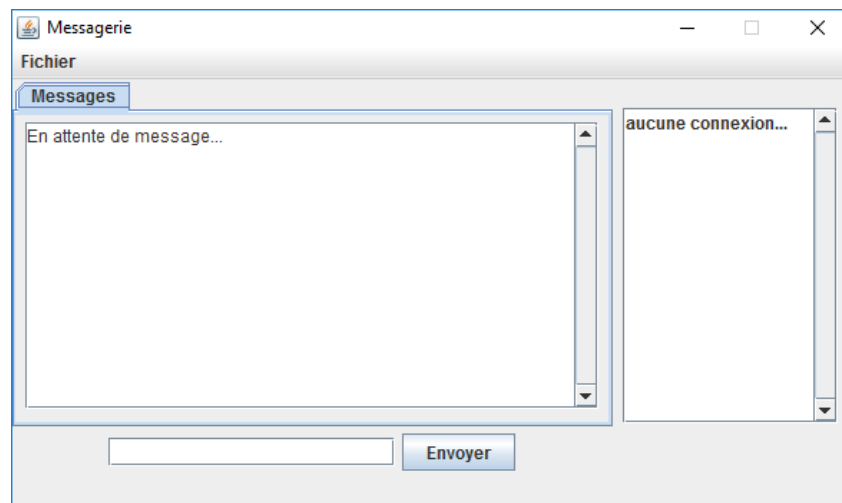


## **RAPPORT DE PROJET TUTEURÉ – SEMESTRE 3**

### **PROJET TUTEURÉ : MESSAGERIE INSTANTANÉE SÉCURISÉE PAR SSL**



#### **RÉALISÉ PAR**

**ERVIN BRETON - THIBAUD MASSOT  
ALEXIS MINOTTO - MATTHIEU POIROT  
MATHIEU SIMONIN**

#### **SOUS LA DIRECTION DE**

**NADJIB LAZAAR**







UNIVERSITÉ  
DE MONTPELLIER

## REMERCIEMENTS

---



Nous tenons à remercier M. Nadjib LAZAAR de nous avoir guidés tout au long de ce projet. Merci aussi au personnel du pôle informatique de l'IUT d'avoir mis à notre disposition un matériel performant afin de mener à bien notre projet.

## INTRODUCTION

<b>I – Analyse.....</b>	<b>6</b>
1.1 Analyse globale .....	6
1.2 Besoins fonctionnels .....	11
1.3 Besoins non fonctionnels .....	15
1.3.1 Environnement technique .....	15
1.3.3 Environnement utilisateurs .....	17
<b>II – RAPPORT TECHNIQUES .....</b>	<b>18</b>
2.1 Conception .....	18
2.1.1 Choix des technologies .....	18
2.1.2 Bibliothèque Socket .....	18
2.1.3 Bibliothèque Swing .....	18
2.1.4 Technologie SSL et bibliothèque sockets .....	19
2.1.5 Conception de chaque version .....	19
2.2 Architecture .....	19
2.3 Perspectives de développement .....	20
<b>III – MANUEL D’UTILISATION .....</b>	<b>21</b>
3.1 Génération des certificats SSL .....	21
3.2 Lancement de l’application .....	21
3.3 Connexion au serveur .....	21
3.4 Connexion à un utilisateur connecté .....	21
3.5 Envoi de message .....	22
<b>IV – RAPPORT D’ACTIVITÉ .....</b>	<b>23</b>
4.1 Méthodes de développement .....	23
4.2 Planification des outils de travail .....	24
4.2.1 Planification des tâches .....	24
4.2.2 Outils d’aide à la répartition et au suivi des tâches .....	26
4.3 Rétrospective générale .....	29
4.3.1 Les valeurs agiles .....	29
4.3.2 Les principes agiles .....	30

## CONCLUSION





Figure 1 : Diagramme de cas d'utilisation initial.....	11
Figure 2 : Diagramme de classe initial.....	12
Figure 3 : Backlog de projet initial.....	12
Figure 4 : Diagramme de cas d'utilisation final.....	13
Figure 5 : Diagramme de classe final.....	14
Figure 6 : Backlog de projet final.....	14
Figure 7 – Diagramme de séquence.....	19
Figure 8 : Burnup chart du produit.....	24
Figure 9 : Graphique de vélocité.....	25
Figure 10 : Backlog général du projet.....	26
Figure 11 : Interface Trello - Second sprint.....	27
Figure 12 : Interface Trello d'une user story.....	28

**Product Owner** : Représentant client/utilisateur auprès de l'équipe, communique en externe sur l'avancement et met à jour le Backlog, priorise le Backlog (sprint).

**Scrum Master** : responsable de l'application quotidienne des directives Scrum. C'est lui qui est en charge d'assurer un environnement de travail agréable pour l'ensemble des membres de l'équipe

**Backlog/ Backlog de sprint** : Recensement des tâches et de leur état au cours d'un sprint.

**Burn up chart** : Graphique de suivi de l'avancement du projet par rapport à la date de rendu.

**Graphique de vélocité** : Graphique de suivi de l'effort restant dans le sprint.

**User story** : L'intégralité du travail à réaliser est découpée en incréments fonctionnels et les activités de développement s'organisent autour de ces incréments appelés « User Stories » pour insister sur l'aspect narratif et l'adoption du point de vue de l'utilisateur.

**Sprint** : Période de plusieurs jours/semaines durant laquelle un groupe de personnes effectue un travail d'équipe sur un projet.

**Chiffrement bout en bout** : Pas d'intermédiaires. Vos messages sont protégés avec un cadenas, et seuls le destinataire et vous avez la clé spéciale qui permet de débloquer et lire votre message. Afin d'assurer une protection supplémentaire, chaque message que vous envoyez à son propre cadenas unique et sa clé unique.



## I – Analyse

### 1.1 Analyse globale

Les messageries instantanées chiffrées ont le vent en poupe, poussant progressivement les utilisateurs et les sociétés à investir dans un chiffrement accessible et simple. Nous nous sommes lancés sur un projet de messagerie instantanée sécurisée par SSL et de ce fait, une analyse des produits existants sur le marché est nécessaire.

Nous avons donc décidé de nous pencher sur une rapide étude de cinq entreprises déjà implantées sur le marché :

- Blackberry BBM
- WhatsApp
- iMessage
- Google Hangouts
- Telegram

### **Blackberry BBM ou le chiffrement professionnel**



La sécurité est un des points essentiels du système BlackBerry. En plus de posséder les qualités d'une messagerie instantanée mobile (emojis, appels ...), l'éditeur canadien met en effet l'accent sur la confidentialité client, notamment pour les professionnels, ce qui implique une messagerie instantanée BBM chiffrée. Les messages sont donc chiffrés pendant leur transfert et passent par les serveurs BlackBerry afin de prévenir toutes failles.

De plus, BlackBerry possède une clé maîtresse pour déchiffrer tous les messages de particuliers (pour une mise à disposition pour les autorités par exemple). Seuls les professionnels peuvent protéger leurs messages avec une clé qu'ils sont seuls à détenir.

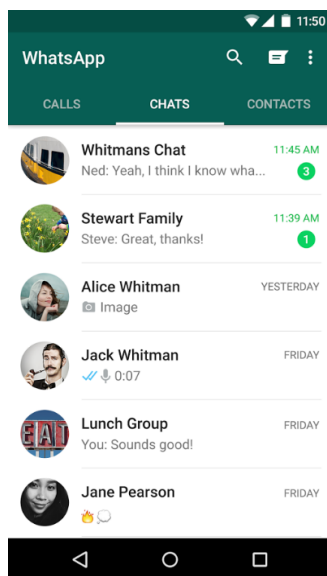


BBM est une solution propriétaire dont le code source n'est pas mis à disposition de l'audimat public.

### WhatsApp, Facebook et confidentialité



WhatsApp, racheté par Facebook, est aujourd'hui une messagerie instantanée de tout premier plan, avec quelques 350 millions d'utilisateur mensuel dans le monde. Menacé par la fuite de ses utilisateurs pour des applications plus sécurisées (Telegram) ou plus tendances (Snapchat), l'entreprise a donc choisi de mettre en place au début Avril 2016 un chiffrement de bout en bout par défaut sur leur application.



Cette solution permettra ainsi de montrer à l'internaute que sa sécurité reste une préoccupation pour la société et qu'il peut faire confiance à l'outil puisque le chiffrement de bout en bout assure que l'application n'a pas accès au contenu des messages, ni autre personne que le destinataire, sauf lors d'une mise sur écoute par exemple. Les messages sont en effet chiffrés de leur point d'expédition jusqu'à leur réception.

Mais attention, les messages certes sont chiffrés, mais les métas données de la messagerie sont conservées ! C'est-à-dire qu'il est par exemple possible pour Facebook de tout savoir sur nos habitudes de discussions et de transmettre ces informations sur demande. WhatsApp repose donc sur la confiance qu'ont les utilisateurs en Facebook en ce qui concerne la protection des données.



## iMessage ou Apple de bout en bout



iMessage est l'application de messagerie par défaut installée par Apple sur ces systèmes IOS et OS X.



