cet algorithme est le suivant :

---

**Algorithm 1** Algorithme de partitionnement k-means

---

**INPUT** : k : nombre de clusters, U : l'ensemble des utilisateurs

**OUTPUT** : k clusters

Choisir aléatoirement k centroïdes initiaux de clusters

**repeat**

    Réaffecter chaque utilisateur de U au cluster dont il est le plus similaire

    Recalculer les distances des utilisateurs dans chaque cluster

    Mettre à jour les centroïdes

**until** Stabilité des centroïdes

---

Cet algorithme présente comme avantage la facilité de mise en oeuvre ainsi que son efficience. De plus, si on prend un groupe de n utilisateurs et que l'on souhaite réaliser k clusters en t itérations, la complexité de l'algorithme est en  $\Theta(knt)$ .

En revanche, l'algorithme est très sensible aux données aberrantes. En effet, un utilisateur très éloigné de tous les autres sera tout de même intégré à un cluster et affectera son centroïde.

Il existe un autre algorithme qui est moins sensible à ce type de données : le "PAM" pour Partitioning Around Medoids ou encore k-medoid. Sans rentrer dans les détails, cet algorithme ressemble au k-means à la différence qu'il prend un utilisateur comme centroïde du cluster (qui est alors appelé médoïde). Néanmoins, cet algorithme a une complexité plus élevée, en  $\Theta(tk(n - k)^2)$ . C'est pourquoi il est moins utilisé que k-means.

## 1.6 Calcul de similarités

Un calcul de similarité est utile pour déterminer la distance entre un utilisateur et un produit mais également entre deux utilisateurs (ou produits). Plusieurs outils existent pour nous permettre d'effectuer ce calcul. Nous verrons dans cette partie quelques unes de ces méthodes et discuterons des avantages et inconvénients qu'ils présentent.

### 1.6.1 La distance Euclidienne

La distance euclidienne est obtenue par la formule suivante :

$$L(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2} \quad (1.1)$$

### 1.6.2 Distance cosinus

On considère les vecteurs ligne de x et y. Le but est de calculer la distance, l'angle entre ces deux vecteurs. Dans notre cas, les vecteurs x et y seront des vecteurs associés aux profils de deux utilisateurs par exemple.

$$CosSim(\vec{x}, \vec{y}) = \frac{\sum_{i=1}^m x_i y_i}{\sqrt{\sum_{i=1}^m x_i^2} \sqrt{\sum_{i=1}^m y_i^2}} \quad (1.2)$$

L'inconvénient majeur de cette méthode est qu'elle considère tout film non noté comme un zéro, qui est une mauvaise note, ce qui va réduire la distance entre deux utilisateurs qui en réalité seraient jugés différents.

### 1.6.3 Corrélation de Pearson

Afin de pallier le problème cité précédemment, il est possible de normaliser au préalable les notes en leur soustrayant la moyenne des notes données par l'utilisateur. Cette méthode est également appelé la distance cosinus centrée. On a alors :

$$Corr(x, y) = \frac{\sum_{i=1}^m (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^m (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^m (y_i - \bar{y})^2}} \quad (1.3)$$

Le résultat obtenu se situe alors dans l'intervalle  $\llbracket -1; 1 \rrbracket$  avec -1 signifiant que les deux utilisateurs ont des goûts complètement opposés, 0 que les utilisateurs non rien en commun et enfin 1 que les deux utilisateurs sont totalement similaires.

# Chapitre 2

# Rapport

## 2.1 Introduction

Le but de ce projet est de concevoir un système de recommandation comme le propose Netflix, Google (Youtube), Amazon et beaucoup d'autres pour la recommandation de leurs produits.

Au cours de ce rapport, nous aborderons les choix que nous avons pris tout au long de ce projet, la présentation du code réalisé, le choix des structures ainsi que l'analyse des résultats obtenus.

## 2.2 Les objectifs

Nous avons vu dans l'Etat de l'art les principales fonctions présentent dans un système de recommandation. Toutefois, nous ne les implémenterons pas toutes en raison du temps disponible à la réalisation du projet.

Pour commencer, nous avons jugé qu'il était intéressant d'aborder les deux types de recommandations (par le contenu et collaborative) car il s'agit des deux approches différentes qui présentent des avantages et inconvénients différents.

La méthode TF-IDF aurait pû être intéressante dans la recommandation basée sur le contenu, mais nous avons estimé que la taille des synopsis ne permettrait pas de fournir des résultats suffisamment pertinents.

De la même manière, nous avons écarté l'idée d'implémenter une méthode de clustering afin de réduire les matrices en raison du nombre d'items et d'utilisateurs présents dans notre base de données. En effet, avec 100 films et une centaine d'utilisateurs, l'exécution du programme ne sera pas ralenti par la taille de la base de données. À l'inverse, réduire le nombre de données traités entraînera une perte d'information réduisant la qualité de la recommandation.

Pour ce qui est des interfaces, nous souhaitons développer une interface console ainsi qu'une interface graphique qui disposerons des mêmes fonctionnalités :

- Identification avec un compte
- Consultation de tous les films
- Notation d'un film
- Visualisation des recommandations par les 2 méthodes
- Visualisation des tendances (films classés par les moyennes)
- Exportation des résultats

## 2.3 Amélioration de la base de données

Dans le but d'améliorer l'expérience de l'utilisateur sur l'interface graphique, nous avons décidé de compléter la base de données mise à disposition afin d'y inclure les liens vers les bandes-annonces des films ainsi que leurs affiches respectives. De plus, le fonctionnement de l'algorithme de recommandation collaboratif requière une base de données d'utilisateurs ayant notés plusieurs films.

Nous avons décidé de réaliser ces opérations à partir de scripts python qui sont plus simples et rapides à mettre en oeuvre, grâce notamment à ses nombreuses librairies disponibles en ligne. Il est important de remarquer que ces scripts ont été exécutés en amont afin de récupérer les éléments souhaités mais ne font en rien partie du code du programme.

### 2.3.1 Liens des bandes annonces

Comme notre interface graphique sera gérée grâce à la librairie WebKitWebView, il sera possible d'intégrer à notre page un lecteur youtube avec la bande-annonce. Il suffit donc de récupérer les liens URL des vidéos pour que l'ensemble fonctionne. Ces résultats seront alors stockés dans un fichier texte avant d'inclure les liens au fichier JSON.

Les grandes étapes dans la réalisation de ce script ont été :

- Générer la requête à partir du lien youtube ([www.youtube.com/results?search\\_query=](http://www.youtube.com/results?search_query=))
- Récupérer l'intégralité du code HTML et générer l'exécution de cette requête
- Scanner le contenu HTML afin de ne conserver que le lien du premier résultat (on suppose que l'algorithme de recommandation youtube est efficace)
- Isoler la partie utile du lien récupéré (il s'agit des 11 derniers caractères après [https://www.youtube.com/watch?v=](http://www.youtube.com/watch?v=))
- Stocker ce résultat dans la base de données

### 2.3.2 Récupération des affiches des films

Avoir une affiche de chaque film de la base de données permet une fois de plus d'améliorer grandement l'interface graphique. Après plusieurs recherches, nous avons décidé de récupérer les images du site [TheMovieDB].

Pour ce faire nous avons, de la même manière que pour les liens youtube, utilisé les url pour effectuer nos recherches. Puis nous avons scanné le contenu de la page html résultante pour détecter le résultat correspondant à notre recherche (correspondance des dates notamment). Une fois trouvé, nous naviguons vers la page du résultat et scannons à nouveau la page à la recherche de l'image souhaitée avant de la télécharger dans un dossier spécifique et d'inclure le chemin d'accès dans notre base de données.

