



UNIVERSITÉ
LAVAL

PROJET DE COURS

GLO-2005 - HIVER 2019 – EQUIPE 10

Stéphane Galibert
Joffrey Escobar
Raphael Bouchard

TABLE DES MATIÈRES

Table des matières

Problématique	1
Démarrage.....	1
Modèle entité relation	2
Modèle relationnel.....	3
Peuplement de la base de données	4
Implémentation et fonctionnalités.....	5
L'espace connexion	5
La page d'accueil :	6
Les pages de catégories :	7
La page d'affichage :	8
Le profil acteur	9
La recherche.....	10
Le profile utilisateur	11
L'indexation des données et l'optimisation des requêtes	12
La sécurité du système.....	12
L'organisation, la gestion de l'équipe et la division des tâches.....	13
Accès aux sources.....	13

PROBLEMATIQUE

Pour ce projet de session nous avons décidé de faire un site de notation de diffèrent support audio-visuel, semblable au site « sens critique », qui permet donner son avis sur une large liste de film, série, jeu, livre, ...

Pour notre version du site nous avons décidé de nous limiter aux films, séries et jeux vidéo. Afin de pouvoir interagir avec cette base de données le site doit proposer plusieurs fonctionnalités :

- Créer un compte et pouvoir se connecter sur un compte unique sécurisé par un mot de passe ;
- Noter un des éléments proposés avec une note entre 1 et 10, la note doit pouvoir être modifier ou supprimer par la suite ;
- Commenter un élément, le commentaire doit pouvoir être modifier ou supprimer par la suite ;
- Ajouter ou supprimer un élément qui intéresse l'utilisateur ;
- Voir la liste des éléments dans la base de données en utilisant plusieurs filtres : les mieux notés, les plus commentés ou les plus les éléments démontrant le plus d'intérêt ;
- Faire une recherche parmi les éléments ainsi que les acteurs associés aux éléments ;
- Voir le profil d'un membre ainsi que ses interactions ;
- Voir tous les films et séries auquel un acteur est rattaché.

Les opérations d'administration ne seront volontairement pas implémentées, l'ensemble des données seront générer par un script, en utilisant des données en provenance d'une API publique, le but à la fin du projet est d'avoir 80-100 éléments dans chaque catégorie (jeux vidéo, films et séries) accessible en utilisant de vraies données pour donner une impression de réalisme a la plateforme.

DEMARRAGE

Avant de commencer à architecturer la base de données, nous avons décidé de voir ou nous pourrions trouver des données pertinentes afin de peupler notre base de données. Pour cela, nous avons trouvé une API non officielle du site IMDB qui nous permet de récupérer des informations sur les trois domaines que nous avons choisi. Les résultats de l'API étant intégralement en anglais, nous avons décidé de développer la plateforme dans cette langue.

En fonction des éléments que nous pouvions récupérer via cette API nous avons sélectionner les champs qui nous allons utiliser dans notre propre base de données. Nous avons aussi décidé d'utiliser l'api de Google Image afin de trouver une photo des acteurs qui ne nous été pas proposé par l'API d'IMDB.

Les données destinées à simuler les interactions des utilisateurs avec la base de données sont générées de façon pseudo-aléatoire en utilisant une librairie JS « Faker » afin d'avoir des informations cohérentes.

