**Introduction**

**Présentation du projet**

**Descritpion rapide du site**

**Interface + explication d’utilisation**

**Chemin du site**

**choix d’implémentations**

**HomeServelt**

Dans la page index.jsp, le connection et l’inscription ce faisait via un modal.

Il y avait donc a gerer les Servlet de la page d’accueil, dde connection, inscription et déconnection.

Dans un premier temps, j’ai séparé ces Servlets mais dans chaque il y avait une partie dans le code qui était identique donc trop de rondondance inutile.

C’est pour cela que j’ai décidé de tout concentré en un seul servlet.

Pour cela , j’ai utilisé une des propriétés de l’annotaion @WebServelt qui permet de spécifier différents paternes d’url ( ici /Home , /Inscription….)

Ce qui permet à un seul servlet de traiter des requêtes provenant de diverses url.

Une fois que le servlet a été appelé, je vérifie l’URI (seulement la dernière partie de l’URL, sans le localhost :8080…) afin de savoir quel traitement effectuer.

**AdministrationTraitement et Payement**

Pour c’est deux page jsp, en fonction de l’action précédente de l’utilisateur elle est appliqué différament .

Pour le payement, qui est la meme pour achat, louer, et pour devenir premium nous avions besoins d’un moyen pour informé la page du type d’action attendu. Pour administrateur, il y a au totale 6 actions possibles sur cette page ( ajout, suppresssion client et film , et la generation de pdf audit ou du CA ) qui se déclenchent avec des boutons.

Nous avons donc choisie d’utiliser l’url pour transmettre ses informations à l’autre servlet.

**Modèle (Corentin Bouchaudon)**

Dans cette partie du projet il y avait trois sous partie majeur, la classe DataBase.java, le script d’insertion de données et les classes.java restante.

Concernant les classes.java, il n’y avait aucune difficulté à les coder il s’agissait juste de créer des classes ayant pour attributs les mêmes que ceux de la Base de Données. Il restait plus ensuite qu’à créer les constructeurs et les accesseurs nécessaire à l’utilisation des données.

Vient ensuite le script d’insertion de données, c’est ici que j’ai renconté ma première difficulté. En effet il m’a fallut plus de temps que prévu pour la réalisé. Ce retard est expliqué par des problèmes de cohérences de données, les données devant rester cohérente quelque soit la quantité. Le retard peut aussi être expliqué par le fait que rentrer les valeurs tests a pris plus de temps que prévu (entrer 100 résumé de film/série/documentaire est asser long).

Pour finir vient la classe DataBase.java. La première difficulté a été de réussir à créer la base de données. En effet, habituellement nous utilisions les Bases de Données de la facuté (BD local). Or nous n’avons pas accès à ces données depuis nos domicile, c’est ainsi que nous avons décidé d’installer postgresql sur nos machines respectives et de créer une BD cbouch3\_a qui a pour user cbouch3\_a (avec mdp cbocuh3\_a). Ce qui nous a permis d’economiser les aller retour à la faculté pour tester les fonctions.

La seconde a été le choix d’implementation des fonctions sur la BD, nous avions deux choix le premier écrire les select/update/insert/delete directement dans le code java (accès rapide en cas de changement ou d’erreur mais code moins lisible). Le second étant de créer des fonctions psql puis de les appelre dans le code java (code plus propre et fonction psql facielment modifial). J’ai décidé de prendre la première solution car dans un premier c’est plus facile pour mes camarades de comprendre ce que fait la fonction et dans un seconde temps je ne maitrise pas entière la création de fonction psql.

La dernière a été de réalisé certaine fonction qui était très longue à faire car avec les boucles for/while et les if je finissait pas m’embrouiller dans les valeurs (cf. rechercheVideoMC ou suggestions). Du coup je testais les fonctions une par une pour vérifier qu’elle fonctionne correctement ce qui a pris beaucoup de temps. Dans le code que l’on vous rend aujourd’hui toute les fonctions retournent les bonnes valeurs mais, elles ne sont pas obtimiser (cf. fonction énoncer ci-dessus)

**Conclusion**