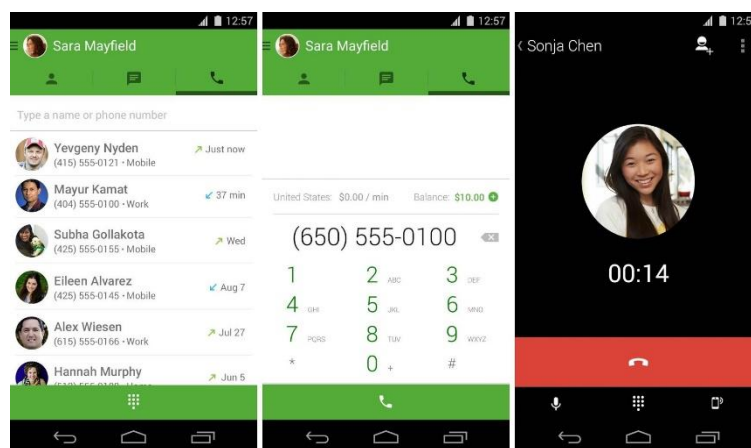
Cette application est l'une des toutes premières à avoir été chiffrée de bout en bout, tout comme WhatsApp. Elle propose donc une solution dans laquelle seul le destinataire et l'expéditeur ont les clés des messages échangés. Le seul souci est que cette messagerie est exclusive à l'écosystème d'Apple, ce qui oblige les deux utilisateurs à posséder un appareil de la firme.

En creusant un peu plus sur la sécurité de l'application, on se rend compte que lorsque l'utilisateur réalise une sauvegarde d'historique de messagerie dans iCloud (service Cloud de l'écosystème Apple), celui-ci verra sa clé récupérée par l'entreprise. Il faudra donc ici aussi avoir confiance en l'entreprise Apple ...



## Google Hangouts, le mauvais élève

Google Hangouts est la nouvelle application de messagerie par défaut installée par Google sur la plupart des systèmes Android et sur de nombreuses autres plateformes. Celle-ci vient remplacer les anciens Google Talk, Google+ et offre un chiffrement de message par défaut. Le chiffrement est donc effectué seulement pendant le transport, il n'y a pas de système de clé par utilisateur. Google conserve donc l'accès en clair aux messages échangés !



Certes accessible, il sera cependant préférable pour un professionnel averti cherchant une grande confidentialité de ses données d'utiliser un autre service comme Telegram que nous allons étudier pour clore cette étude.

## Telegram, messagerie star

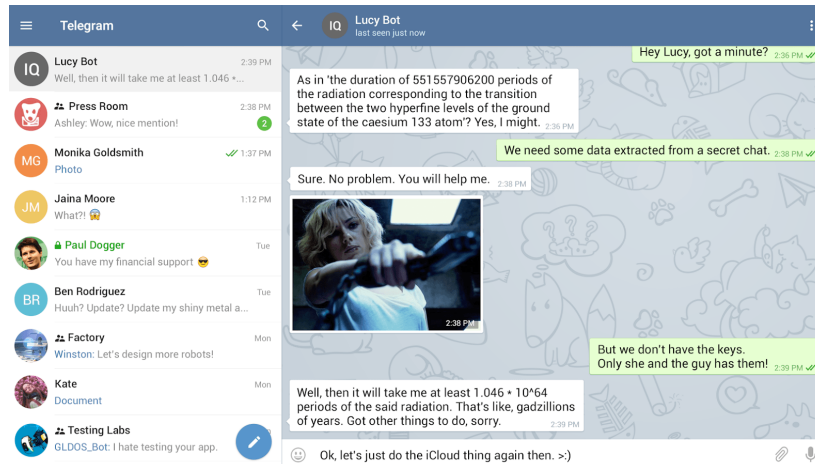
Telegram est une application de messagerie créée par deux frères Russes qui a fait parler d'elle notamment lors des attentats de Paris à cause de sa soi-disant notoriété auprès des djihadistes. Loin du terrorisme, cette application est une réplique des deux frères aux différents scandales sur la surveillance des données.



Avec plus de 100 millions d'utilisateurs mensuels, cette application propose des échanges sécurisés entre utilisateurs via une solution bout en bout totale. En effet, l'utilisateur doit passer par le mode « Secret Chat » pour pouvoir bénéficier de ce service, les messages

n'étant pas chiffrés par défaut. De plus, cette application est une des premières à gérer les chatbots grâce à ses API et à son code source mis à disposition sur Github de sa grande communauté d'utilisateurs.

Telegram propose aussi de supprimer les messages du terminal de notre destinataire ou de planifier leur disparition !



### Pour conclure ...

Malgré l'avantage d'être ergonomiques et très répandues chez les utilisateurs, les messageries citées comme WhatsApp ou encore iMessage ne peuvent pas convenir à tout le monde, notamment pour les plus soucieux de la confidentialité de leurs échanges. La solution est donc de se tourner vers des solutions libres et open source dont le code est soumis à l'audition et peut être exécuté sans conservation des métadonnées. Nous avons donc pour objectif dans notre projet de se rapprocher au plus du mode de fonctionnement d'une de ces messageries.

## 1.2 Besoins fonctionnels

Suite à notre analyse de l'existant nous avons réussi à établir l'architecture de notre projet et ainsi nous avons pu en déduire les diagrammes initiaux suivants :

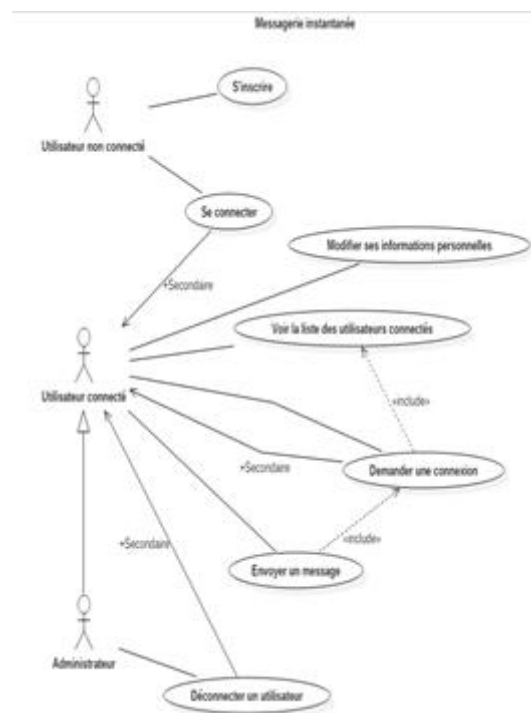


Figure 1 : Diagramme de cas d'utilisation initial

Un utilisateur non connecté peut choisir de s'inscrire ou se connecter. Une fois connecté il peut commencer par accéder à la modification de ses informations personnelles. Il peut ensuite consulter la liste des utilisateurs connectés et ainsi demander une connexion avec un autre utilisateur afin de pouvoir s'envoyer des messages.

Un acteur administrateur a aussi été défini pour permettre la déconnexion manuelle d'un utilisateur (si celui-ci perturbe ou en cas d'infiltration par exemple).

Ce premier diagramme de cas d'utilisation nous donne donc le diagramme de classe suivant :

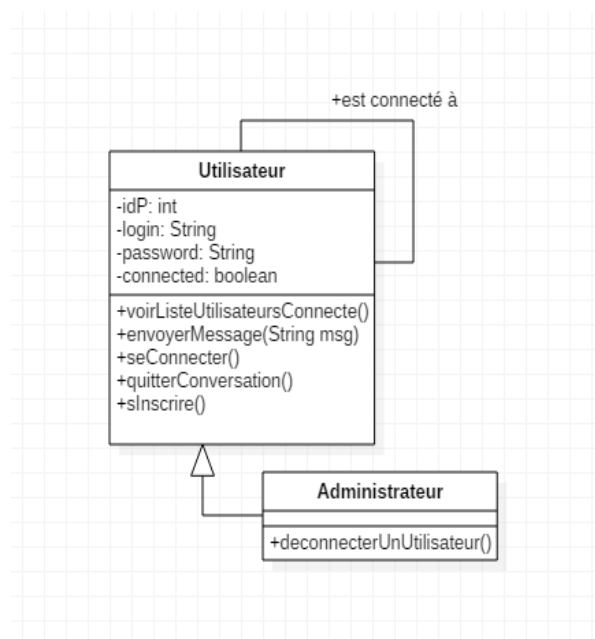


Figure 2 : Diagramme de classe initial

Ce diagramme succinct nous illustre ici l'interaction client/client ainsi que les différentes fonctions mises en jeu dans cette interaction. Ainsi apparaît encore dans ce diagramme les fonctions de connexion, d'inscription ou encore d'envoi de messages.

Les diagrammes générés ainsi que l'utilisation des méthodes agiles nous ont permis d'établir un premier Backlog de projet suivant :

	Gestion des comptes	Connexion	Interaction
Utilisateur non-connecté	Créer un compte	Se connecter	
Utilisateur connecté	Modifier ses informations personnelles		Consulter la liste des utilisateurs connectés
			Demander une connexion
			Envoyer un message
Administrateur	Modifier ses informations personnelles	Déconnecter un utilisateur	Consulter la liste des utilisateurs connectés
			Demander une connexion
			Envoyer un message

Figure 3 : Backlog de projet initial

Au cours de l'avancement du projet, nous avons été en mesure de récolter de nombreuses informations et spécifications auprès de notre tuteur. De plus nous avons aussi rencontrés des difficultés techniques qui nous ont poussé à redéfinir certains aspects du projet

pour être le plus possible en adéquation avec la vision du tuteur, sans pour autant viser l'impossible.