MODELE ENTITE RELATION

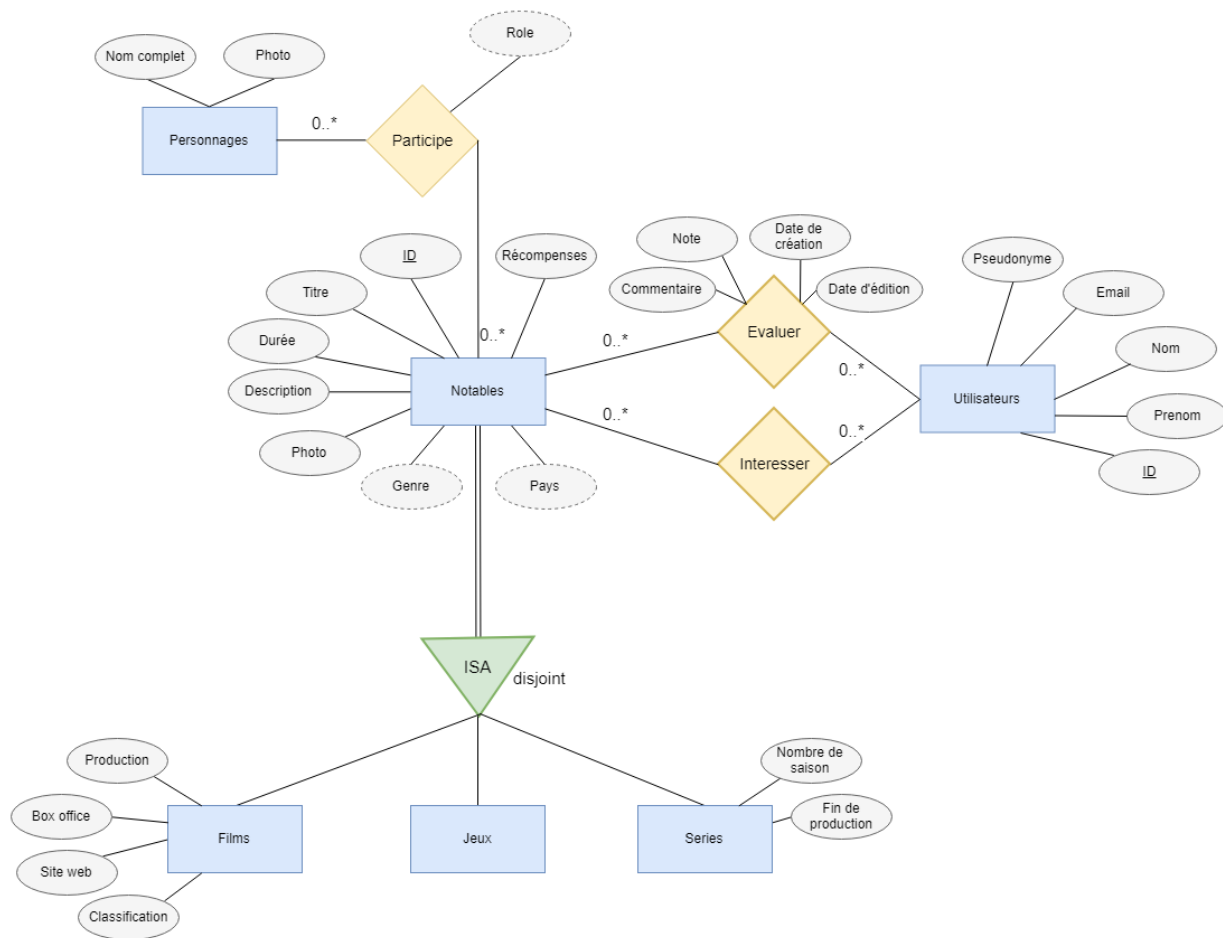


Figure 1 - Diagramme Entité - Relation

Le diagramme entité-relation respecte les spécifications du projet, en ajoutant les champs récupérables par l'API IMDB. L'ensemble est très classique, on peut néanmoins noter la présence de plusieurs attributs dérivés tel que *pays*, *genre* et *rôles* afin d'éviter les redondances et pouvoir faire des recherches facilement via ces éléments, même si cela n'est pas géré dans la portée du projet pour ce cours.