### 2.3.3 Crédit d'une base de données utilisateurs

Indispensable au fonctionnement de la méthode de recommandation collaborative, une base de données d'utilisateurs ayant attribués des notes aux films était nécessaire. Plusieurs pistes ont été explorées :

- Création fictive d'utilisateurs et de notes (aléatoires)
- Création d'un formulaire à partager avec des amis pour constituer notre propre base de données
- Trouver ces informations en ligne

Nous avons au final retenu cette dernière solution après avoir trouvé une base de données de [MovieLens] qui regroupe près de 100000 notes d'utilisateurs sur 10000 films. Le seul inconvénient est que cette base de données ne prend pas en compte les séries. Par conséquent, les résultats exposés ne seront pas les plus justes mais permettront d'avoir un aperçu de cet algorithme.

La base de données de [MovieLens] est au format csv, il était donc très simple de récupérer les données souhaitées. Le seul problème était les titres des films qui étaient en anglais et ne correspondaient pas toujours à nos titres. Il a donc fallu prendre quelques minutes pour manuellement faire correspondre leurs titres avec nos id de films. Ensuite il n'est plus question que d'extraire les bons éléments du fichier csv et le tour est joué !

## 2.4 Création d'une distance entre les films

Chaque film est caractérisé par la donnée de différents attributs. Dans le cas présent, les attributs que nous considérerons sont :

- L'année de parution
- Les acteurs
- Le réalisateur
- Le genre
- Le type (film ou série)

Pour chacun de ces attributs, on va calculer une distance, dont la valeur sera comprise entre 0 et 1 (0 pour identique, et 1 pour complètement différent). On fait ensuite simplement une moyenne pondérée des distances obtenues. Les poids de cette moyenne sont modulables, ce qui nous permet de les modifier comme on le souhaite, pour avoir des proportions qui correspondent à ce que nous voulons.

Cette distance doit vérifier certaines propriétés de la distance mathématique :

- $d(a, b) = d(b, a)$  (la symétrie)
- $d(a, b) \geq 0$  (la positivité)
- $d(a, a) = 0$

Si la distance créée ne vérifie pas ces propriétés, alors elle n'est pas bonne.

### 2.4.1 La distance de Jacard

En mathématique, la distance entre 2 ensembles est souvent calculée comme étant le minimum des distances entre les points de ces ensembles :

$$d(E, F) = \min_{\substack{x \in E \\ y \in F}} d(x, y)$$

Cependant, dans notre cas, cette distance n'est pas très utile, puisqu'elle ne prend pas en compte les autres éléments des ensembles. Ainsi, une telle distance ne fera pas de différence entre 2 films où tous les acteurs sont les mêmes, et 2 films qui ont un seul acteur en commun. Il faut donc trouver autre chose : la distance de Jacard.

Cette distance permet de mesurer la similarité entre 2 ensembles finis : soient  $X$  et  $Y$  deux ensembles finis. Alors,

$$\text{dist\_Jacard} = 1 - \frac{|X \cap Y|}{|X \cup Y|}$$

On a bien que si  $X = Y$ , alors la distance de Jacard est nulle, et que si les deux ensembles sont complètement distincts, alors la distance vaut 1.

Cependant, on peut se dispenser de certains calculs pour trouver la distance de Jacard. En effet, on peut utiliser le fait que  $|X \cup Y| = |X| + |Y| - |X \cap Y|$ . Il n'y a alors plus qu'à calculer  $|X \cap Y|$  pour pouvoir ensuite calculer la distance de Jacard.

Cette distance de Jacard pourra être utilisée pour calculer la distance entre les attributs des films qui sont des ensembles. Dans notre cas, cela représente les acteurs, les réalisateurs et les genres.

Après quelques tests, on se rend compte que même si l'algorithme marche la plupart du temps, il y a certains cas où il plante à l'exécution et indique une division par zéro. Après quelques recherches, on comprend que cela est du au fait que parfois, la liste des acteurs, des réalisateurs ou des genres est vide. Ainsi, la distance de Jacard ne peut pas être calculée, puisqu'on divise par  $|X \cup Y|$ , qui vaut zéro si  $X$  et  $Y$  sont vides.

Ce bug nous permet aussi de nous rendre compte de l'absurdité de certains calculs : en effet, comment interpréter la distance entre un ensemble vide est un autre ensemble qui ne l'est pas forcément ? (par exemple, si on sait uniquement que le réalisateur du film 1 est Spielberg, et rien pour le film 2) Dans ce cas, donner une distance n'a aucun sens. Il faut donc aussi prendre en compte cela avant de faire les calculs.

Après quelques tests, on se rend compte que l'on obtient parfois une distance négative, ce qui est impossible pour une distance normalement : il y a donc un problème dans le code. Il se trouve en fait qu'il y avait un problème au niveau de certains types, ce qui faussait la fonction equals, qui sert à

comparer 2 éléments entre eux. Après correction, on obtient quelque chose de cohérent. Notamment, la distance entre un film et lui-même est toujours 0.

Cette distance de Jacard permet de donner une distance entre 2 ensembles : on peut donc l'utiliser pour tous les attributs qui sont des listes, i.e. les acteurs, les réalisateurs, et les genres.

### 2.4.2 La distance entre les années

Pour créer une distance entre 2 années, on va bien évidemment considérer la distance entre ces 2 entiers : cela signifie prendre la valeur absolue de la différence ( $|a_1 - a_2|$ ). Notre distance sera donc une fonction  $f$  qui sera à valeur dans  $\mathbb{R}_+$ . On va chercher des conditions sur cette fonction.

Tout d'abord, on veut obtenir une distance entre 0 et 1, donc doit aller de  $\mathbb{R}_+$  dans  $[0; 1]$ . De plus, on veut que l'ordre soit gardé : si  $a > b$ , on veut que  $f(a) > f(b)$ . Cela signifie que la fonction est croissante. Enfin, on doit avoir que  $f(0) = 0$  et  $\lim_{x \rightarrow +\infty} f(x) = 1$ .

En résumé, on cherche une fonction croissante de  $\mathbb{R}_+$  dans  $[0; 1]$  qui atteint ses bornes.

une première fonction à laquelle on peut penser est une fonction que l'on rencontre souvent en physique :

$$f(x) = 1 - e^{-\frac{x}{\tau}}, \tau \in \mathbb{R}_+^*$$

On peut alors faire varier le paramètre  $\tau$  pour moduler la fonction comme on veut. La valeur que nous avons choisi est  $\tau = 10$ , car on a alors que pour avoir une distance de 0.5, il que les deux dates soient séparées de 7 ans, ce qui correspond à peu près à ce qu'on veut.

Une autre fonction utilisable possible est la fonction arctangente. On a alors

$$f(x) = \arctan(\alpha \cdot x) \times \frac{2}{\pi}, \alpha \in \mathbb{R}_+^*$$

Comme précédemment, on cherche une valeur du paramètre  $\alpha$  qui corresponde à nos attentes. Dans ce cas, on prend  $\alpha = 0.15$ , ce qui donne une distance de 0.5 pour 7 années d'écart.

Dans tous les cas, c'est une bonne chose d'avoir des paramètres modifiables dans ces fonctions pour pouvoir mieux approcher ce que l'on veut.

### 2.4.3 Amélioration de la distance entre les genres

Pour la distance entre les genres, on utilisait jusqu'à présent la distance de Jacard. C'est une méthode qui marche, mais elle omet certains éléments dans son calcul, ce qui le rend moins précis. En effet, on ne prend pas en compte les potentielles ressemblances entre genres, on regarde simplement si un même genre est présent dans les deux films. Ainsi, si l'on considère 3 films dont les genres sont respectivement crime, mystère et romance, la distance de Jacard entre ces 3 films sera nulle, et on ne pourra pas les classer. Cependant, on se rend bien compte que les genres crime et mystère sont proches, alors qu'ils ne le sont pas de romance. Il faut donc essayer de prendre en compte les distances intrinsèques entre deux genres.

Pour cela, la meilleure méthode aurait été d'utiliser le machine learning. Avec une telle méthode, l'humain n'a pas besoin d'intervenir, il suffit d'analyser la base de donnée des films et de repérer ceux qui sont proches d'après les utilisateurs pour estimer des distances entre les genres. L'avantage est que l'on peut alors rajouter un nouveau genre sans problème, et sans expliciter son lien avec les autres genres : l'algorithme fait cela tout seul. Cependant, pour qu'une telle méthode marche, il faut une base de données assez conséquente pour avoir des résultats corrects. De plus, c'est une méthode difficile à mettre en place, et qui requiert des connaissances que nous n'avons pas.