Nous avons donc développés l'aspect de la communication avec un serveur, ainsi que décidé, pour ce projet comportant un délai relativement court, de ne pas mettre en place prématurément le système de comptes utilisateurs. L'administrateur a quant à lui été mis de côté, son action influant trop peu le projet et n'étant pas intéressante pour l'aspect principal du projet.

Ces réflexions nous ont donc amenés à proposer une version finale pour le temps imparti du projet basée sur les diagrammes suivants :

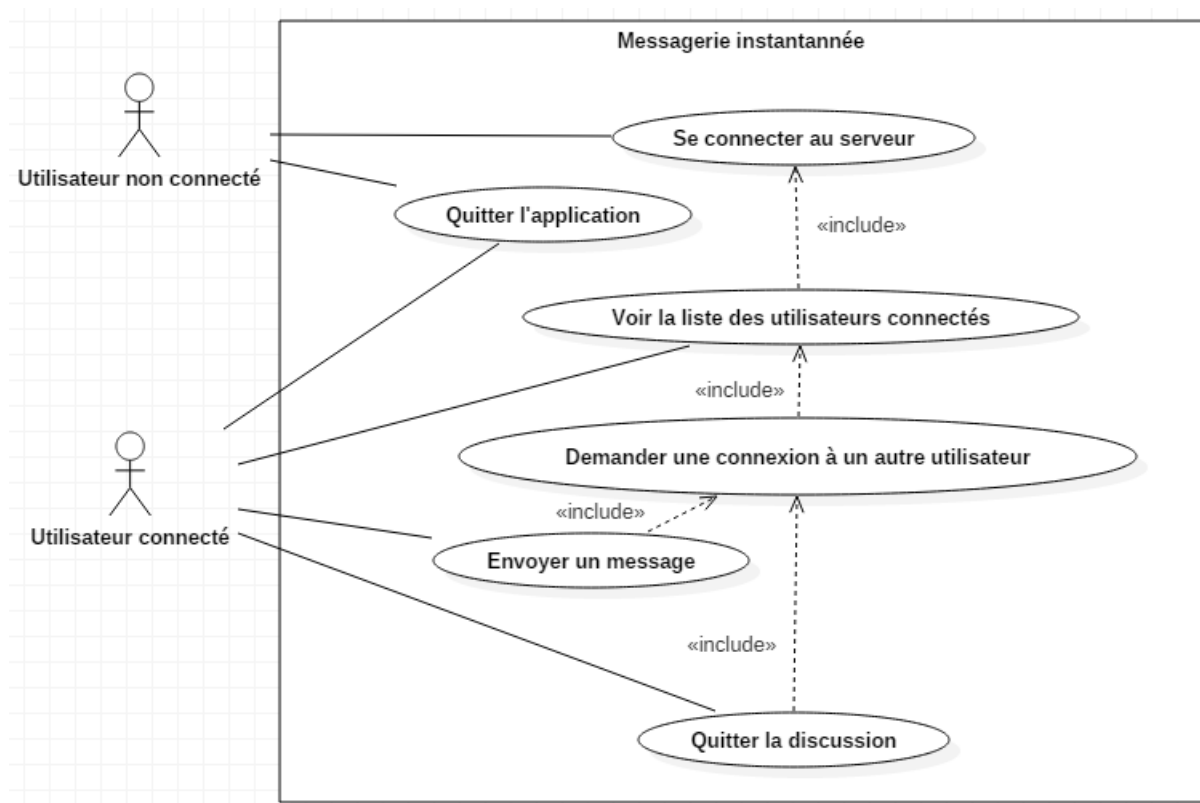


Figure 4 : Diagramme de cas d'utilisation final

Un utilisateur non connecté a le choix entre quitter l'application et se connecter au serveur. Une fois connecté, l'utilisateur possède à sa disposition la liste des autres utilisateurs connectés au serveur. Il peut ainsi choisir de demander à se connecter avec un autre utilisateur connecté afin de lui envoyer un message. Il peut aussi quitter la discussion ou tout simplement quitter l'application.



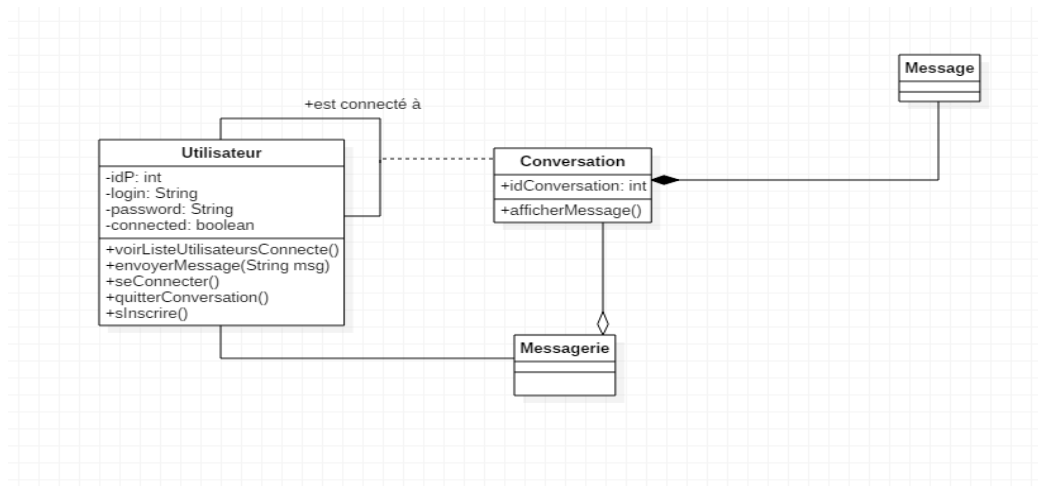


Figure 5 : Diagramme de classe final

Ce diagramme de classe un peu plus détaillé nous illustre toujours ici l'interaction client/client et les fonctions mises en jeu dans l'interaction. On remarque ici l'idée des procédés de communication qui s'aiguise avec l'apparition des classes conversation et message qui composent la messagerie. Le serveur n'intervient pas dans ce diagramme car il s'agit tout simplement d'une application à part entière.

Nous pouvons, à la suite de ces réflexions, établir le Backlog final suivant :

	Gestion des connexions	Gestion des interactions
Utilisateur non-connecté	Se connecter au serveur	Quitter l'application
Utilisateur connecté	Se connecter avec un autre utilisateur	Consulter la liste des utilisateurs connectés
	Ouvrir une nouvelle conversation	Sélectionner un onglet de conversation
		Envoyer un message
		Quitter l'application
Serveur		Sécuriser les interactions par SSL

Figure 6 : Backlog de projet final

Une possibilité de développement dans une version future du projet serait donc de créer la notion d'administrateur, ainsi que d'ajouter un système de gestion de comptes.

## 1.3 Besoins non fonctionnels

### 1.3.1 Environnement technique

La messagerie proposée dans ce projet est destinée à être utilisée par une entreprise, sans être accessible en dehors de celle-ci pour plus de protection.

La connexion des clients au serveur se fera donc en local.

### 1.3.2 Spécifications techniques et contraintes

#### **Langage**

Nous utiliserons Java comme langage de programmation (langage étudié au préalable).

Les bibliothèques spécifiques utilisées:

- java.security fournit les classes et interfaces pour le cadre de la sécurité.
- java.net fournit les classes pour l'implémentation des applications de réseaux, nous avons surtout utilisé la partie Socket de cette bibliothèque.
- javax.swing fournit les composants pour la création d'une interface graphique pour l'utilisateur

#### **Sécurité**

La Messagerie sera sécurisée par SSL Socket Secure Layer.

#### **Performance**

L'exécution du programme est instantanée car c'est un petit programme.

Une fois la connexion établie entre deux clients, l'échange de message est instantané. Il n'y a pas de temps d'attente.

#### **Capacité**

La capacité maximum de connexion au serveur sera limitée par le nombre de port et par la machine qui héberge le serveur. Cela ne devrait pas poser de problème car une fois la connexion établie avec l'autre client, il n'y a plus besoin d'avoir une interaction avec le serveur.

Le serveur ne risque pas un déni de système car le nombre de connexions simultanées ne sera pas important et de plus il sera contrôlé.

## **Disponibilité**

La messagerie, devra être disponible tous les jours 24h/24.

Il peut arriver que ce service soit indisponible suite à des problèmes techniques comme par exemple un problème de serveur.

Ce service doit être disponible depuis tous les ordinateurs connectés au réseau.

## **Fiabilité**

L'application est censée rester fiable étant donné qu'elle est utilisée dans un cadre local d'une entreprise et est donc contrôlée (pas de surcharge des ports etc.).

En cas de problème technique (bug de l'application, panne de courant etc.) un technicien devra relancer l'application du côté serveur, l'utilisateur de son côté client aussi. Ce processus de rétablissement du bon fonctionnement dépendra de la taille de l'entreprise utilisant l'application et du problème en question.

Le seuil acceptable de temps d'indisponibilité est relative vis à vis de la dépendance de l'entreprise envers l'application (taille, moyens de communication préexistants etc.).

## **Compatibilité**

L'application devra être exécutable sous Windows, Linux et MacOS, donc sera un .jar.

La version minimum qui sera requise pour Java est 1.7.

## **Aptitude à la maintenance**

Il faudra un fichier de configuration qui permet de stocker l'IP du serveur. La maintenance consistera à modifier ce fichier de configuration sur toutes les machines du serveur.