MODELE RELATIONNEL

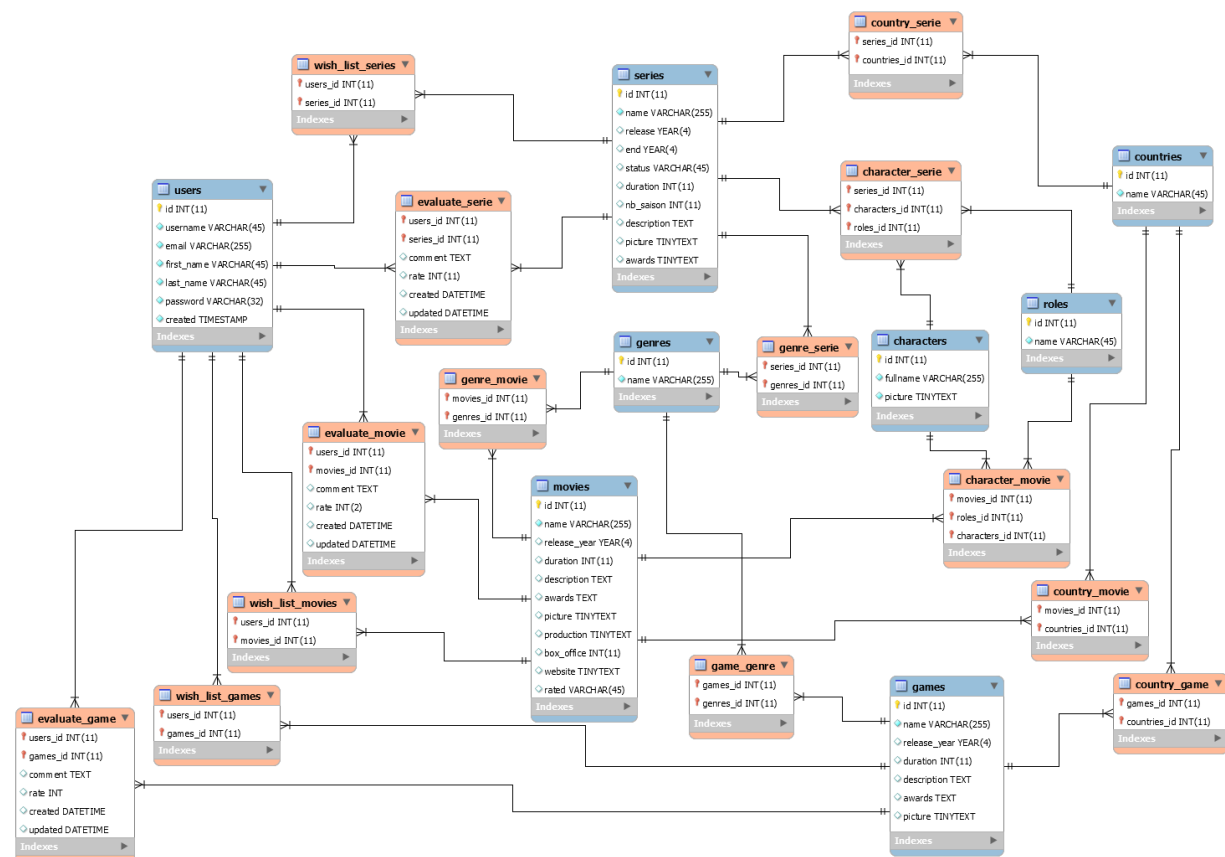


Figure 2 - Modèle relationnel réalisé avec MySQL Workbench

Le modèle relationnel découle logiquement du modèle entité-relation, on retrouve toutes les entités présente dans le schéma ci-dessus en bleu. Le principal point particulier réside au dédoublement de l'entité « notable » qui est transformé en trois tables (*game*, *movie* et *serie*). Nous avons fait ce choix, plutôt que d'ajouter des champs vide afin de faciliter les maintenances du projet, si nous voulions ajouter des tables afin de noter des musiques ou des livres il suffirait d'ajouter les tables correspondantes. De plus, nous avons créé plusieurs vues afin de faciliter les recherches et affichage groupé d'éléments, en modifiant simplement ces vues, il est possible d'ajouter facilement un comportement.

On peut aussi voir que la table *games* ne dispose pas de *characters* associé car l'API ne nous permet pas de les récupérer. Nous avons donc imaginé la base afin de ne pas être obligé pour chaque *notables* d'avoir des acteurs, et des genres et des pays associés.

PEUPLEMENT DE LA BASE DE DONNEES

Une fois la base de données terminée, nous avons créé un générateur en javascript de données pour ajouter un contenu réaliste. Les données sont importées comme dit précédemment de l'API du site IMDB afin d'avoir des valeurs réelles. Le reste des données est généré de façon pseudo aléatoire en utilisant Faker.js qui nous permet d'avoir des noms, email, ... cohérent pour nos utilisateurs.

Le script nous permet plusieurs commandes :

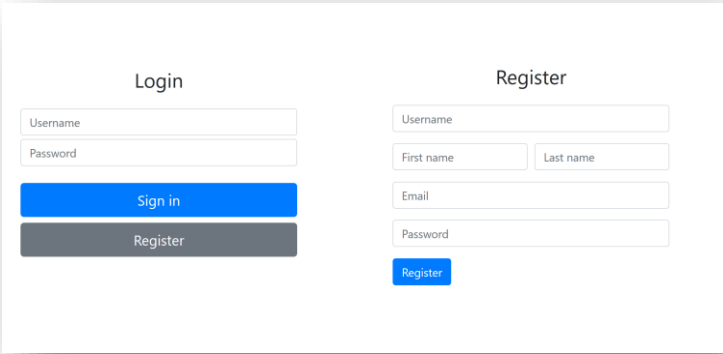
- `node .\generator\index.js init` : Permet de remplir les tables dont les champs sont fixes (rôles, ...).
- `node .\generator\index.js create_user` : Permet de créer 1000 utilisateurs aléatoire ayant des valeurs cohérentes.
- `node .\generator\index.js add_movie {nom du film}` : Permet d'ajouter les meilleurs résultats correspondant à un nom donné à notre base de données.
- `node .\generator\index.js add_serie {nom de la serie}` : Permet d'ajouter les meilleurs résultats correspondant à un nom donné à notre base de données.
- `node .\generator\index.js add_game {nom du jeu}` : Permet d'ajouter les meilleurs résultats correspondant à un nom donné à notre base de données.
- `node .\generator\index.js add_activity` : Permet de simuler de l'activité sur le réseau, le script va ajouter à chaque membres une note entre 1 et 10 et des commentaires entre 0 et 30 éléments de chaque catégories. Cela va avoir pour effet de lisser les notes vers 5,5.
- `node .\generator\index.js add_picture` : L'api d'IMDB ne nous donne pas accès aux photos des acteurs, afin de rendre l'expérience plus interactive nous utilisons l'api de google Image afin de donner une photo aux acteurs. Néanmoins l'api n'autorise que 100 appels par jours, de ce fait beaucoup d'acteurs n'ont malheureusement pas de photo associée.