Il faut donc utiliser une autre méthode, plus fastidieuse et moins précise. On va pour cela créer nous-même une matrice contenant les distances entre les films, puis on fera une moyenne des distances entre les genres du premier film avec ceux du deuxième film. Pour cela, il faut d'abord lister les films existant, qu'on obtient grâce à un simple algorithme qui parcours tous les film. On obtient la liste suivante : Action, Thriller, Drama, Adventure, Family, Romance, Fantasy, Sci-Fi, Animation, Comedy, Horror, Crime, Mystery, History, Musical, Sport, Biography, War et Music ; soit un total de

19 genres. On crée ensuite une matrice de distances qu'on utilisera pour nos calculs :

0.0	0.3	0.5	0.1	0.6	0.9	0.2	0.2	0.7	0.7	0.4	0.6	0.6	0.7	0.9	0.5	0.8	0.2	0.9
0.3	0.0	0.3	0.4	0.7	0.9	0.5	0.4	0.8	0.8	0.1	0.2	0.3	0.6	0.9	0.9	0.8	0.5	0.9
0.5	0.3	0.0	0.6	0.7	0.4	0.5	0.6	0.8	0.8	0.3	0.3	0.3	0.2	0.4	0.6	0.5	0.6	0.8
0.1	0.4	0.6	0.0	0.2	0.4	0.1	0.3	0.4	0.3	0.7	0.8	0.4	0.7	0.8	0.5	0.8	0.3	0.9
0.6	0.7	0.7	0.2	0.0	0.7	0.5	0.6	0.2	0.1	0.8	0.7	0.5	0.8	0.3	0.4	0.9	0.7	0.6
0.9	0.9	0.4	0.4	0.7	0.0	0.7	0.8	0.7	0.5	0.9	0.8	0.6	0.4	0.1	0.6	0.6	0.5	0.3
0.2	0.5	0.5	0.1	0.5	0.7	0.0	0.3	0.5	0.4	0.5	0.7	0.4	0.8	0.7	0.9	0.9	0.2	0.8
0.2	0.4	0.6	0.3	0.6	0.8	0.3	0.0	0.7	0.4	0.5	0.4	0.2	0.8	0.7	0.9	0.9	0.6	0.8
0.7	0.8	0.8	0.4	0.2	0.7	0.5	0.7	0.0	0.1	0.7	0.7	0.4	0.8	0.3	0.8	0.9	0.7	0.4
0.7	0.8	0.8	0.3	0.1	0.5	0.4	0.4	0.1	0.0	0.8	0.8	0.4	0.9	0.3	0.5	0.7	0.9	0.6
0.4	0.1	0.3	0.7	0.8	0.9	0.5	0.5	0.7	0.8	0.0	0.2	0.4	0.6	0.9	0.9	0.8	0.3	0.8
0.6	0.2	0.3	0.8	0.7	0.8	0.7	0.4	0.7	0.8	0.2	0.0	0.1	0.3	0.8	0.9	0.7	0.3	0.9
0.6	0.3	0.3	0.4	0.5	0.6	0.4	0.2	0.4	0.4	0.4	0.1	0.0	0.4	0.6	0.6	0.7	0.5	0.8
0.7	0.6	0.2	0.7	0.8	0.4	0.8	0.8	0.9	0.6	0.3	0.4	0.0	0.7	0.6	0.1	0.2	0.8	
0.9	0.9	0.4	0.8	0.3	0.1	0.7	0.7	0.3	0.3	0.9	0.8	0.6	0.7	0.0	0.6	0.8	0.6	0.1
0.5	0.9	0.6	0.5	0.4	0.6	0.9	0.9	0.8	0.5	0.9	0.9	0.6	0.6	0.6	0.0	0.7	0.9	0.9
0.8	0.8	0.5	0.8	0.9	0.6	0.9	0.9	0.9	0.7	0.8	0.7	0.7	0.1	0.8	0.7	0.0	0.7	0.9
0.2	0.5	0.6	0.3	0.7	0.5	0.2	0.6	0.7	0.9	0.3	0.3	0.5	0.2	0.6	0.9	0.7	0.0	0.8
0.9	0.9	0.8	0.9	0.6	0.3	0.8	0.8	0.4	0.6	0.8	0.9	0.8	0.8	0.1	0.9	0.9	0.8	0.0

C'est assez indigeste et laborieux à créer, mais une fois cela fait, l'utilisation est très simple. Chaque ligne et chaque colonne correspond à un genre, dans l'ordre de la liste, et à la ligne i colonne j, on a la distance entre le genre i et le genre j. Les propriétés de la distance nous donne alors aussi des propriétés sur la matrice : puisqu'une distance est symétrique ( $d(a, b) = d(b, a)$ ), la matrice l'est aussi. De plus la distance entre un genre et lui même est nul, donc les éléments diagonaux sont tous nuls.

Pour créer notre distance entre 2 ensembles de genres, on va simplement parcourir les 2 ensembles, et faire la moyenne de toutes les distances que l'on trouve grâce à notre matrice. Ainsi, si le film 1 a  $n$  genres et le film 2 en a  $m$ , on va faire la moyenne de  $m \times n$  distances entre genres.

Il y a cependant un inconvénient à cet algorithme : il donne toujours des distances assez basses, même s'il permet de mieux ordonner. En effet on peut constater que la distance entre 2 ensembles identiques ne vaut pas toujours 0, ce qui peut poser certains problèmes.

#### 2.4.4 Recommandation de contenu

Une fois qu'on a une méthode qui nous calcule une distance entre 2 films, on peut facilement créer une méthode qui prend en argument un film, et qui nous en recommande d'autres. Il suffit pour cela de calculer la distance entre ce film et chacun des autres, puis de trier par ordre décroissant. Les premiers films de la liste sont alors ceux à recommander.

### 2.5 Système de recommandation collaboratif

Comme vu dans l'Etat de l'art, la distance entre deux utilisateurs est définie à partir de toutes les données recueillies sur eux, ce qui comprend :

- les données personnelles
- les données de navigation
- les notes attribuées par l'utilisateur
- et plus encore...

Toutefois, comme notre base de données ne contient que les notes des utilisateurs, nous avons décidé de ne prendre en compte que ces notes dans notre algorithme.

#### 2.5.1 Crédation d'une distance entre 2 utilisateurs

La première étape consiste à déterminer, à partir d'une liste de notes pour chaque utilisateur, une distance entre ces derniers. La solution retenue a été d'utiliser la corrélation de Pearson (détailé dans l'Etat de l'art). Définissons  $U_1$  comme étant l'utilisateur cible et  $U_2$  l'utilisateur que l'on compare.

On détermine dans un premier temps les notes moyennes pour  $U_1$  et  $U_2$  avant de calculer la corrélation de Pearson en ne prenant en compte que les films vus par  $U_1$  et  $U_2$ .

Il en résulte une note comprise entre  $[-1; 1]$ . Cependant, pour la suite des calculs, nous avons besoin d'une note comprise dans l'intervalle  $[0; 1]$ . On ajuste donc le coefficient de Pearson à l'aide de la formule :

$$distance(U_1, U_2) = \frac{1 + coef_{Pearson}}{2}$$

Maintenant qu'il est possible de calculer la distance entre deux utilisateurs, il suffit de réitérer cette opération pour tous les autres utilisateurs. On obtient alors une liste de  $n-1$  couples ( $identifiant_{U_i}, distance(U_1, U_i)$ ) avec :  $0 \leq i < n; i \neq 1$

### 2.5.2 Estimation d'une note

L'objectif est maintenant de pouvoir estimer la note que  $U_1$  donnera à un film donné en se basant sur celles des autres utilisateurs qui ont déjà noté ce film. Pour cela, on applique la formule suivante.

