# Licence d'Informatique 3ème année

## Projet de TIA : Plateforme de Communauté Virtuelle

### 1) Introduction

Le projet de TIA consiste à programmer en Java et en équipe de 4 étudiant-e-s, une plateforme de communauté virtuelle permettant aux utilisateurs d'interagir autour d'un centre d'intérêt commun. Les spécifications de l'application ne sont pas imposées de manière très précise : vous êtes relativement libres dans le choix de l'architecture de votre application et des fonctionnalités à implémenter.

L'objectif du projet est la mise en œuvre des technologies étudiées en TIA ce semestre. Les différentes parties de votre projet (réalisation, rapport, soutenance) seront évaluées en fonction de la maîtrise que vous montrerez de chacune des technologies. Il faudra donc prendre soin de bien mettre en avant l'utilisation que vous en faites dans votre application.

### 2) Plateforme de Communauté Virtuelle

#### 2.1) Fonctionnalités

Le projet consiste donc à programmer en Java une plateforme de communauté virtuelle permettant aux utilisateurs d'interagir autour d'un centre d'intérêt commun :

- Un centre d'intérêt commun : à vous de choisir le thème de votre communauté virtuelle (sports, loisirs, information, culture, études, etc.). L'important est que ce thème ne soit pas limitant pour le développement du projet et se prête donc bien à la construction d'une communauté virtuelle. On peut prendre en exemple la communauté virtuelle *LibraryThing*[1] qui rassemble 1,7 millions de membres amateurs de littérature ;
- Interagir : c'est-à-dire partager des données et des ressources et communiquer autour de ces données / ressources ;
- Données / ressources : la nature des données et des ressources que les membres de la communauté virtuelle partageront n'est pas important mais il faut qu'elle soit clairement identifiée et en adéquation avec l'objectif de la plateforme. Ainsi, on peut envisager que les ressources soient des documents textuels, des adresses Web, des images, des sons ou des vidéos, et que des données leur soient associées. Les ressources gérées par *LibraryThing* sont des livres, auxquels sont associées toutes sortes d'informations (auteur, éditeur, page de couverture, prix, commentaires, votes, fiches de lecture, librairies, etc.) ;
- Communiquer : partager des informations, discuter via une messagerie, un forum, un chat, etc. ;
- Et bien sûr, tout cela nécessite que les utilisateurs soient authentifiés lors de leur connexion à la plateforme (connexion, pseudo, mot de passe, gestion des sessions, déconnexion).

Outre le partage d'informations, on peut imaginer toutes sortes de fonctionnalités, comme par exemple :

- Commentaires sur les ressources partagées ;
- Notes / recommandations sur les ressources partagées ;
- Construction collaborative d'une base de connaissance à la Wikipédia ;
- Forum / messagerie / chat ;
- Fil d'actualités / « murs » ;
- Groupes, liens d'amitiés / de contacts ;
- Connexions multiples : une communauté virtuelle n'est intéressante que lorsque plusieurs utilisateurs peuvent l'utiliser en même temps ;
- Public / privé : associer à chaque publication un degré de visibilité qui permet de restreindre son accès aux amis / famille / collègues (mode *Facebook*) ou à tout le monde (mode *Twitter*) ;
- Multimédia : gestion du partage d'images, de vidéos, de sons, etc. ;
- Etc.

[1] LibraryThing : http://www.librarything.com

Il n'est évidemment pas demandé de développer chacune de ces fonctionnalités, mais seulement quelques-unes. La liste est loin d'être exhaustive : vous pouvez proposer d'autres fonctionnalités qui vous semblent intéressantes. Vous pouvez partir à la pêche aux idées sur les réseaux sociaux, et laisser parler votre imagination !

### 2.2) Architecture : serveur, client lourd, client léger

Votre projet doit être développé suivant un modèle client-serveur dans lequel le serveur aura pour objectif de :

- permettre la création et la gestion d'une communauté virtuelle ;
- mettre en relation les utilisateurs de la plateforme ;
- héberger les ressources partagées au sein de la communauté virtuelle ;
- gérer la communication entre les membres de la communauté virtuelle.