Ce script n'est donné qu'à titre d'information, il n'a pas été entièrement testé et peut contenir des erreurs de conception.

IMPLEMENTATION ET FONCTIONNALITES

Dans cette partie nous allons décrire chaque fonctionnalité de l'application et comment nous l'avons implémenté pour chacun des trois niveaux de l'application. Seuls les pages les plus importantes seront présentées ici.

L'espace connexion



The image displays a user interface for a web application with two distinct sections for user management. On the left, the 'Login' section contains two input fields labeled 'Username' and 'Password'. Below these fields are two buttons: a prominent blue 'Sign in' button and a grey 'Register' button. On the right, the 'Register' section contains five input fields: 'Username', 'First name', 'Last name', 'Email', and 'Password'. A single blue 'Register' button is positioned at the bottom of this section. The entire interface is presented within a white rectangular box with a subtle drop shadow.

Figure 3 - Page de connexion et d'inscription

Côté client :

Pour le côté client, l'espace connexion est séparé en deux pages, la première pour l'inscription d'un utilisateur, et la deuxième pour sa connexion. Dans les deux cas, tous les champs étant requis une première vérification consiste à vérifier que tous les champs sont remplis et voir si l'adresse email est au bon format en utilisant un regex.

Côté serveur :

Du côté du serveur, les données sont contrôlées pour vérifier que toutes les informations sont présentes, ensuite elles sont envoyées à la base de données.

Côté BD :

Dans le cas de l'inscription, les données sont ajoutées à la base de données, en cas d'erreur du a une duplication de l'adresse email par exemple, une exception est levée qui sera capturée par la logique d'affaire qui retournera une erreur à l'utilisateur.

La page d'accueil :

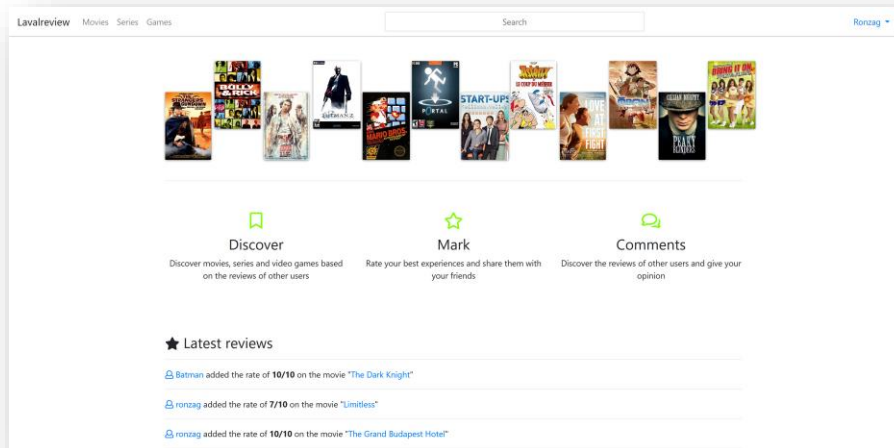


Figure 4 - Page d'accueil

Coté client :

La page d'accueil est une page qui contient seulement de l'affichage. Elle affiche 12 éléments aléatoire présent en base de données, ainsi que les 6 dernières notes qui ont été mise à un élément.

Coté serveur :

La logique d'affaire ne fait que récupérer les données qui lui sont transmises par la base de données.