## **Ergonomie**

Une des contraintes ergonomiques est de disposer d'une interface graphique. L'interface mise en place doit être facile d'utilisation pour l'utilisateur. Cela signifie que l'interface homme-machine doit être cohérente avec les besoins de l'utilisateur.

Un affichage clair. Il faudra que l'utilisateur dispose d'une fenêtre avec un menu ainsi qu'un affichage de la liste des utilisateurs.

Pour ne pas compliquer l'interface, nous utiliserons au maximum 5 couleurs.

Il y devrait y avoir un bouton pour se connecter et se déconnecter disponible depuis le menu.

Les onglets correspondant à la conversation devront avoir le nom du login de l'utilisateur avec qui on converse pour savoir où se trouve tel conversation.

L'utilisateur devra disposer d'un moyen déclencheur pour confirmer l'envoi du message comme par exemple un bouton "Envoyer" sur lequel il devra cliquer pour envoyer son message ou bien la possibilité d'envoyer le message avec la touche "Entrée" car c'est aujourd'hui la touche la plus instinctive pour l'envoi des messages sur une messagerie instantanée.

La langue proposée pour le logiciel sera le français.

## **Documentation**

L'utilisateur disposera d'un manuel d'installation ainsi que d'un manuel d'utilisation afin de prendre en main au mieux l'application. Comme lors de la mise en place de nouveaux SI - Systèmes d'Informations -, une formation à la prise en main du logiciel sera proposée afin d'éviter le rejet des employés vis à vis de cet outil.

## **Juridique**

Pas de contrainte juridique relevée

### **1.3.3 Environnement utilisateurs**

L'accès à la messagerie se fera via les machines de l'entreprise.

Afin que l'utilisateur puisse échanger avec les autres, il faut avoir fait au préalable une installation des certificats sur la machine qu'utilise l'utilisateur. Une machine nécessite une installation.

## II – RAPPORT TECHNIQUES

### 2.1 Conception

La phase de conception du projet s'est effectuée selon le schéma suivant :

1. Choix des technologies
2. Apprentissage de la bibliothèque Socket
3. Apprentissage de la bibliothèque Swing
4. Apprentissage de la technologie SSL et de la bibliothèque Java SSL Sockets ainsi que la génération des certificats

#### 2.1.1 Choix des technologies

Nous avons tout d'abord choisi de réaliser ce projet grâce au langage Java pour la bonne raison que nous l'avons déjà tous étudié préalablement. Ce projet étant orienté réseau nous avons utilisé l'API Socket de Java afin d'établir des connexions entre les machines de façon relativement simple. Enfin, une interface graphique étant nécessaire à nos yeux, notre choix s'est tourné vers la bibliothèque Swing pour sa documentation abondante, qui s'est révélée être plus difficile à utiliser que prévu.

#### 2.1.2 Bibliothèque Socket

Les sockets permettent d'implémenter le protocole TCP au sein de notre application. Elles permettent d'obtenir un flux d'entrée et de sortie entre une entité appelée client et une entité appelée serveur. Le serveur attend une connexion, et le client effectue cette connexion. A la suite de cela on peut utiliser les flux pour recevoir ou envoyer des données entre les deux entités.

Du point de vue de notre projet :

Un utilisateur qui demande une connexion avec un autre utilisateur va être considéré comme un client. En effet, les utilisateurs sont de bases des serveurs et attendent des connexions en permanence.

#### 2.1.3 Bibliothèque Swing

La bibliothèque Swing en Java propose de nombreux composants permettant de créer des fenêtres et de gérer des événements, comme un clic ou la pression d'une touche.

Ainsi les méthodes de notre projet récupèrent les « return » des éléments de Swing pour effectuer ce que l'utilisateur veut faire. L'apprentissage de cette bibliothèque s'est fait de zéro pour nous.

### 2.1.4 Technologie SSL et bibliothèque sockets

La technologie SSL permet de sécuriser la communication entre deux hôtes :

- Chiffrement des messages
- Intégrité des messages
- Authentification des hôtes

Ici nous avons utilisé la classe SSLSocket pour sécuriser ce que nous faisons déjà avec les Sockets.

L'apprentissage de la technologie SSL nous a pris beaucoup de temps, nous avons eu du mal à comprendre son utilisation dans le code avec les SSLSockets.

### 2.1.5 Conception de chaque version

Nous avons décidé grâce aux méthodes agiles de mettre en place un système d'itération. En effet nous avons développé quatre applications indépendantes.

Vous pourrez trouver en annexe un récapitulatif de ces différentes versions. (ALEXIS DOIT T'ENVOYER CA NORMALEMENT)

## 2.2 Architecture

Nous avons choisi d'adopter l'architecture MVC, nous avons trouvé qu'elle se prêtait bien à notre projet. Ainsi la bibliothèque Swing est utilisée seulement par la partie View et Controller, et l'utilisation des Sockets et la communication entre le serveur et les utilisateurs se fait dans le Model.

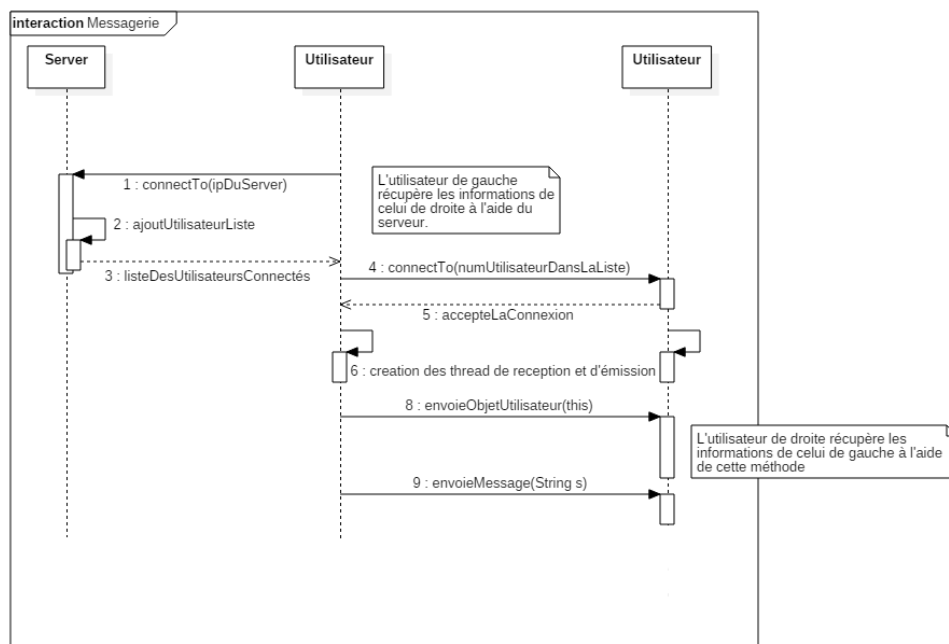


Figure 7 – Diagramme de séquence

Ce diagramme de séquence permet d'expliquer approximativement le code de notre application.

Le projet est donc composé de deux programmes, un serveur à lancer et à laisser actif, et une messagerie, qui constitue la partie client. Chaque client lance une instance de messagerie sur son ordinateur pour ensuite se connecter au serveur. (cf. Annexe 8 – Diagramme de classe)

Une petite explication du code se trouve aussi dans les annexes. (cf. Annexe 9 – Explication des algorithmes)

## **2.3 Perspectives de développement**

L'application est développée de façon à faciliter la programmation future. Par exemple la classe utilisateur n'admet pour l'instant qu'un nom et une ip, mais elle peut contenir d'autres choses comme une image de profil par exemple. Il faudrait juste ensuite développer l'affichage, dans la partie View.

Il en va de même pour les messages, on pourrait inclure un nouvel attribut, comme une image. Il est aussi possible de parcourir le texte envoyé pour remplacer par exemple « :joyeux: » par un émoticône.

On se retrouve donc avec une application de messagerie qui peut être développée afin de contenir toutes les fonctionnalités des messageries actuelles, comme Facebook

## III – MANUEL D’UTILISATION

S’il souhaite installer la messagerie, le client devra tout d’abord télécharger l’archive « jar.zip » à l’adresse suivante :

<https://drive.google.com/file/d/0B6KLsTOK1jbuZHBHYmFBLVJXSEO/view>

Suite à ça, il faudra décompresser cette archive dans le dossier d’installation. (Par exemple : /opt, Programme File,...) L’utilisateur devra aussi créer des raccourcis / lanceurs de fichiers dans un dossier facile d’accès. (Par exemple : Bureau)

NB : Sous linux la création d’un lanceur de fichiers prendra come paramètre « java -jar PATH\_TO\_JAR\_FILE »

### 3.1 Génération des certificats SSL //////////////////////////////////////

### 3.2 Lancement de l’application

Pour lancer l’application, rien de plus simple. Il suffira de double cliquer sur le raccourci créé au préalable dans la phase d’installation ou encore directement sur le fichier « .jar ».

### 3.3 Connexion au serveur

La connexion au serveur se fait en plusieurs étapes très simples. Une fois l’application lancée (cf. Annexe 1 – Application lancée), l’utilisateur peut sélectionner l’onglet « Connexion » dans le menu. (cf. Annexe 2 – Connexion au serveur) Lorsqu’il a cliqué sur ce bouton pour se connecter, un pop-up apparaît et demande l’adresse IP du serveur (cf. Annexe 3 – Pop-up IP serveur).