Le client aura pour fonctionnalité de permettre aux membres d'une communauté virtuelle de se connecter au serveur et de gérer leurs activités dans cette communauté. Pour faciliter la vie des utilisateurs, vous développerez un client lourd et un client léger :

- Client lourd : une application Java que l'utilisateur doit télécharger et exécuter sur sa machine. L'interface graphique « lourde », bénéficiant de la puissance de *Swing*, permet d'obtenir la meilleure ergonomie ;
- Client léger : votre application doit également permettre à un utilisateur de se connecter via un simple navigateur Web, éventuellement sur un appareil mobile (par exemple un smartphone). Dans ce cas-là, les technologies mises en œuvre sont : HTML, CSS, Sevlets, JSP.

Attention : si le design et l'ergonomie peuvent différer entre le client lourd et le client léger, les fonctionnalités accessibles doivent être identiques.

## 3) Technologies à mettre en œuvre

L'objectif du projet est la mise en œuvre des technologies étudiées en TIA ce semestre. Vous devez donc montrer votre maîtrise de **chacune d'entre elles**. Les spécifications du projet ne détaillent pas la manière dont vous devez les utiliser : nous vous donnons quelques pistes / indications dans cette section.

### 3.1) Modélisation

Dans un premier temps, il va falloir réaliser une analyse exhaustive des éléments de votre application : identifier les différents acteurs qui interviennent dans votre application, identifier les différents rôles que ces acteurs peuvent endosser et les fonctionnalités auxquelles ils peuvent avoir accès.

Une fois cette tâche réalisée, il va vous falloir définir les *uses cases* de votre application. Pour cela, appuyez-vous sur un diagramme de cas d'utilisation. Ce diagramme aura pour principal objectif de représenter les liens entre les acteurs de votre système et les fonctionnalités offertes par l'application. Vous compléterez par la suite ce diagramme avec une description textuelle de deux ou trois scénarios concrets.

Aussi, afin de compléter votre modélisation, il vous est demandé de réaliser le *diagramme de classes* afin de décrire les objets-clés de votre application. Vous veillerez à spécifier pour chaque classe ses attributs, ses méthodes et le type de relations que peut avoir la classe avec d'autres classes.

Enfin, afin d'illustrer les scénarios que vous avez préalablement définis, il vous est demandé de réaliser les *diagrammes de séquence* correspondant à chaque scénario.

### 3.2) Réseau

Pour la partie réseau, il vous est demandé de mettre en place une architecture 3-tiers (ou n-tiers) à travers laquelle vous procéderez à un découpage logique de votre application. Vous mettrez donc en œuvre le schéma suivant :

Client ←→ Serveur d'application ←→ Serveur de données.

Le serveur doit donc centraliser la majorité des échanges d'informations entre les utilisateurs. Toutefois, un certain type d'informations (par exemple les messages privés) circuleront d'utilisateur à utilisateur, sans passer par le serveur. Il s'agit donc d'une architecture pair-à-pair :

Client ←→ Client

#### 3.2.1) Le client

Dans un premier temps, faites une version basique de votre communauté virtuelle, qui ne comporte que deux utilisateurs connectés en utilisant des sockets TCP. Faites en sorte que le premier se lance en écoute (avec un numéro de port connu) via une socket d'écoute comme vu en TD et sans essayer de se connecter à l'application d'un autre utilisateur. Ensuite, lancez l'application du deuxième utilisateur qui aura pour mission de se connecter à celle du premier utilisateur. On obtient ainsi une situation simple, avec laquelle vous pouvez commencer à faire vivre la communauté virtuelle (assez réduite pour le moment). En traitement des commandes envoyées, faites déjà en sorte que les deux applications puissent s'envoyer des informations (par exemple des messages privés) et que chacune affiche l'information envoyée par l'interlocuteur. Vous devriez ainsi obtenir une version simpliste, partielle mais fonctionnelle avec deux applications Java.