Coté BD :

Pour les deux requêtes que nous faisons sur cette page, nous utilisons deux vues qui nous permette d'interagir plus facilement avec nos différents types dans la base de données. Ces deux vues sont des unions de certain champs de chaque table qu'il suffirait de modifier si nous rajoutions une nouvelle catégorie.

```
CREATE VIEW elements AS
SELECT
  id,
  name,
  picture,
  'movie' AS type
FROM movies
UNION SELECT
  id,
  name,
  picture,
  'serie' AS type
FROM series
UNION SELECT
  id,
  name,
  picture,
  'game' AS type
FROM games;
```

Figure 5 - Exemple de vue utilisé sur le projet

Les pages de catégories :

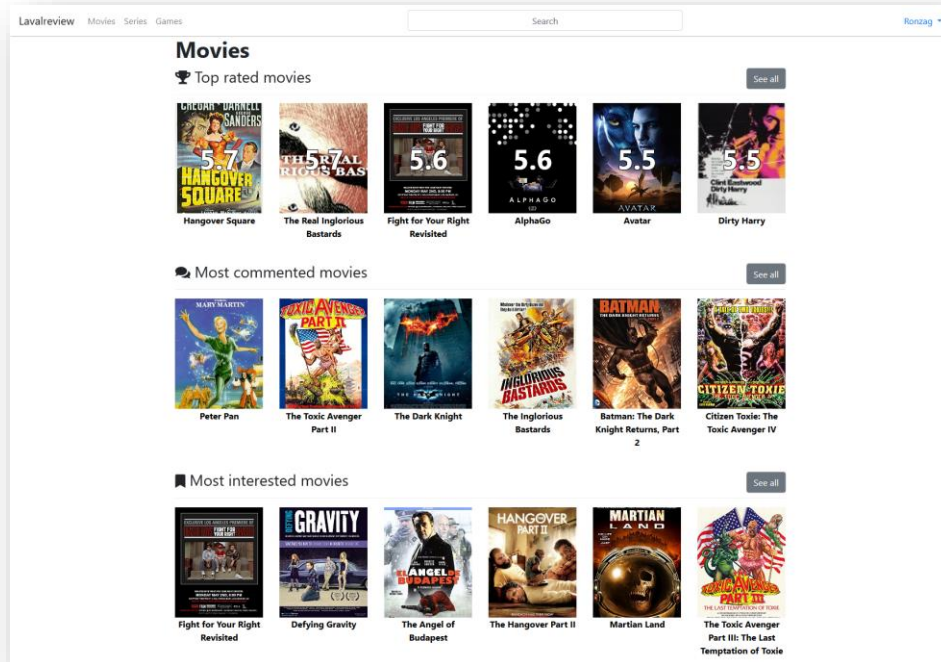


Figure 6 - Pages d'affichages des films avec plusieurs filtres

Coté client :

Cette page affiche les éléments en fonction des notes, du nombre de commentaire ou du nombre de personne qui ont ajouté un élément à leur « wishlist »

Coté serveur :

La logique d'affaire ne fait que récupérer les données qui lui sont transmises par la base de données.

Coté BD :

Pour cette page, trois requêtes sont implémentées chacune contenant une jointure avec les évaluations, les commentaires ou la liste de souhait.

La page d'affichage :

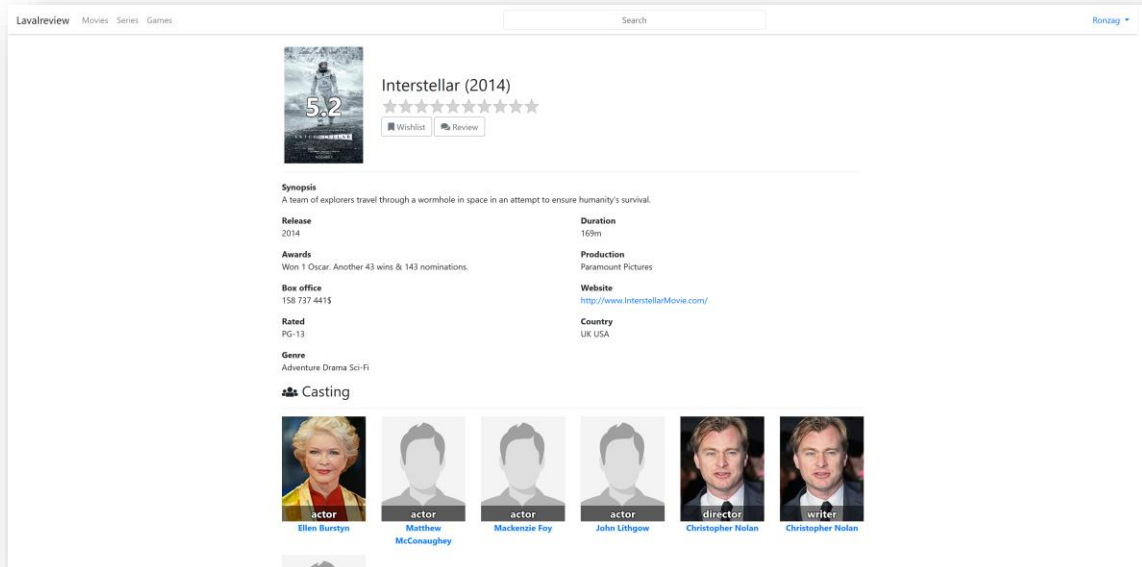


Figure 7 - Page du film Interstellar

Coté client :