Soit  $x$  un film donné et  $U$ , l'ensemble des utilisateurs ayant notés ce film. On a :

$$Note\_estime(x) = \frac{\sum_{i \in U} sim_i.note_i}{|sim_i|}$$

On effectue donc ce calcul pour tous les films non notés par  $U_1$  et obtenons ainsi une liste non triée de tous les films non vus par  $U_1$ . Il ne reste alors plus qu'à les trier.

### 2.5.3 Tri des films

Pour trier les films en fonction de la note estimée, on applique un algorithme récursif inspiré des algorithmes MergeShort et QuickShort. Le principe est le suivant :

- On prend le premier élément de la liste qu'on considérera comme un pivot et qu'on stocke dans un nouvelle liste appelé "centre"
- Pour tous les autres éléments de la liste :
  - si l'élément est supérieur au pivot, on le stocke dans la liste appelé "droit"
  - si l'élément est inférieur au pivot, on le stocke dans la liste appelé "gauche"
  - si l'élément est égal au pivot, on le stocke dans la liste "centre"
- On réitère ce même algorithme jusqu'à ce qu'on ait plus que 1 ou 0 éléments dans les listes "gauche" et "droite" (étape récursive)
- Enfin on concatène l'ensemble dans l'ordre : "gauche" + "centre" + "droit"

La liste obtenue est maintenant triée dans l'ordre croissant des notes estimées.

## 2.6 Structures de données

Les structures de données du projet peuvent être séparées en 2 types :

- Statiques
- Dynamiques

Les structures statiques sont celles qui sont directement inscrites dans le code du programme C, écrites en dur dans le code, ce sont des structures de bas niveau. Ce sont les structures que l'on va définir avec le mot-clé struct en C, et qui sont destinées à contenir des données connues par avance, un tableau, une autre structure, des variables, etc...

Les structures dynamiques sont celles qui utilisent une autre structure plus flexible et qui permet l'agencement de données pour créer d'autres structures de données. Ce sont les structures basées sur le JSON, qui vont utiliser lib cJSON, et qui permettent d'ajouter et supprimer dynamiquement des éléments via des fonctions. Pour ces structures nous avons défini un fichier de référence en Markdown sur le dépôt git, afin que le programme utilise un format uniforme de données, mais une structure JSON permet en réalité de représenter n'importe quelle structure de données.

## 2.6.1 Structures de données statiques

### Structures provenant de libcommon

Les structures suivantes proviennent de libcommon :

- String\_t : Structure représentant une chaîne de caractère, avec de nombreuses fonctions associées.
- AutoString\_t : Structure identique à String\_t, pointeur automatiquement géré en mémoire.
- Map\_t : HashMap permettant l'association entre un String\_t et un void\*
- Vector\_t : Tableau dynamique permettant le stockage de void\*

La documentation détaillée de libcommon se trouve à cette adresse :

<https://tanaka1238.ovh/libcommon-doc>

### Structures provenant de libcjson

Les structures suivantes proviennent de libcjson :

- JSONObject\_t : Représente un objet JSON (associatif)
- JSONArray\_t : Représente un tableau JSON (dynamique)
- JSONInt\_t : Représente un nombre entier en JSON
- JSONDouble\_t : Représente un nombre flottant en JSON
- JSONString\_t : Représente une chaîne de caractère en JSON
- JSONNull\_t : Représente une valeur nulle en JSON
- JSONBoolean\_t : Représente une valeur booléenne en JSON

Ces structures sont utilisées pour créer les structures dynamiques expliquées plus bas. Elles permettent aussi de parser un JSON, le manipuler, et exporter les données.

La documentation détaillée se trouve à cette adresse :

<https://tanaka1238.ovh/libcjson-doc/>

### Structures provenant de pthread

- pthread\_t : Structure représentant un thread (fil d'exécution)
- pthread\_mutex\_t : Structure représentant un mutex, permettant de gérer les accès concurrents aux structures de données

### Structures provenant de GTK+-3.0

- GtkWidget : Représente un widget GTK+
- GtkWindow : Représente une fenêtre GTK+
- WebKitWebView : Représente un moteur de rendu Webkit pour GTK+
- GtkApplication : Représente une application GTK+
- GtkDialog : Représente une boîte de dialogue GTK+

### Base de données

```
struct BDD
{
    JSONObject_t json;
    pthread_mutex_t mutex;
    Vector_t clients;
    Map_t requests;
};

typedef struct BDD* BDD;
```

Cette structure a été créée en vue d'une utilisation côté serveur, elle contient l'ensemble des données correspondant aux films et aux utilisateurs dans une structure JSON dynamique expliquée plus bas, ainsi qu'un mutex permettant de gérer les accès concurrents des différents clients, un tableau

dynamique contenant les différents clients sous forme de Client\*, et une hashmap contenant des associations entre un nom dans le protocole réseau, et une fonction (pointeur de fonction).

### Client

```
struct Client
{
    int fd;
    BDD bdd;
    pthread_t thread;
    pthread_mutex_t mutex;
};

typedef struct Client* Client;
```

Cette structure a été créée en vue d'une utilisation côté serveur, elle contient un descripteur de fichier (fd) correspondant au socket réseau d'un client, un pointeur vers la base de données du serveur, un pthread\_t correspondant au thread sur lequel est géré le client par le serveur, et un mutex permettant de gérer les accès concurrents à la structure.

### Connexion

```
struct Connexion
{
    String_t addr_s;
    String_t sid;
    JSONObject_t user;
};

typedef struct Connexion* Connexion_t;
```

Cette structure a été créée en vue d'une utilisation côté client, elle contient l'adresse du serveur, un ID de session utilisateur, et la représentation JSON d'un utilisateur au moment de la connexion de celui-ci.

### Date

```
struct Date
{
    int day, month, year;
};

typedef struct Date* Date_t;
```

Cette structure a été créée en vue d'une utilisation générale, elle représente une date, et des fonctions sont disponibles pour permettre sa conversion directe en JSON.

## 2.6.2 Structures de données dynamiques

### Film

```
{
    "Actors": [
        [ "", ... ],
    ],
    "Description": "",
    "Directors": [
        [ "", ... ],
    ],
    "Duration": 0,
    "Genres": [
        [ "", ... ],
    ],
    "Id": 0,
    "Title": "",
    "Type": "" ,
```

```

    "Year" : 0,
    "Rank" : 0,
    "Rate" : 0.0,
}
```

Cette structure représente un film dans la base de donnée, les films sont toujours traités par le programme sous cette forme, coté client comme serveur. Chaque film à un Id unique, qui commence à 1, et un titre. Les autres données sont facultatives en fonction des données que nous avons sur chaque film. Cette structure permet de gérer l'intégralité des données d'un film (ou d'une série). Les array en JSON permettent de stocker indépendamment chaque acteur, genre, réalisateur, etc...

### User (utilisateur)

```

{
    "Id" : 0,
    "Name" : "",
    "FirstName" : "",
    "Login" : "",
    "Password" : "",
    "Birth" : {Date},
    "Preferences" : { Preferences },
    "History" : { History },
    "Friends" : [0, ...],
    "Follow" : [0, ...]
}
```

Cette structure représente un utilisatuer dans la base de données, les utilisateurs sont toujours traités par le programme sous cette forme, coté client comme serveur. Chaque utilisateur à un Id unique qui commence à 1, ainsi qu'un login, et un mot de passe. Les autres données sont facultatives, et initialisées à leur valeur par défaut si inexistantes.

Cette structure permet de conserver les informations d'authentification d'un utilisateur. Initialement, il était prévu de pouvoir permettre la sauvegarde d'un historique, ainsi qu'un certain nombre d'amis, ou de personnes que l'utilisateur pourrait suivre, ce qui permettrait d'améliorer les recommandations. Cette fonctionnalité n'a pas été implémentée par manque de temps, mais reste envisageable pour des évolutions futures du projet. L'historique était prévu pour stocker l'intégralité des actions de l'utilisateur afin de pouvoir analyser son comportement et lui suggérer un contenu plus pertinent. Dans la version actuelle du projet, l'historique n'est utilisé que pour sauvegarder les notes que chaque utilisateur donne à un film, ce qui permet de lui suggérer du contenu, de générer des tendances et d'établir une liste des films vus par l'utilisateur.