#### 3.2.2) Le serveur d'application

Une fois l'implémentation de la communication pair-à-pair terminée, on développera une application Java pour jouer le rôle du serveur, qui aura pour tâche de centraliser les informations envoyées par les membres de la communauté (exceptées celles envoyées en pair-à-pair !). Cela permet, entre autres, à un utilisateur de récupérer les informations que les autres membres lui ont envoyé alors qu'il était hors-ligne. Pour cela, implémenter d'abord une version mono-utilisateur (un serveur dédié à une seule personne) : il suffit pour cela d'ajouter au schéma que vous avez développé plus haut une socket d'écoute à laquelle se connecte le client. Le serveur envoie alors au client toutes les informations qu'il a en cache.

Après, l'exécution peut reprendre de deux manières :

- soit le serveur continue de fournir les nouvelles informations au client,
- soit celui-ci se connecte directement (via une socket d'écoute) au flux des amis et récupère de lui-même ces informations.

Vous êtes libres de choisir celle qui vous semble la plus appropriée, cela dit, il faut faire attention à la cohérence des informations reçues.

Enfin, une fois votre version mono-utilisateur terminée, il faut envisager une version multi-utilisateurs, qui peut offrir les mêmes fonctionnalités à plusieurs personnes à la fois.

#### 3.2.3) Le serveur de données

Voir la section suivante.

### 3.3) Bases de données

Les ressources et les données de la communauté virtuelle devront être stockées dans une base de données MySQL. La structure des données devra être formalisée préalablement à l'aide d'un modèle Entité-Association.

### 3.4) XML

Lors d'échanges de données entre deux utilisateurs, comme par exemple l'envoi d'une partie de la liste des livres (et des données associées) d'un utilisateur de *LibraryThing* à un autre utilisateur de *LibraryThing*, ou d'un utilisateur au serveur, vous devez utiliser le format XML. Le client et le serveur doivent donc disposer de la / des DTD et ils doivent pouvoir analyser (lecture) et générer (écriture) les fichiers XML respectant cette / ces DTD, pour importer / exporter des informations.

### 3.5) Swing

L'interface graphique du client « lourd » sera réalisée à l'aide de *Swing* (cf. cours de POO du premier semestre).

### 3.6) Servlets, JSP et Applets

Le client « léger » sera basé sur les technologies des Servlets, des JSP et des Applets. Il vous faudra proposer une architecture pertinente combinant des servlets et des JSP, et imaginer une fonctionnalité pour laquelle l'interface graphique ne pourra pas se satisfaire des Servlets et des JSP et nécessitera le développement d'une applet.

### 3.7) Patrons de conception (Design Patterns)

Au cours de la conception et de l'implémentation de votre application, une attention particulière devrait être accordée au respect des patrons de conceptions vus en cours. Il s'agit également de rendre compte dans le rapport final des problèmes de conceptions rencontrés et des patrons utilisés pour y remédier. Par ailleurs, il faut veiller à ce que votre architecture respecte rigoureusement le patron de conception modèle-vue-contrôleur (MVC) étudié dans le cours. Tout projet ne se conformant pas à ce patron sera pénalisé car considéré « hors sujet ».

### 3.8) Cadre d'Application (Framework)

La mise en œuvre d'une « bonne » architecture MVC demeure une tâche complexe. C'est pourquoi, de nombreux cadres d'application ont été proposés pour faciliter le développement d'applications Web Java respectant le modèle MVC. Par conséquence, il est impératif de développer votre application en vous appuyant sur un des cadres vu en cours (Struts, Spring ou JSF). Il faut également justifier votre choix quant au cadre d'application choisi et si nécessaire rendre compte des limites de son utilisation.