Cette page contient beaucoup de données, déjà toute la fiche correspondante à l'élément sélectionné, le casting, la note moyenne et les commentaires avec un système de pagination. En plus de ça, l'utilisateur doit pouvoir soumettre un commentaire, une note ou ajouter l'élément à sa Wishlist. Lorsque que l'utilisateur clique sur « review » une fenêtre modale s'ouvre et lui permet de rentrer un commentaire. S'il a déjà mis un commentaire il peut le modifier ou le supprimer en validant un commentaire vide. De même si l'utilisateur a déjà mis une note, cette note apparait et il peut la modifier.

Si l'utilisateur n'est pas connecté, les boutons et la possibilité de noter l'œuvre ne sont pas disponibles.

Coté serveur :

Dans un premier temps, le serveur vérifie que l'id passé en paramètre est bien présente et existe sinon l'utilisateur est redirigé sur l'accueil. Ensuite toutes les autres requêtes sont effectuées. Le serveur regarde aussi si l'utilisateur est connecté pour afficher les boutons d'édition.

Coté BD :

Coté base de données les requêtes sont très classique, sauf pour l'ajout et l'édition de commentaire. Pour cela nous utilisons la syntaxe « ON DUPLICATE KEY UPDATE » qui nous permet de ne pas avoir à nous soucier de savoir si l'utilisateur a déjà évalué cet élément pour effectuer la mise à jour et ainsi optimiser le nombre de requêtes.

Le profil acteur

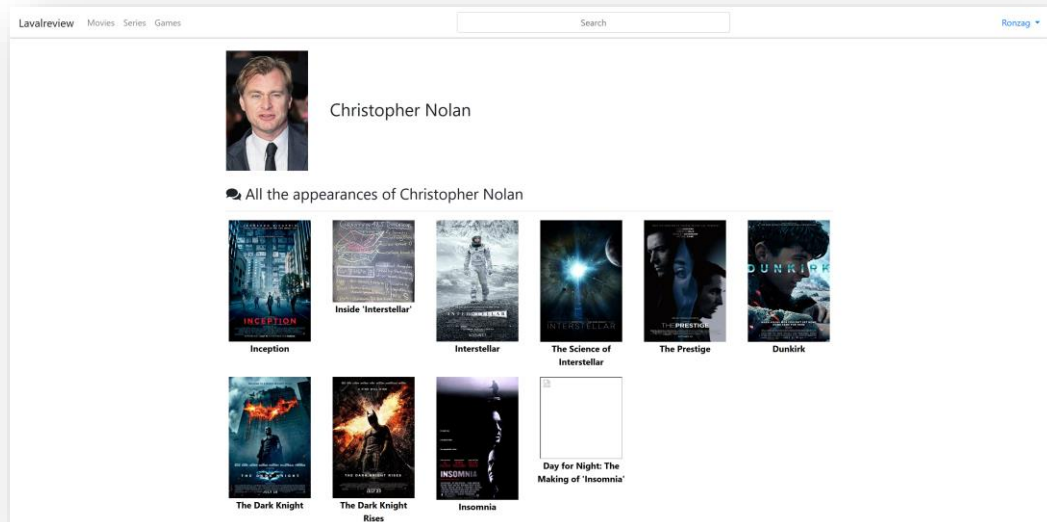


Figure 8 - Page d'acteur de Christopher Nolan

Coté client :

Cette page affiche tous les éléments dans lesquels on retrouve une certaine personne.

Coté serveur :

Le serveur vérifie que l'id passé en paramètre est bien présente et existe sinon l'utilisateur est redirigé sur l'accueil.

Coté BD :

Pour cette requête, nous n'avons pas utilisé une vue car cette page était le seul cas où nous voulons trouver un acteur en fonction de ses rôles, pour cela nous faisons une union des deux tables dans lesquels les éléments pourraient se trouver (films et series).