### Date

```

{
    "Day" : 0,
    "Month" : 0,
    "Year" : 0,
}
```

En JSON, la date est représentée de cette façon.

### Preferences

```

{
    "StaticPreferences" : { StaticPreferences },
    "DynamicPreferences" : { DynamicPreferences }
}
```

Ces préférences ont été prévues pour permettre de nourrir l'algorithme de recommandation avec les préférences d'un utilisateur. Elles ne sont pas utilisées dans cette version du programme, mais pourraient permettre des recommandations beaucoup plus pertinentes une fois implémentée.

## StaticPreferences

```
{  
    "Hated" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    },  
    "Liked" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    },  
    "TopLiked" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    },  
    "TopHated" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    }  
}
```

Non utilisée dans cette version du programme, elle à été pensée pour stocker les préférences d'un utilisateur, afin d'améliorer les recommandations.

## DynamicPreferences

```
{  
    "Hated" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    },  
    "Liked" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    },  
    "TopLiked" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    },  
    "TopHated" : {  
        "Films" : [0, ...],  
        "Genres" : ["", ...]  
        "KeyWords" : ["", ...]  
    }  
}
```

Non utilisée dans cette version du programme, l'idée derrière les préférences dynamiques était d'utiliser l'historique de l'utilisateur et les notes moyennes en fonction des films pour générer des « traits » spécifiques à cet utilisateur.

En somme, cela permettra d'avoir un profil dynamique qui garde une mémoire de ses préférences, et qui évolue au fur et à mesure des interactions entre l'utilisateur et le programme, toujours dans le but de suggérer implicitement un contenu plus pertinent.

### **History**

```
{
  "Log" : [{ HistoryAction }, ...],
  "Viewed" : [{ HistoryViewed }, ...],
  "Searches" : [{ HistorySearched }, ...],
  "Recommended" : { HistoryRecomendation },
  "OldRecommended" : [{ HistoryRecomendation }, ...],
  "Displayed" : { HistoryDisplayed },
  "Rates" : [{ HistoryRate }, ...],
}
```

Cette structure a été prévue pour stocker toutes les actions d'un utilisateur. Dans cette version du programme, elle ne gère que les notes (Rates), mais sa conception aurait pu permettre, avec davantage de temps, de dresser un profil complet d'un utilisateur pour connaître ses habitudes, ses préférences, et faire évoluer les listes de films recommandés en fonction de ce qui attire son clic ou non.

### **HistoryRates**

```
{
  "Id" : 0,
  "Rate" : 0,
  "Liked" : 0,
}
```

Cette structure permet d'assigner une note à un film, pour chaque utilisateur. Dans la version actuelle du programme, seul le système de notes à été implémenté, le système de likes n'est pas présent.

### **HistoryAction**

```
{
  "Time" : 0,
  "Action" : "",
  "ActionId" : 0
}
```

Cette structure a été pensée pour sauvegarder les actions de l'utilisateur, elle n'est pas utilisée dans cette version du programme.

### **HistoryViewed**

```
{
  "FilmId" : 0,
  "Times" : [0, ...],
```

Cette structure a été pensée pour sauvegarder l'historique des films vus par l'utilisateur, ainsi que les dates auxquelles il les a consultés (timestamp). Elle n'est pas utilisée dans cette version du programme.

### **HistorySearched**

```
{
  "SearchId" : 0,
  "Query" : [ "", ... ],
  "FullText" : "",
  "Time" : 0
}
```

Cette structure a été pensée pour sauvegarder l'historique des recherches effectuées par l'utilisateur, les mots clés qui auront été retenus, etc... Elle n'est pas utilisée dans cette version du programme car nous aurions eu besoin de davantage de temps pour inclure une barre de recherche, mais l'historique de recherche aurait lui aussi permis d'établir un profil plus complet de l'utilisateur.

### **HistoryRecomendation**

```
{
    "Id" : 0,
    "Films" : [0, ...],
    "Time" : 0,
    "Genres" : ["", ...],
    "KeyWords" : ["", ...],
    "Used" : [0, ...]
}
```

Cette structure a été pensée pour sauvegarder l'historique des recommandations proposées à l'utilisateur, dans le but de pouvoir les faire varier si les éléments n'attirent pas le clic au bout d'un certain temps. Elle n'est pas utilisée dans cette version du programme mais pourrait constituer une amélioration prometteuse.

### **HistoryDisplayed**

```
{
    "FilmId" : 0,
    "Displayed" : [
        {
            "Time" : 0,
            "Position" : 0
        }, ...
    ]
}
```

Cette structure a été pensée pour garder une trace des recommandations qui ont été affichées à l'utilisateur, dans le but de pouvoir compter le nombre d'apparition de chaque recommandation, noter si elles ont attiré le clic ou pas, à quel moment elles ont été recommandées, dans quelle position, etc... dans le but d'avoir des recommandations changeantes si l'utilisateur ne semble pas intéressé par ce qu'on lui propose. Elle n'est pas utilisée dans la version actuelle du programme, mais pourrait beaucoup servir dans des améliorations futures.

### **Section**

```
{
    "SessionId": "",
    "UserId": 0,
}
```

Cette structure est utilisée exclusivement côté serveur, elle permet de faire une association entre une connexion d'un utilisateur, et son Id.

Lorsque que l'utilisateur envoie des identifiants valides, le serveur génère un nouvelle session, avec un Id de session associé. Cet Id est renvoyé au client qui l'utilise ensuite à la manière qu'un cookie de session pour réaliser des requêtes sans avoir à transmettre les identifiants à chaque fois.

### **Requête au serveur**

```
{
    "Sid" : "",
    "Type" : "",
    "Query" : {queryObject}
}
```

Cette structure est envoyée par le client, et receptionnée par le serveur pour faire une requête au serveur. Le champ « Sid »(peut être vide) correspond à la session courante de l'utilisateur. Le champ « Type »permet de spécifier le type de requête à traiter par le serveur. « Query »est un champ contenant les données à traiter. Vous trouverez davantage d'informations dans la partie expliquant le protocole réseau.

### Reponse du serveur

```
{  
    "Type" : "",  
    "Time" : 0,  
    "Code" : 0,  
    "Error" : "",  
    "Answer" : {answerObject}  
}
```

Cette structure permet de recevoir les réponses du serveur, « Type »est initialement conçue pour préciser le type de réponse dans le cas de multiples requêtes, mais n'est pas utilisé dans cette version car le protocole ne peut pas avoir ce type de conflits. « Time »est initialement prévue pour indiquer à quel moment a été effectuée la requête en cas de conflit avec de multiples requêtes, mais elle n'est pas utilisée dans cette version car le protocole ne peut pas avoir ce type de problèmes. « Code »contient le code de retour de la requête, voire le tableau contenant les codes de retour dans le détail du protocole. « Error »contient une chaîne représentant l'erreur (le cas échant) qui s'est produite lors du traitement de la requête. « Answer »contient les données de réponse de la requête.

## 2.7 Fonctionnement de l'application produite

### 2.7.1 Fonctionnement du serveur

Au démarrage, le serveur charge la base de données, il est le seul à y avoir accès. Il vérifie l'intégrité des données pour prévenir des crashs dans les autres fonctions plus tard, et supprime automatiquement les données invalides. Ensuite il écoute sur le port 10000 en tcp et attend de recevoir un client. Quand un client se connecte, le serveur crée un thread qui sera chargé d'interagir avec lui.

Ce thread attend d'abord l'envoi de données de la part du client, dès que celui-ci envoie un 0 (valeur entière) cela indique qu'il a fini d'envoyer. Il essaye alors de parser en JSON les données reçues avec JSONParser, si il y parvient, il interprète la requête, et la transmet à la fonction permettant son traitement en se basant sur le type de requête obtenue. L'association entre le type de requête et la fonction se fait via une HashMap de type Map\_t. En cas de problème le serveur retourne automatiquement une erreur au client. En cas de succès le serveur retourne une réponse au format JSON au client contenant les données attendues. Dans tous les cas, le serveur termine sa réponse par un 0 (valeur entière) afin d'indiquer au client qu'il a terminé. Suite à cela, la connexion se termine entre le client et le serveur.