### 3.9) Correspondance Objet Relationnel

L'utilisation des objets manipulés par votre application avec une base de donnée relationnelle (cf. point 3.3) nécessite de convertir ces objets en données relationnelles et vice-versa. Cela implique donc de programmer cette conversion pour chaque objet que vous souhaiteriez rendre persistant et pour chaque donnée vous souhaiteriez manipuler au sein de votre application. Il faut donc avoir à l'esprit ce besoin et de choisir un cadre d'application facilitant la correspondance Objet-Relationnel (e.g., Hibernate).

## 4) Organisation, rendu et soutenance

Le projet doit être conçu et développé en groupe de 4 étudiants (dont la composition doit être envoyée par mail au plus tard le 10/3 à midi, sous peine d'un point de pénalité par jour de retard). Néanmoins, la notation pourra être individualisée. Le projet doit être réalisé avec l'aide d'un outil de gestion de version (SVN, GitHub, etc.).

Vous devez déposer un document de conception intermédiaire sur le cours en ligne « TIA L3 » (espace « Travaux ») au plus tard le 30 mars 2014 à 23h59 (sous peine d'un point de pénalité par jour de retard). Ce document synthétique (quelques pages : allez à l'essentiel), au format PDF, contiendra :

- Une description générale de l'application (thème de la communauté virtuelle, fonctionnalités envisagées, etc.) ;
- Une première version (simplifiée) des diagrammes de modélisation UML, du plan du site, du/des template(s) de présentation, et autres schémas entité-association ;
- Une indication de l'utilisation prévue des différentes technologies (de **chacune d'entre elles**) ;
- Le rôle de chaque membre de l'équipe ;
- La désignation d'un chef d'équipe.

Vous devez déposer votre projet final sur le cours en ligne « TIA L3 » (espace « Travaux ») au plus tard le 23 avril 2014 à 23h59 (sous peine d'un point de pénalité par jour de retard), sous la forme de 3 fichiers :

- Le rapport au format PDF.

- L'application complète sous la forme de fichiers *.jar* (client) et *.war* (serveur), incluant le code source complet, un fichier de sauvegarde des ressources et des données, et les éventuelles ressources graphiques utilisées.
- La documentation *Javadoc* sous la forme de fichiers HTML compressés dans un fichier zip.

Le rapport doit mettre en valeur votre travail. Il contiendra notamment :

- Une introduction qui décrit succinctement le but du projet et le plan du rapport ;
- Un rapport de conception du projet comprenant diagrammes de modélisation UML, plan du site, template(s), et autres schémas entité-association ;
- Une description des fonctionnalités de l'application ;
- Les technologies de TIA mises en œuvre (**important !** Il s'agit de bien mettre en avant la manière dont elles sont exploitées dans votre application) ;
- Une description précise de l'origine de chaque partie du code : code 100% personnel, librairies de la plateforme Java, librairies trouvées sur le Web, fragments de code trouvés sur Internet ou réalisés en collaboration avec un autre groupe, etc. ;
- Une estimation du temps consacré au projet, ainsi que de l'organisation de votre travail (répartition des tâches au sein de l'équipe, etc.) ;
- Une liste complète des ressources utilisées (outils, livres, tutoriels, sites Web),
- Une conclusion qui pourra comporter un bilan critique de votre travail (résultats, points forts / faibles, etc.), ainsi que vos impressions sur les aspects de TIA qui vous ont particulièrement intéressé (ou non), que vous avez trouvé plus difficile, etc.

Votre code doit :

- utiliser à bon escient les technologies vues en TIA ;
- être compact, lisible, commenté ;
- avoir été soigneusement testé ;
- et enfin, pouvoir être compilé en ligne de commande sur l'environnement utilisé en TP.

Chaque équipe doit réaliser son propre projet. Vous pouvez bien évidemment utiliser des librairies externes (vous le devez !), mais il est impératif de préciser soigneusement les parties de code que vous avez développé entièrement et celles que vous réutilisez.

La présentation finale du projet se fera le 28/4 matin (à partir de 8h30) en mode soutenance.