La recherche

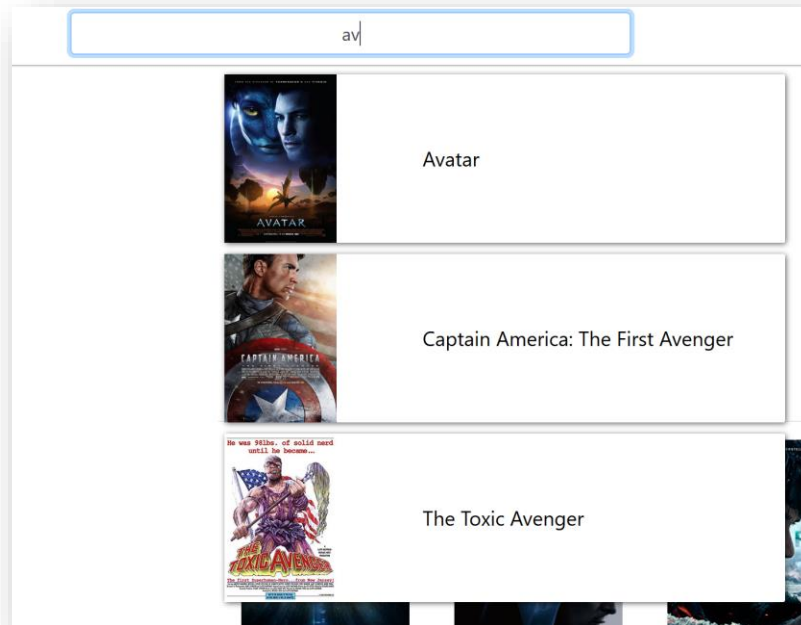


Figure 9 - Recherche

Coté client :

Cette fonctionnalité a demandé de faire un appel ajax afin de récupérer les résultats sans avoir à rafraichir la page. Elle permet d'afficher le meilleur résultat possible dans les éléments et les acteurs en fonction de la chaine passé, afin d'accéder à la page de l'élément.

Coté serveur :

La logique d'affaire va retourner un JSON contenant les trois résultats retournés par la base de données.

Coté BD :

Pour cette requête nous avons dû créer une troisième et dernière vue afin de rassembler les différents « notable » mais aussi les acteurs, ensuite nous utilisons la syntaxe « LIKE » afin de trouver les meilleurs résultats.

Le profile utilisateur

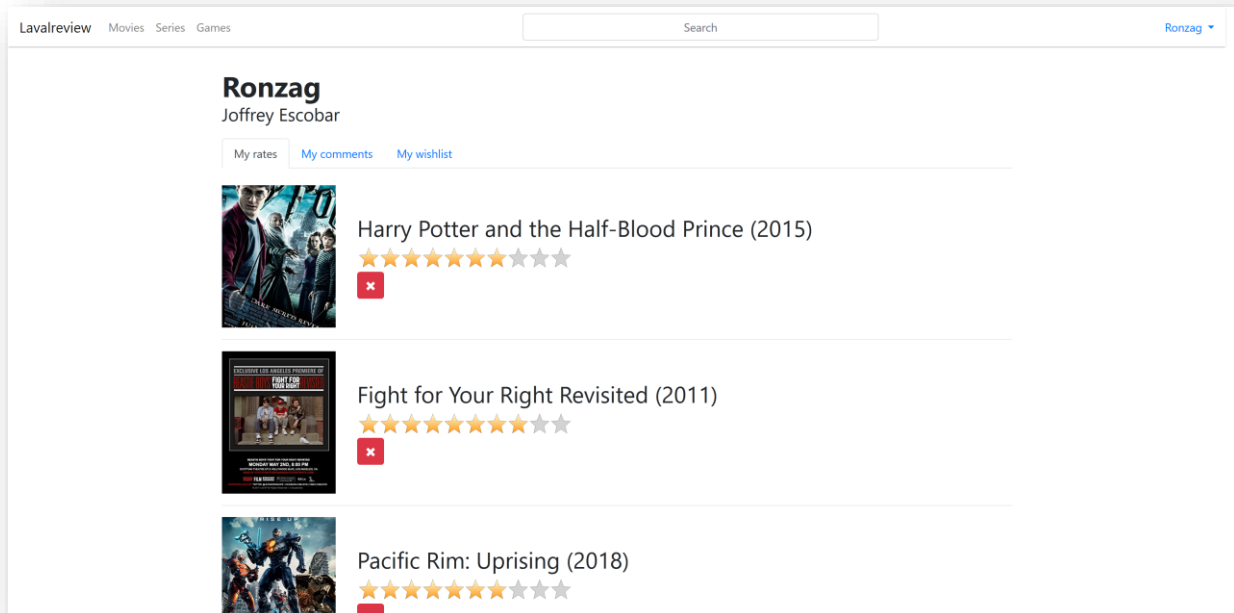


Figure 10 - page profile

Coté client :

Cette page est très complète, elle permet de voir et de modifier toutes ses interactions sur la plateforme (les votes, les commentaires et sa liste de souhait). Cette page est publique, il est donc possible de voir le profil d'un autre utilisateur mais il est alors impossible de modifier les éléments.