La base de données du serveur est sauvegardée automatiquement par celui-ci à chaque modification et à l'extinction, et un backup est créé afin de pouvoir revenir en arrière en cas de problème.

### 2.7.2 Fonctionnement du client CLI

Le client en console nécessite d'avoir en paramètre l'ip ou le nom de domaine du serveur auquel se connecter. Dès son lancement, il vérifie que le serveur est joignable, puis affiche le menu principal.

Les menus de l'interface s'inspirent d'un fonctionnement « cisco », il suffit d'entrer le début d'un mot affiché pour que la commande complète soit reconnue, si plusieurs mots correspondent, l'interface affiche les choix possibles et demande à l'utilisateur de préciser sa demande.

L'interface console permet de faire autant de choses que l'interface graphique, elle permet de consulter les films, voir les tendances, voir des recommandations personnalisées, voir les films qui ont été notés, attribuer une note à des films, etc...

Les fonctions du menu principal sont dans un tableau et sont activées par un pointeur de fonction, le but de ce procédé est de rendre le code plus modulaire.

### 2.7.3 Fonctionnement du client GUI

Le client GUI peut recevoir en paramètre l'ip ou le nom de domaine du serveur auquel il doit se connecter, mais c'est facultatif, il le demandera à l'utilisateur si on ne lui donne pas. Une fois connecté au serveur, le client affichera la liste des films dans l'ordre de la base de données, on pourra alors se connecter à un compte, s'inscrire, avoir des recommandations personnalisées, voir les tendances, etc....

Le client utilise le moteur de rendu Webkit pour afficher l'interface, le code HTML donné au moteur est généré en interne par le client. Les liens cliquables sont interceptés, les requêtes web sont annulées, et remplacées par une exécution de code local. Les seules véritables requêtes HTTP réalisées sur l'interface sont celles permettant d'afficher le lecteur youtube.

Les pages « web » de l'interface sont réalisées à partir de fichiers json. exec ://main.json va demander au client de charger le fichier se trouvant dans : web/main.json. Ce fichier sera interprété par un générateur de code HTML maison qui génère l'intégralité de l'interface en se basant sur ce qui est présent dans le json. Les fichiers json permettent d'ajouter des fichiers css, des fichiers de scripts, des tags HTML, inclure des fichiers, ou appeler des fonctions C.

La majorité du code de l'interface est généré via des fonctions C, chaque fonction à la même signature, ce qui permet de faire une hashmap de pointeurs de fonction qui associe un nom à une fonction. Le nom est entré dans le json, ce qui permet à l'interface de choisir la bonne fonction, et les paramètres sont passés à la fonction depuis le fichier JSON. Il est possible de faire des appels récursifs de fonctions, par exemple, le lecteur youtube peut être appelé depuis un fichier JSON pour afficher une vidéo, mais il peut aussi être appelé depuis une autre fonction du générateur pour l'intégrer facilement dans la mise en page.

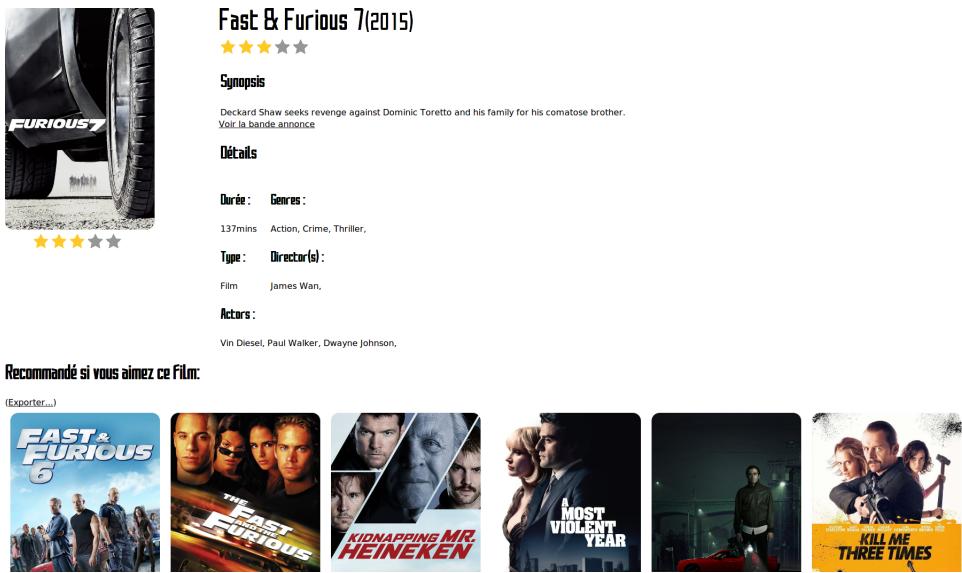


FIGURE 2.1 – Interface graphique de la page du film Fast & Furious 7

Les interactions directes avec un utilisateur, liens cliquables, formulaires et redirections sont assurées par la transmission dans l'URL de la requête à l'interface, ces paramètres sont extraits par le générateur HTML, et sont transmis par un Vector\_t (tableau dynamique de chaînes de caractères) aux fonctions permettant d'obtenir le code final qui sera exécuté par Webkit.

Une certain nombre de fonctions de Webkit ont été modifiées et/ou désactivées pour que l'interface ne puisse être utilisée que dans le cadre du projet, et non pas pour surfer n'importe où sur internet, ces modifications nous ont permis d'obtenir une interface stable et dans l'ère du temps.

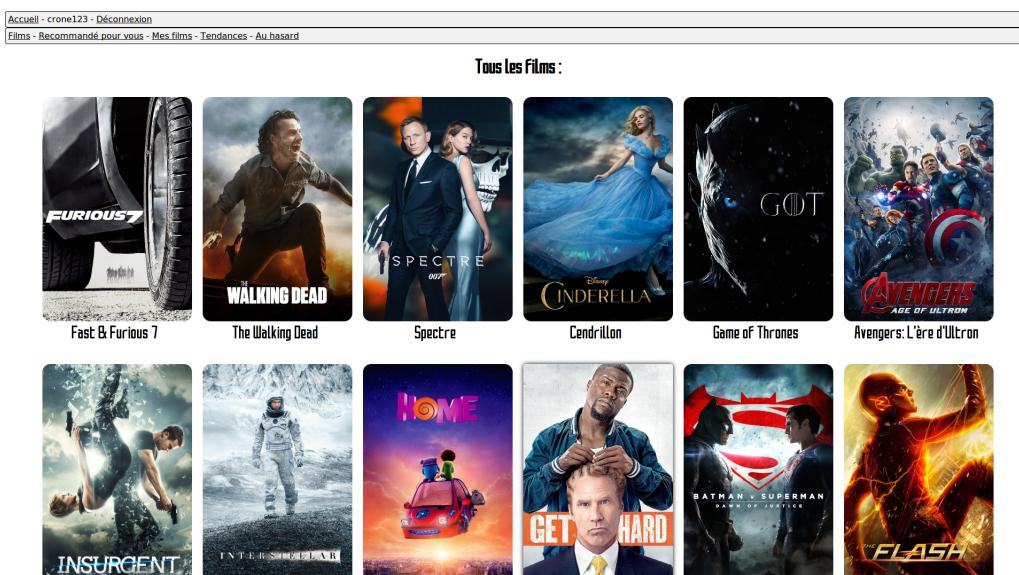


FIGURE 2.2 – Interface graphique du menu films

## 2.8 Protocole de communication entre client et serveur

Le protocole est basé sur un système similaire au REST qui utilise HTTP, cependant notre projet n'utilise pas HTTP. La requête s'effectue exclusivement en JSON, une fois la requête en JSON reçue par le serveur et selon la structure définie dans la section concernant les structure dynamiques, il va regarder si le type de requête correspond à ce qu'il sait gérer, si ce n'est pas le cas il retournera une réponse d'erreur.