L’utilisateur rentre alors l’adresse IP du serveur auquel il souhaite se connecter correspondant au réseau local sur lequel est connectée la machine du client. Enfin, une fenêtre s’ouvre affichant le message « En attente de message, ... » Et la liste des utilisateurs connectés apparaît. (cf. Annexe 4 – Connexion réussie)

### 3.4 Connexion à un utilisateur connecté

Une fois qu’un utilisateur est connecté au serveur et a accès à la liste des autres utilisateurs connectés, il est très simple de se connecter à un autre utilisateur afin de communiquer avec lui. En effet, la fenêtre affiche la liste de tous les utilisateurs connectés. (cf. Annexe 5 – Liste des utilisateurs connectés) Il faut donc simplement double-cliquer sur le pseudonyme de l’utilisateur avec qui on souhaite communiquer et une nouvelle fenêtre avec le message « Connexion réussie ! » apparaît. (cf. Annexe 5 – Connexion utilisateur réussie)



### 3.5 Envoi de message

Pour envoyer un message, rien de plus simple, l'utilisateur écrit dans la zone dédiée puis clique sur le bouton « Envoyer ». Le receveur ainsi que l'expéditeur auront le message affiché dans la fenêtre correspondant à leur conversation. (cf. Annexe 7 – Envoi de message)

## IV – RAPPORT D’ACTIVITÉ

NB : Il est fortement conseillé de consulter le lexique avant d’entamer cette partie qui comporte des termes clés relatifs aux méthodes agiles.

Au cours de la réalisation de ce projet de messagerie sécurisée par SSL, il nous a fallu mettre en place la gestion de notre projet, élément plus que nécessaire et déterminant pour la division du travail et la réalisation des tâches dans les temps impartis. Nous nous pencherons dans cette partie sur cet aspect « gestion » du projet ainsi que sur les différents outils utilisés pour simplifier diverses étapes du projet

Nous allons donc effectuer cette présentation en trois temps. Nous commencerons par une rapide présentation du groupe ainsi que sur la méthode générale de développement de notre projet. Nous nous pencherons ensuite sur la planification du projet ainsi que sur les outils qui nous ont accompagnés tout au long de la réalisation. Nous finirons enfin avec une rétrospective générale au regard des méthodes utilisées et du déroulement du projet au fil du temps.

### 4.1 Méthodes de développement

Pour mener à bien ce projet, nous avons une équipe de 5 personnes composée de :

- Ervin BRETON
- Thibaud MASSOT
- Alexis MINOTTO
- Matthieu POIROT
- Mathieu SIMONIN

Dans un premier temps, un travail de conception a été mis en place durant avant toute réalisation afin de nous permettre de jauger le projet et d’établir un premier Backlog général de projet. Nous avons ensuite entamé la réalisation du projet sur la base des méthodes agiles, un système de développement de projet itératif et proche du client que nous détaillerons dans la troisième partie.

C’est une collaboration munie d’une communication efficace qui, tout au long du développement du projet, notamment grâce aux réunions avec le tuteur de projet M.Lazaar ainsi que les réunions d’équipe, nous a permis de gérer au mieux l’avancement du projet, le tout appuyé par les outils étudiés en première année que nous aborderons dans la partie suivante.

Les créneaux libres dans l’emploi du temps nous ont permis de travailler en commun sur certaines tâches et de nous fixer sur certaines décisions en plus des tâches individuelles distribuées à chacun

## 4.2 Planification des outils de travail

### 4.2.1 Planification des tâches

Notre planning de projet s'est découpé en quatre sprints basés sur les méthodes agiles. Ces méthodes consistent à incrémenter les tâches du Backlog général au fur et à mesure de l'avancement du projet dans les sprints de celui-ci en fonction de valeur client et de valeur d'effort attribuées à chaque User story.

Analysons maintenant le Burnup chart ainsi que le graphique de vélocité relatif au projet :

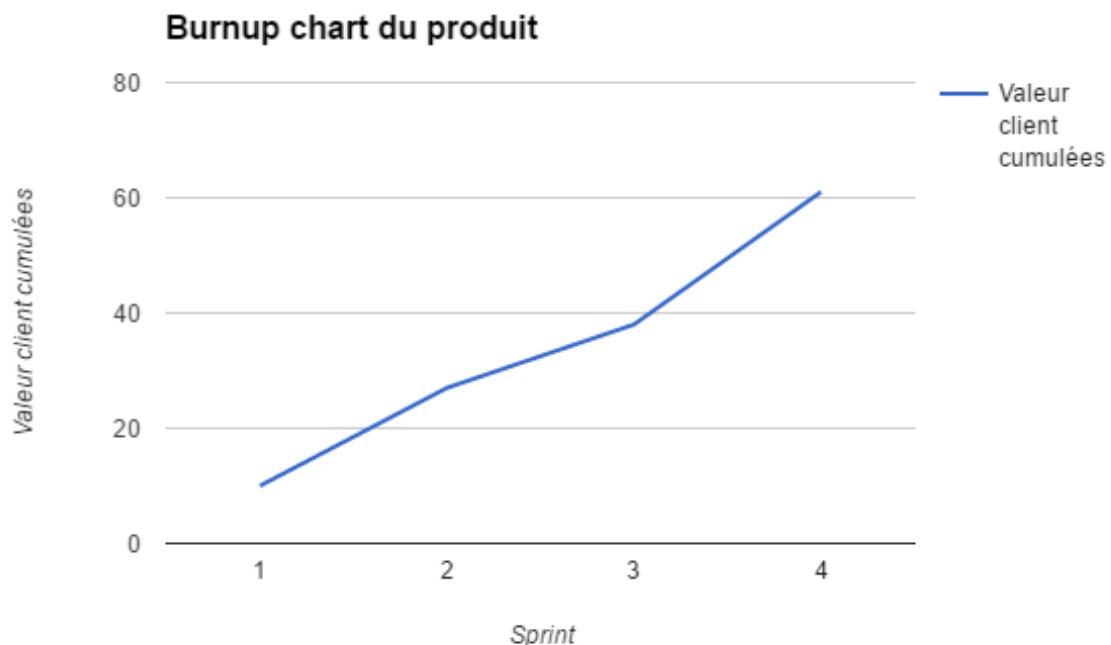


Figure 8 : Burnup chart du produit

On étudie ici l'avancement du projet en termes de « valeur client », représenté par des points d'importance attribués à chaque User story en fonction de sa valeur aux yeux du client, ici représenté par notre tuteur (noté sur une échelle de 1 à 9). Nous parlerons toujours dans les parties suivantes de client, symbolisé par le tuteur.

On peut voir qu'au début du projet lors du rendu du premier sprint, les stories implémentées n'ont pas de réelle importance pour le client car il s'agit principalement d'installations techniques ou de compréhension d'outils nécessaires au développement du projet. Une fois que les composants nécessaires sont pris en main, on commence alors à développer les fonctionnalités les plus importantes dégagées du client par le Product Owner.

La rapidité et le temps impartis de ce projet fait qu'il a été développé en seulement quatre sprints. Sur un projet plus important, ou si on décidait de continuer le projet, on verrait alors la courbe s'aplatir au fur et à mesure de la finalisation du projet car les nouvelles fonctionnalités développées auraient de moins en moins d'importance aux yeux du client puisque les plus importantes ont été développées au début du projet.

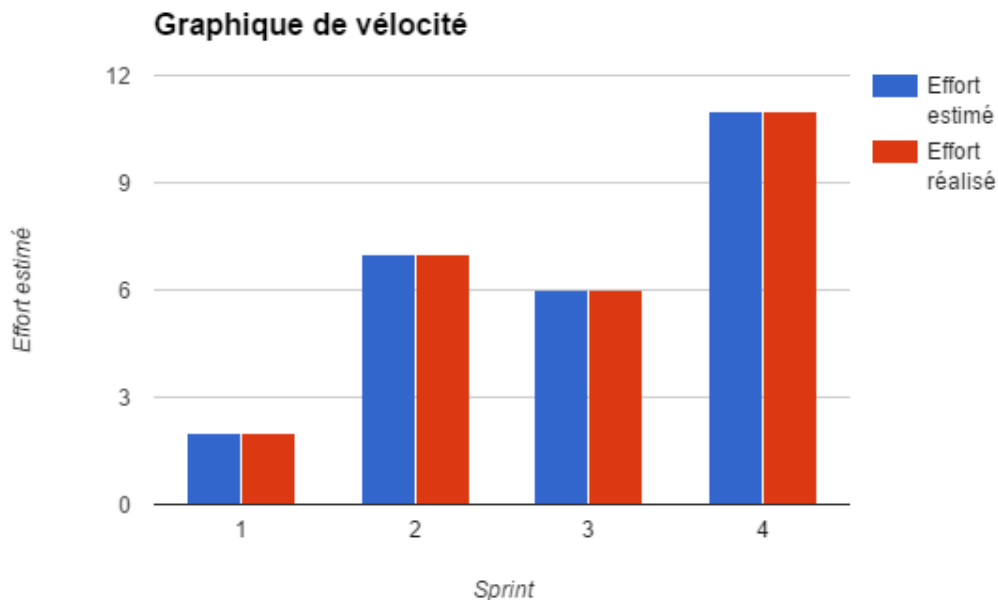


Figure 9 : Graphique de vélocité

On étudie ici l'avancement du projet en termes de « valeur d'effort » représenté par des points d'effort attribués à chaque User story en fonction de sa difficulté à implémenter aux yeux de l'équipe de développement (noté sur une échelle de 1 à 5). On compare donc ici l'effort réalisé à ce qui était prévu à chacun des quatre sprints.