Coté serveur :

La logique d'affaire va vérifier si l'utilisateur est sur son profile ou sur un autre profile et en fonction autorisé l'édition ou non. Si l'id du profile n'est pas attribué, l'utilisateur est redirigé sur la page d'accueil.

Coté BD :

Les requêtes utilisé ici sont les mêmes que celle utilisé sur la page d'affichage des éléments.

L'INDEXATION DES DONNEES ET L'OPTIMISATION DES REQUETES

Afin d'optimiser les performances de notre base de données nous avons analysé parmi nos requêtes quels champs étaient les plus demandés. Nous avons trouvé plusieurs occurrences identiques. Vu que la plupart des données dans la base de données ne sont pas souvent mise à jour mais souvent sélectionnées, nous pouvons mettre des index sans qu'ils n'influent négativement les performances.

Lors des recherches et de l'affichages des éléments à l'écran nous récupérons à chaque fois l'id, le nom et l'image. Nous avons donc créé un index qui regroupe ces trois éléments pour les tables *characters*, *games*, *movies* et *series*. De plus nous avons ajouté un index de type FULLTEXT sur les champs « nom » de ces quatre éléments afin d'optimiser les recherches que nous effectuons dessus.

L'ensemble de tables de liaisons ont comme clé primaire la clé étrangère des deux autres tables afin d'optimiser les jointures et empêcher les duplications.

Une optimisation possible serait de dupliquer les données des tables rôles et genres à l'intérieur des tables des éléments afin de réduire les jointures et ainsi d'optimiser le temps d'exécution des requêtes mais cela entraînerait une augmentation non négligeable de la taille de ces tables.

LA SECURITE DU SYSTEME

Pour la sécurité de notre système, nous avons décidé d'utiliser PyMysql afin de faire nos requêtes en python et ainsi échapper les variables de nos requêtes pour éviter les failles d'injection SQL.

Pour la sécurité de nos utilisateurs, nous avons mis en place un système de connexion par mot de passe ; qui sont hash en base de données en MD5. De cette façon nous nous assurons de protéger les utilisateurs en cas d'intrusion dans notre base de données.

Enfin, les données sont vérifiées du côté utilisateurs en javascript, ainsi que du côté serveur en python, ou via les contraintes de la base de données en fonction des cas. Cela nous assure de conserver l'intégrité des données, mais aussi d'avertir l'utilisateur sans passer par le serveur.

L'ORGANISATION, LA GESTION DE L'EQUIPE ET LA DIVISION DES TACHES

Pour ce projet, nous avons organisé des rendez-vous à l'université ainsi qu'en ligne afin de parler du choix du sujet ainsi que les exigences du projet. Nous nous sommes entendus sur l'architecture de la base de données ensemble en faisant le diagramme Entités – Relations puis nous nous sommes divisé les tâches restantes.

Stéphane Galibert s'est occupé de l'architecture de la base de données, ses contraintes et les optimisations qui pouvait être réaliser via le design des tables.

Joffrey Escobar s'est occupé du design et du fonctionnement du site. Le côté serveur a été réaliser en utilisant Flask avec PyMysql afin d'interagir avec la base de données. Le côté client a été réaliser en utilisant Bootstrap comme Framework web, en utilisant du javascript et css.

Raphaël Bouchard a travaillé sur le rapport, les schémas et a aidé sur la partie optimisation.

ACCES AUX SOURCES

Le site est accessible sur github à l'adresse : <https://github.com/Escobaj/db>, vous retrouverez à cette adresse l'intégralité des sources du projets, le générateur de donnée ainsi qu'un « dump » de l'état final de la base de données contenant les données.

Un README inclut avec les source explique la procédure pour mettre en place la base de données et lancer le projet sur Windows, néanmoins la procédure doit être assez similaire sur Linux ou Mac.

En cas de problème pour lancer le projet, vous pouvez contacter Joffrey Escobar, qui s'est occupé du fonctionnel du site à l'adresse : joffrey.escobar.1@ulaval.ca