Le serveur va d'abord vérifier si le Sid est défini dans la requête, auquel cas il tentera de vérifier la validité de la session en cours, et associera la session avec cette requête, ce qui permettra d'identifier l'utilisateur appelant dans la requête.

Dans le cas où le serveur sait gérer la requête reçue, il délègue le travail à une fonction prévue pour chaque type de requête via un pointeur de fonction, récupère son résultat, le formate pour être envoyé par le réseau et le transmet au client. Suite à la bonne transmission de la réponse, le client est déconnecté.

De son côté, le client reçoit le JSON de réponse, contenant les informations nécessaires à son bon fonctionnement. Exception : Si la connexion au serveur n'a pas pu se faire, la réponse est un pointeur null, ce cas est géré par le client qui sait alors que la connexion a échoué. Dans le cas où tout s'est passé correctement, le client commence par vérifier le code de retour de la fonction du serveur. « 0 » signifie que tout s'est bien passé, il procède alors au traitement des données le cas échéant, ou se contente simplement de savoir que sa demande a bien été traitée, car en effet, toutes les fonctions ne nécessitent pas forcément de traiter une réponse, la confirmation de bon traitement par le serveur peut suffire.

### 2.8.1 Table des codes de retour

Les codes de retour suivant ont été définis dans le protocole réseau du programme :

Code de retour	Description.
0	Succès.
1	Inconnu.
2	Nécessite d'être connecté.
3	Requête invalide.
4	Protocole invalide.
5	Données de requête invalides.
6	Erreur interne.
7	Identifiant de connexion invalide.
8	Identifiant de session invalide.

# Chapitre 3

## Gestion de Projet

### 3.1 matrice SWOT

	positif	négatif
interne	<b>FORCES</b> <ul style="list-style-type: none"><li>— plusieurs bibliothèques déjà existantes</li><li>— bibliothèques déjà créées</li></ul>	<b>FAIBLESSES</b> <ul style="list-style-type: none"><li>— Manque de certaines compétences, telle que le machine learning</li><li>— Potentiel manque de temps</li></ul>
externe	<b>OPPORTUNITÉS</b> <ul style="list-style-type: none"><li>— Un sujet très utilisé, donc beaucoup de sources disponibles</li></ul>	<b>MENACES</b> <ul style="list-style-type: none"><li>— les systèmes les plus performants sont souvent protégés</li></ul>

### 3.2 Estimation du temps passé sur le projet

	Nathan BARLOY	Valentin CRONE	Johan TOMBRE
Total	env 30 h	env 110h	env 70h

Tâches	Début	Durée	Fin	15		16		17		18																					
				9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4	5	6
Recherches	12/04/18	16 jours	27/04/18																												
Serveur/Client	23/04/18	19 jours	11/05/18																												
Code contenu	30/04/18	19 jours	18/05/18																												
Code collaboratif	30/04/18	26 jours	31/05/18																												
Écriture rapport	23/04/18	40 jours	01/06/18																												
Interface graphique	07/05/18	19 jours	25/05/18																												
Interface console	14/05/18	12 jours	25/05/18																												

Tâches	Début	Durée	Fin	19		20		21		22																					
				7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	1	2	3
Recherches																															
Serveur/Client																															
Code contenu																															
Code collaboratif																															
Écriture rapport																															
Interface graphique																															
Interface console																															

Diagramme Gantt

### **3.3 Comptes Rendus de Réunion**

## Compte rendu de réunion 1

Minutes for March 27, 2018, 16h00 - 50mins

**Present :** Nathan BARLOY, Valentin CRÔNE (Chair), Johan TOMBRE

**Absent :** aucun

### Announces

1. Nous avons commencé par désigner le chef de projet, qui sera désormais Valentin CRÔNE.

### Compte rendu

La réunion démarre à 16h00, nous avons d'abord désigné le chef de projet (Valentin CRÔNE), puis nous avons chacun proposé un point sur lequel discuter.

1. Johan à proposé qu'on réfléchisse aux différentes tâches à réaliser.
2. Nathan à proposé qu'on réfléchisse à la répartition des tâches.
3. Valentin à abordé la manière de travailler de manière flexible, en faisant des petits commits propres pour revenir facilement en arrière en cas de problème.

Les moyens de communication utilisés seront : Telegram pour les échanges, et Discord pour les éventuelles réunions à distance, ces moyens de communications avaient été définis avant cette réunion.

Nous en avons convenu qu'il était pertinent de d'abord rechercher les différents algorithmes et idées proposées existantes à partir de recherches effectuées par des experts du domaine, afin de pouvoir rédiger un état de l'art, et prendre le meilleur et le plus adapté pour concevoir un programme capable de répondre aux besoins du client.

Nous avons évoqué différentes idées qui pourraient intervenir dans le programme, au niveau de la recommandation, de l'interface graphique, etc...<sup>1</sup>

Durant cette réunion, Valentin CRÔNE a fait une explication à tout le groupe sur le format JSON, pour que tout le monde sache l'utiliser. Il a aussi été choisi d'utiliser une bibliothèque JSON compatible C/C++ développée par Valentin CRÔNE par le passé.

### Idées proposées (brainstorming)

1. Comptes utilisateur
2. Recommandation générale (utilisateur non connecté/utilisateur pour lequel on a pas d'info)
3. Sondage d'inscription (netflix ?)/régulier (interroger l'utilisateur sur ses préférences de manière naturelle et non intrusive, « Aimez vous tel ou tel film »)
4. Distance entre 2 films
5. Boutons j'aime/j'aime pas
6. Par rapport aux autres utilisateurs qui aiment un contenu similaire
7. Date limite de validité des données (pour garder un algorithme de recommandation qui s'adapte aux changements d'envies de l'utilisateur)
8. Compteur de fois affichées, ne plus proposer un film si il a été affiché trop de fois sans attirer l'utilisateur, classer/positionner correctement pour attirer le regard avant d'enlever le film des propositions.

---

1. cf : Brainstorming

## **Idée annulée**

Nous avions préalablement eu l'idée de créer un serveur HTTP permettant à plusieurs utilisateurs de se connecter simultanément via un navigateur web. Après proposition à notre client, cette idée a été annulée car le client à peur que notre système de recommandation ne fasse pas l'intégralité du traitement côté serveur. Nous sommes donc en recherche d'une nouvelle solution qui pourrait être agréable pour l'utilisateur.

## **Ordre du jour**

1. Désignation du chef de projet
2. Désignation des voies de recherches à explorer
3. Formation rapide JSON

## **A faire pour la prochaine réunion**

1. Trouver des documents de recherche sur les algorithmes de recommandation (Tous)
2. Essayer de transformer la BD en JSON (Valentin CRÔNE)
3. Rédiger le compte rendu propre de la réunion en latex (Valentin CRÔNE)
4. Commencer/finir la rédaction de l'état de l'art (Tous)

**Next Meeting :** April 12, 2018, 12h00

## Compte rendu de réunion 2

Minutes for April 12, 2018, 12h00 - 50mins

**Present :** Nathan BARLOY, Johan TOMBRE (Chair), Valentin CRÔNE

**Absent :** aucun

### Ordre du jour

Les éléments discutés lors de cette réunion ont été :

1. L'avancement du projet, ce qui comprend :
  - Détails sur le code produit, fonctionnement des librairies.
  - Discussion sur la structure interne des différents objets.
  - Choix du type d'interface graphique à utiliser.
2. Rappel sur l'importance d'un travail régulier.
3. Discussion sur comment être plus efficace sur ce projet.

### Compte rendu

La base de données a été transformé en JSON (Valentin) selon un schéma décrit dans la structure interne. Il a également été décidé que ce format de données sera celui utilisé pour toutes les informations à stocker. Valentin a donc expliqué au reste du groupe le fonctionnement de la librairie lib cJSON que nous utiliserons à l'avenir.