On remarque ainsi que nous avons réussi à prévoir et nous tenir à la réalisation de nos User stories à chacun de nos sprints avec une légère baisse de rendement lors du sprint 3 où nous nous sommes concentrés sur l'amélioration de notre second sprint (multi conversations avec système d'onglets). La quantité d'effort demandé pour chaque sprint démarre assez bas due notamment à une adaptation au projet et aux méthodes utilisées ainsi qu'à l'apprentissage de certains outils pour développer. Une fois cette période d'adaptation passée, on arrive aisément à prévoir une plus grande quantité de tâches réalisables et ainsi à remplir les objectifs fixés à chaque itération. La quantité de User stories implémentées est donc croissante avec un pic au dernier sprint car celui-ci fait intervenir un aspect principal du projet et que nous ne pouvions pas développer avant : la sécurisation SSL en lien avec le serveur.

Toutes les valeurs (effort et client) ont été discutées et attribuées après accord commun des différents membres du groupe lors des différentes réunions de groupe. Nous en discuterons plus en détail dans la partie suivante.

## 4.2.2 Outils d'aide à la répartition et au suivi des tâches

Pour assurer une bonne communication entre les membres du groupe, nous avons utilisé principalement Facebook et Slack avec notamment l'outil de création groupe privé nous permettant d'avoir à notre disposition une messagerie instantanée avec tous les membres du groupe ainsi qu'une plateforme sur laquelle nous pouvions échanger nos ressources. De plus nous avons décidé d'utiliser Github afin de gérer le versionning de notre projet et de simplifier le développement à plusieurs. C'est par ces moyens que nous avons pu éviter des divergences au cours du développement.

Google Drive a aussi été utilisé et nous a permis un partage simple des documents en interne ainsi qu'avec notre tuteur. Cela a été également utile pour la rédaction collaborative et en temps réel du rapport de projet. De plus, Google Sheets, l'Excel de Google Drive a aussi été adopté pour la création et la mise à jour du backlog général du projet et des différentes courbes relatives aux comptes rendus des différents sprints.

	Gestion des connexions	Gestion des interactions
Utilisateur non-connecté	Se connecter au serveur	Quitter l'application
Utilisateur connecté	Se connecter avec un autre utilisateur	Consulter la liste des utilisateurs connectés
	Ouvrir une nouvelle conversation	Sélectionner un onglet de conversation
		Envoyer un message
		Quitter l'application
Serveur		Sécuriser les interactions par SSL

Figure 10 : Backlog général du projet

Tout au long de notre projet nous avons dû établir une liste des différentes User stories à implémenter pour chaque itération. Nous avons donc utilisé l'outil Trello qui nous a permis d'avoir l'équivalent d'un tableau d'avancement de projet en ligne, organisé en 5 catégories de colonnes, « To Do », « Doing », « Done », « OK » et « KO ». Nous expliquerons les éléments relatifs au Trello un peu plus tard.

C'est grâce à cette séparation en 5 colonnes que nous avons évalué en groupe l'effort des différentes User stories à implémenter via l'attribution à chaque colonne d'un chiffre (1 à 5), permettant ainsi d'établir la valeur des différentes User stories définies en les comparant entre elles, en les faisant naviguer comme des post-it entre les colonnes avant de les remettre à leur place dans la colonne « To Do ».

L'organisation via le Trello est donc un élément majeur relatif à la réalisation du projet puisqu'il nous a permis de nous coordonner tout d'abord dans la création, puis dans l'exécution et enfin la répartition des tâches ainsi que nous permettre d'avoir une vision d'ensemble de l'avancement du projet une fois le sprint validé et corrélé au backlog général du Google Drive.

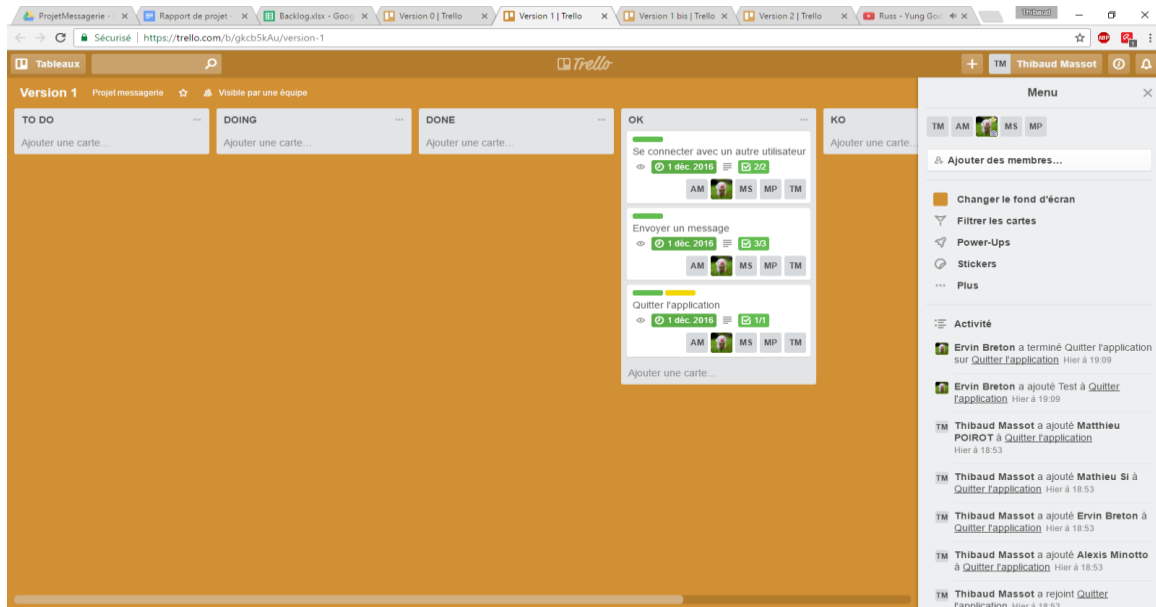


Figure 11 : Interface Trello - Second sprint

Tout d'abord la barre en haut nous sert à ajouter de nouveaux tableaux de sprints et de nouveaux membres ayant accès au tableau. Le menu à droite nous donne une vue sur l'historique des dernières modifications ainsi que des options relatives au tableau.

Au centre nous retrouvons nos 5 colonnes :

- « To Do » pour les User stories qui n'ont pas encore été commencées
- « Doing » pour celles qui sont en cours
- « Done » pour celles terminées
- « OK » pour celles validées lors du compte rendu de sprint avec le client
- « KO » pour celles non validées lors du compte rendu de sprint avec le client

Les colonnes contiennent des cartes représentant chacune une User story. Une petite fenêtre s'ouvre lorsque l'on clique sur une carte, nous permettant de préciser notre User story avec une date limite, un code couleur définissable par des étiquettes, la possibilité de définir des formulaires pour valider des tests utilisateurs par exemple et bien plus encore ....

Cela nous permet à notre retour sur l'interface principale d'avoir une organisation claire et précise de l'avancement du projet à un instant T, avec le nombre total de User stories, leur position, les personnes travaillant dessus et bien plus encore car chaque carte est consultable ...

The screenshot shows a Trello card titled "Envoyer un message" with a close button (X) in the top right corner. Below the title, it says "Dans la liste OK" with an eye icon. The card is divided into several sections:

- Membres:** A row of member avatars and initials: AM, a profile picture, MS, MP, TM, and a plus sign. To the right, there is a green label "Utilisateur connecté" with a plus sign.
- Étiquettes:** A section for labels, currently empty.
- Ajouter:** A sidebar on the right with buttons for "Membres", "Étiquettes", "Checklist", "Échéance", and "Pièce jointe".
- Échéance:** A green label with a checkmark and the text "1 déc. 2016 à 16:00".
- Description:** A section with the title "Description" and an "Éditer" link. The text reads: "En tant qu'utilisateur connecté, je dois pouvoir envoyer un message à un autre utilisateur connecté". Below this, it shows "Valeur: 7" and "Effort: 2". A note at the bottom says: "Il y a des modifications non sauvegardées dans ce champ. [Afficher les modifications](#) - [Supprimer](#)".
- Actions:** A sidebar on the right with buttons for "Déplacer", "Copier", "S'abonner" (which is checked with a green checkmark), and "Archiver". Below these buttons is a link "Partagez et plus...".
- Checklist:** A section with a title "Envoyer un message" and a progress bar at 100%. Below the progress bar are three items, each with a checkmark: "Écrire un message", "Envoyer un message", and "Recevoir un message". At the bottom of the checklist is a link "Ajouter un élément...".
- Commentaires:** A section titled "Ajouter un commentaire" with a profile picture and a text input field "Écrivez un commentaire...". Below the input field are icons for attachments, mentions, emojis, and a message icon. At the bottom of the comment section is an "Envoyer" button.

Figure 12 : Interface Trello d'une user story

### 4.3 Rétrospective générale

Il est maintenant temps de revenir sur le déroulement du projet au sein même de notre groupe. Ervin était en position de Product Owner et Thibaud de Scrum Master.

Nous avons su conserver tout au long de la réalisation du projet une bonne cohésion et communication entre les membres du groupe, ce qui nous a permis d'avoir une bonne organisation et d'arriver à nos fins en validant notre réalisation finale auprès de notre tuteur.