Nous avons ensuite échangé sur la structure interne de la base de donnée JSON du serveur. Nathan a fait remarquer qu'il est intéressant de ne calculer qu'une seule fois les similarités entre films au lancement du serveur puis de les stocker.

Le client nous a donné son accord pour l'utilisation de libcommon, lib cJSON, et lib json, un modèle client/serveur multiutilisateur, ainsi qu'un client graphique utilisant WebkitWebView de WebkitGTK+.

Nous avons conclu par un recadrage de l'équipe suite au déséquilibre de la proportion du travail effectué par chaque membre du groupe. Il a été décidé qu'il fallait effectuer un travail plus régulier. De plus, le groupe se réunira à présent 2 fois par semaine lors des réunions plus courtes pour faire le point ainsi qu'établir de nouveaux objectifs (réalistes) à réaliser pour la prochaine fois.

### A faire pour la prochaine réunion

1. Rédiger le compte rendu propre de la réunion (Johan TOMBRE)
2. Amorcer le code pour la mesure de similarités entre films (Nathan BARLOY)
3. Rédaction de l'état de l'art (Johan TOMBRE)
4. Continuer le code serveur / client (Valentin CRÔNE)

**Next Meeting :** April 16, 2018, 12h00

## **Compte rendu de réunion 3**

Minutes for April 16, 2018, 12h30 - 20mins

**Present :** Nathan BARLOY, Johan TOMBRE (Chair), Valentin CRÔNE

**Absent :** aucun

### **Ordre du jour**

Les éléments discutés lors de cette réunion ont été :

1. Explication du code concernant le serveur
2. Point sur l'avancement de l'état de l'art

### **Compte rendu**

On raison d'un imprévu dans l'emploi du temps d'un des membres du groupe, nous avons eu une réunion plus courte que prévu et n'avons pas pu aborder tous les points souhaités lors de la dernière réunion. Le fonctionnement du code côté serveur a été expliqué à l'ensemble de l'équipe par Valentin car c'est à partir du serveur que sera lancé le programme de recommandation. Il est donc indispensable que tous comprennent comment interagir avec.

Nous avons ensuite échangé sur l'avancement des recherches sur l'état de l'art.

### **A faire pour la prochaine réunion**

1. Rédiger le compte rendu propre de la réunion (Johan TOMBRE)
2. Poursuivre les recherches ainsi que le code pour la mesure de similarités entre films (Nathan BARLOY)
3. Poursuivre l'état de l'art (Johan TOMBRE)
4. Continuer le code serveur / client (Valentin CRÔNE)

**Next Meeting :** April 19, 2018, 12h00

## **Compte rendu de réunion 4**

Minutes for April 19, 2018, 12h - 15mins

**Present :** Nathan BARLOY, Johan TOMBRE (Chair), Valentin CRÔNE

**Absent :** aucun

### **Ordre du jour**

1. Point sur l'avancement depuis la dernière réunion.

### **Compte rendu**

Discussion sur la standart sur les distances / similarités entre films. Les résultats seront bornés entre 0 et 1. Une distance de 0 signifie que les films sont identiques alors qu'une distance de 1 signifie qu'ils n'ont rien en commun.

### **A faire pour la prochaine réunion**

La date de la prochaine réunion n'a pas été fixé en raison des vacances. Toutefois, le groupe continuera à communiquer via Telegram et Discord (pour des visio-conférences) pour faire le point et fixer de nouveaux objectifs. Les points sur lesquels il faut avancer d'ici là sont :

1. Rédiger le compte rendu propre de la réunion (Johan TOMBRE)
2. Terminer les calculs sur les similarités entre films (Nathan BARLOY)
3. Finir l'état de l'art (Johan TOMBRE)

**Next Meeting :** Non déterminé

## Compte rendu de réunion 5

Minutes for May 07, 2018, 18h - 25mins

**Present :** Nathan BARLOY, Johan TOMBRE (Chair), Valentin CRÔNE

**Absent :** aucun

### Ordre du jour

1. Montrer ce qui a été réalisé pendant ces vacances
2. Lancer les discussions sur la réalisation de l'interface graphique

### Compte rendu

Au cours des vacances, Nathan a travaillé sur le calcul de distance entre deux films, Johan a commencé le calcul de distance entre deux utilisateurs, basé sur les notes données et Valentin a travaillé sur l'interface console et sur les requêtes entre client et serveur.

Pendant cette réunion, nous avons débutés les discussions sur l'interface graphique : il faut commencer les premiers tests sur l'utilisation de la librairie GTKWebKit. Nathan a expliqué les améliorations qu'il va implémenter sur les distances entre films, notamment pour utiliser un système de calcul (autre que Jaccard) pour établir les distances entre les différents genres qui ont des liens entre eux. Cela permettra d'améliorer les résultats. Valentin a expliqué au reste du groupe le fonctionnement des requêtes entre serveur et client pour que tous puissent commencer à écrire les différentes fonctions de requêtes.

### A faire pour la prochaine réunion

L'objectif est de commencer à travailler sur les points suivants avant la prochaine réunion pour parler d'éventuels problème rencontrés :

1. Corriger les bugs sur le calcul de la distance de Pearson + écrire le compte rendu de réunion + écrire le rapport sur sa partie (Johan TOMBRE)
2. Améliorer le calcul des distances entre films + écrire sa partie du rapport (Nathan BARLOY)
3. Continuer avec le protocole réseau + commencer les bases de l'interface graphique (Valentin CRÔNE)

**Next Meeting :** Vendredi 11 Mai 2018 à 13h

## Compte rendu de réunion 6

Minutes for May 14, 2018, 17h - 25mins

**Present :** Nathan BARLOY, Johan TOMBRE (Chair), Valentin CRÔNE

**Absent :** aucun

### Ordre du jour

1. Montrer les avancés sur l'interface graphique
2. Explication et démonstration des codes de requêtes
3. Répartition des tâches sur la fin du projet

### Compte rendu

Valentin a codé la base de l'interface graphique qui consiste à générer du code html à partir de fonctions en C pour être utilisé avec la librairie GTKWebKitWebView.

Johan a enrichit la base de données avec les affiches des films à l'aide d'un script python.

Nathan a réalisé le code permettant un calcul de la distance entre genres à partir d'une matrice.

### A faire

Les attributions des tâches sont :

1. Générer une base de données utilisateur afin de tester l'algorithme de distance entre utilisateurs (Johan TOMBRE)
2. Finir l'algorithme collaboratif = estimations de notes et tri des résultats (Johan TOMBRE)
3. Réalisation des outils de gestion de projet (Nathan BARLOY)
4. Réalisation de l'interface graphique (Valentin CRÔNE et Johan TOMBRE)
5. Réalisation interface console (Valentin CRÔNE)
6. Gestion des exports (Valentin CRÔNE)
7. Rédaction du rapport (Tous)

### 3.4 Post-mortem

L'objectif de ce projet était de réaliser un système de recommandations de films. Le code fournit doit être produit en C.

Après un peu plus de 2 mois, notre code comprend :

- Une interaction client / serveur
- Une base de données au format JSON
- Un système de recommandation basé sur le contenu
- Un système de recommandation collaboratif
- Une interface console
- Une interface graphique

L'ensemble fonctionne correctement et présente l'avantage d'être modulaire : il est donc possible de faire évoluer le programme afin d'y inclure davantage de fonctionnalités à l'avenir.

L'ensemble du groupe est satisfait du résultat obtenu. Les éléments ayant bien fonctionnés au cours de ce projet sont le workflow utilisé (très effectif) ainsi que le code produit de façon très modulaire.

Toutefois, nous pouvons également soulever quelques aspects ayant moins bien fonctionnés, comme par exemple le lancement du projet. En effet, nous ne nous connaissions pas avant le début du projet et avons eu du mal à travailler ensemble et à nous répartir les tâches correctement. Cela s'est toutefois grandement amélioré au cours du projet, avec notamment la mise en place de réunions plus fréquentes.

Pour conclure, ce projet a été très bénéfique pour l'ensemble du groupe, tant sur le plan didactique que relationnel.