Notre principal problème a été la gestion du temps dans l'avancement du projet : en effet, ce projet tuteuré est tombé en parallèle à plusieurs autres projets (projet web PHP, projet EXIGE/création d'entreprise, projet ELIGA ...), qui plus est, pendant une période comprenant des partiels, ce qui nous a obligé à partager notre temps entre ces diverses tâches importantes.

Il a donc fallu s'adapter et nous avons réussi à trouver un équilibre qui convenait et s'adaptait à chacun en fonction de ses qualités. Ainsi Alexis, Mathieu et Matthieu étaient davantage sur la partie technique et informatique du projet tandis que Ervin et Thibaud étaient plus basés sur « l'administratif » avec les comptes rendus de sprint, les réunions d'équipe etc...

Nous avons donc travaillé indépendamment de chez nous en nous répartissant le travail et utilisant GitHub et surtout en groupe à l'IUT, pour la réalisation du Trello à chaque sprint et afin de tous nous tenir à jour de l'avancement du projet dans l'administratif et le technique/informatique ou encore expliquer les changements lors du retour d'une personne absente.

Voyons maintenant plus en détail l'utilisation des méthodes agiles au cours du projet.

#### 4.3.1 Les valeurs agiles

Ce projet était mené avec l'utilisation des méthodes agiles, nous avons donc pu en expérimenter les différents principes que nous verrons dans la prochaine partie.

Compte tenu du résultat du projet, nous pouvons dire que l'utilisation des méthodes agiles a bien été prise en main et efficace. Le respect des rôles de Product Owner et de Scrum Master, une communication et une bonne entente omniprésente entre les membres présents tout au long du projet, une bonne relation et adaptabilité au tuteur, notamment pour un point important du résultat attendu : la communication serveur ... Tout ceci nous a permis de bien cerner les attentes de notre projet et de nous adapter pour produire en réponse un résultat au plus proche des attentes de notre tuteur, et nous pensons que la mission est accomplie.



### 4.3.2 Les principes agiles

Énoncé précédemment, voici le détail de l'utilisation des différents principes des méthodes agiles au long du projet.

Concernant tout d'abord le cycle de développement itératif incrémental, la bonne relation établie par notre Product Owner avec le tuteur nous a permis d'arriver à plutôt bien cerner les attentes pour le projet et donc de ne pas trop faire d'erreurs de parcours. Nous avons donc pu très vite recentrer nos User stories d'un sprint à l'autre quand nous nous éloignons un peu du bon chemin. De plus, le fait de présenter directement le projet au tuteur lors de réunion lui permet de se faire une idée plus précise des nouvelles fonctionnalités implémentées et peut aussi nous permettre de nous faire guider voir aider sur certains aspects techniques du projet, comme nous avons pu le faire sur la question de la communication serveur. L'objectif est donc maintenu dans la ligne de mire grâce à cet aspect des méthodes agiles.

L'utilisation du système des User stories basées sur le principe INVEST (Indépendante, Négociable, Valorisable, Estimable, Suffisamment petite et Testable) et sur le modèle « En tant que ..., je veux..., afin de ... » nous a permis de bien cibler chaque point spécifique du développement de l'application en fonction des différents acteurs qui auront à faire à la messagerie, mais aussi de pouvoir en discuter avec le tuteur afin de savoir quelle valeur il apporte à certains des aspects du projet. Cela nous a permis par la suite de prioriser les User stories dans les différents cycles incrémentaux au cours du projet et de mener précisément les tests utilisateurs. (cf.. « L'interface d'une carte du site Trello », plus haut, qui illustre une User story intégrée au Trello)

La planification adaptative a été réalisée de la manière décrite précédemment (4.2.2), directement sur le Trello et en équipe. A la fin de chaque réunion avec le tuteur, nous nous mettions d'accord afin de déterminer le tableau du sprint suivant en regroupant les notions fraîchement discutées ainsi que pour dresser un constat global du groupe. Un compte rendu de sprint était donc établi et envoyé au tuteur.

Ce qui nous emmène au point suivant : le management visuel et le travail collaboratif. L'outil qu'est le Trello nous a permis d'avoir un suivi de projet en temps direct et de donc de pouvoir l'avancer à n'importe quel moment, sans gêner les autres membres du groupe ou sans besoin immédiat de compréhension de tâches antérieures (User stories Indépendantes) avec la possibilité de partager l'aspect que nous venions d'incrémenter sans risque de briser le projet grâce à GitHub. (cf. Annexes 10 à 13 pour les versions de l'applications)

Le dernier point est donc basé sur cet aspect : les rituels de communication. Nous avions pour habitude de communiquer par l'intermédiaire de notre conversation de groupe sur Facebook ou encore via Slack, mais nous avions aussi besoin de nous voir pour être le plus efficace possible. Nous avons donc pris des temps de travaux extrascolaires à l'IUT qui nous servaient à faire le point de notre situation et de nous tenir à jour si le besoin s'en faisait sentir, avant de continuer en groupe le projet. Nous avons aussi mis en place un système de réunion après chaque sprint pour déterminer le tableau du sprint suivant en regroupant les notions fraîchement discutées avec le tuteur ainsi que pour dresser un constat global du groupe. Chacun inscrivait sur un papier les points positifs, négatifs et les voies d'amélioration possibles à la réalisation du projet, ce qui permettait, une fois l'information regroupée et analysée par notre Scrum Master, de se faire une opinion globale de la situation et ainsi de prendre les mesures adaptées pour améliorer cette opinion au prochain sprint. Nous avons pu ainsi apprendre de nos erreurs et par exemple retravailler nos User stories qui n'étaient pas bien définies, et ainsi revoir notre Backlog général de projet qui s'en est trouvé fortement modifié ou encore gérer les petits quiproquos.

Pour clôturer ce rapport, nous aborderons les points essentiels de notre projet qui sont l'apprentissage de l'environnement des messageries et des sockets en Java ainsi que la réalisation d'un projet avec les méthodes agiles.

En effet, ce projet avait pour but de réaliser un système d'échange de messages pour faciliter la communication à l'intérieur d'une éventuelle entreprise. Nous avons donc réalisé une messagerie instantanée sécurisée par SSL afin de répondre à ce besoin. Cette dernière peut donc être mise en place et configurée pour n'importe quelle entreprise en changeant simplement l'adresse IP de connexion. Notre messagerie assure ainsi un échange de messages protégé et pourrait dans le futur proposer de nouveaux services comme l'échange de documents tels que des images, des vidéos ou encore toute sorte de fichiers ainsi que l'amélioration de l'application d'un point de vue esthétique et ergonomique.

Ce projet nous a permis de découvrir le milieu des messageries ainsi que d'approfondir nos connaissances en Java ou encore sur les sockets en général puisqu'il a été entièrement codé sous Java avec l'API Sockets.

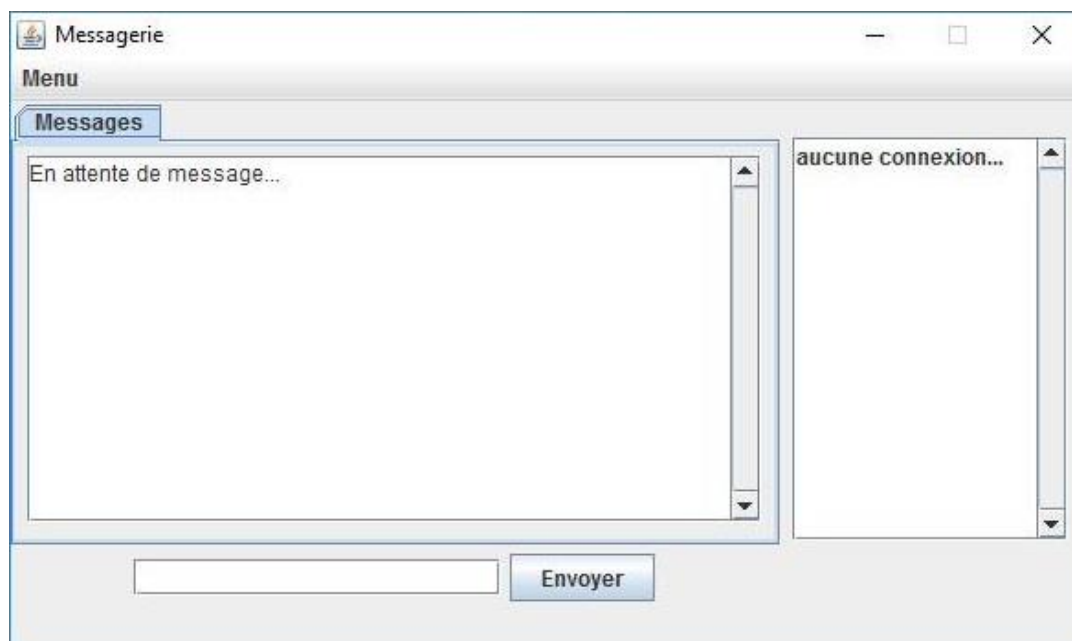
D'un point de vue organisationnel, nous avons utilisé les méthodes agiles avec plusieurs réunions au sein de notre équipe ou avec notre tuteur de stage. Nous avons donc pu apprendre pour certains ou approfondir pour d'autres comment ces méthodes permettent de garder le client au courant de l'avancement du projet et comment elles permettent d'améliorer le rendu final d'un projet.

Sites utilisés pour la communication ou la gestion de projet :

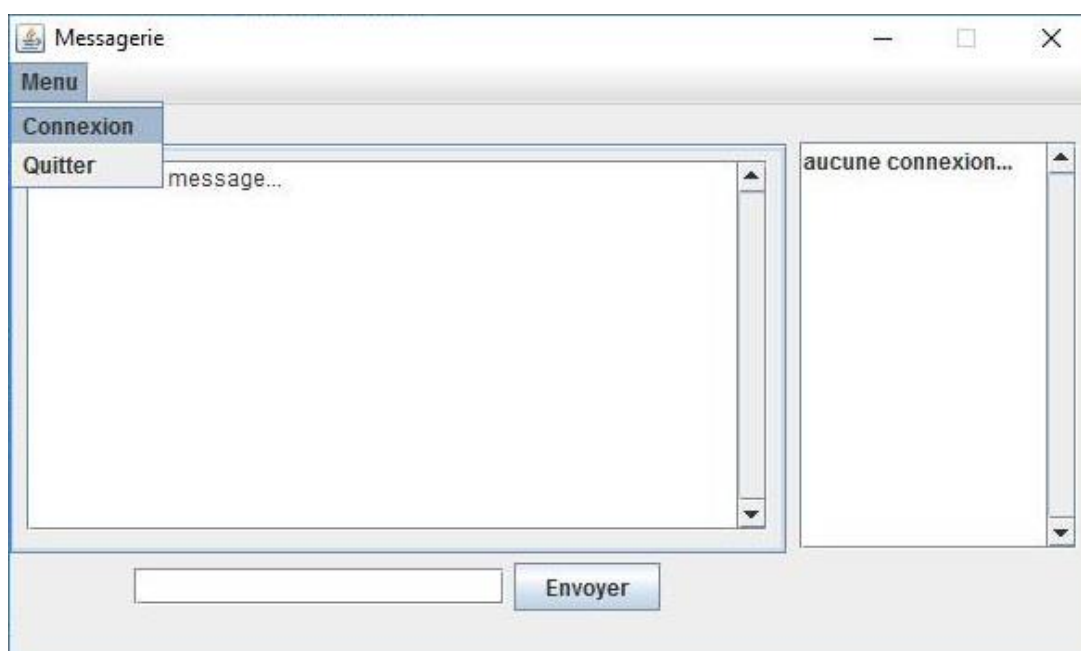
- <https://trello.com/>
- <https://slack.com/>
- <https://www.facebook.com/>
- <https://github.com/>

Sites utilisés pour le développement informatique :

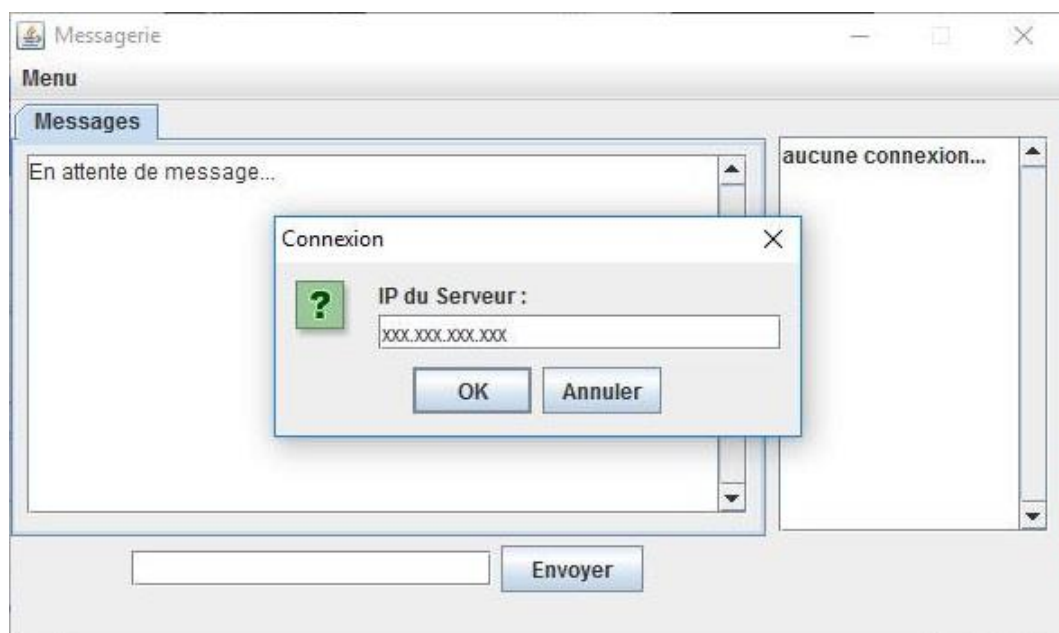
- <https://openclassrooms.com/>
- <http://www.developpez.com/>
- <https://stackoverflow.com/>
- [https://www.jmdoudoux.fr/accueil\\_java.html](https://www.jmdoudoux.fr/accueil_java.html)



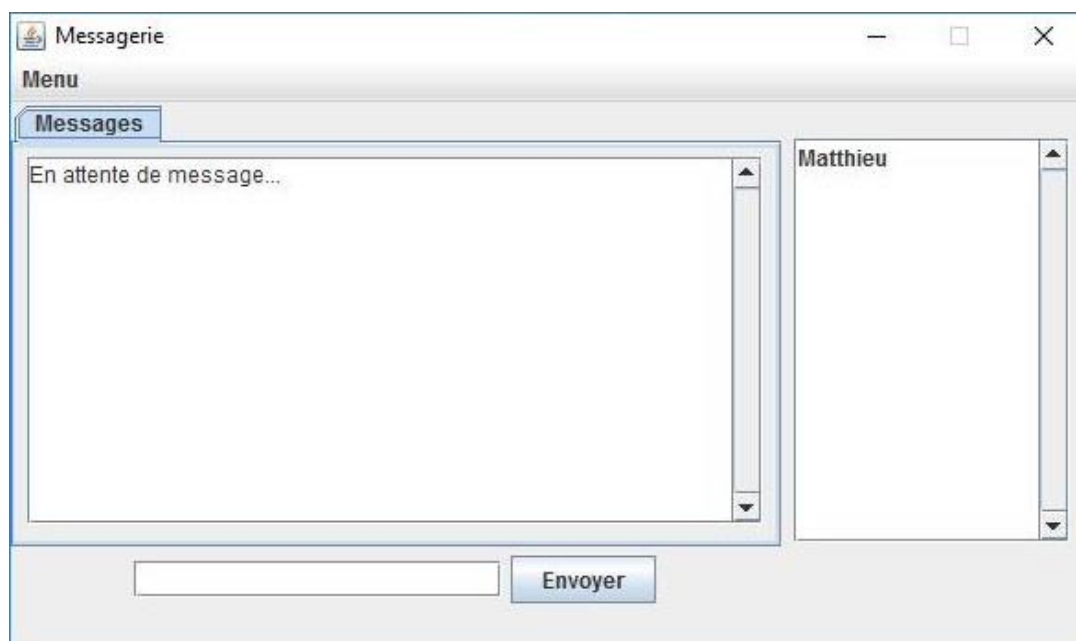
Annexe 1 – Application lancée



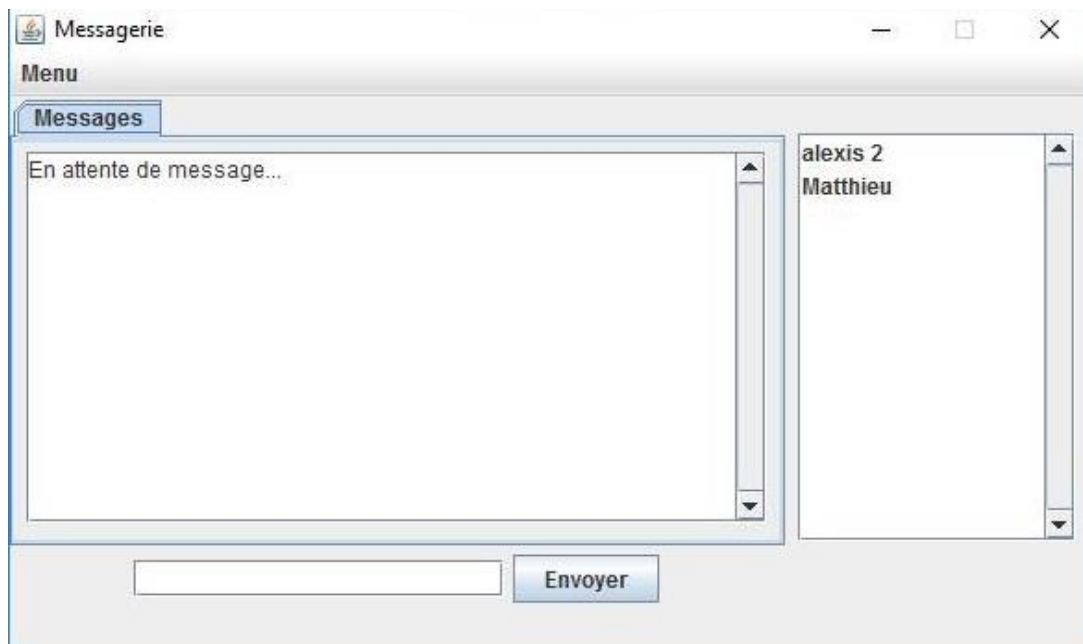
Annexe 2 – Connexion au serveur



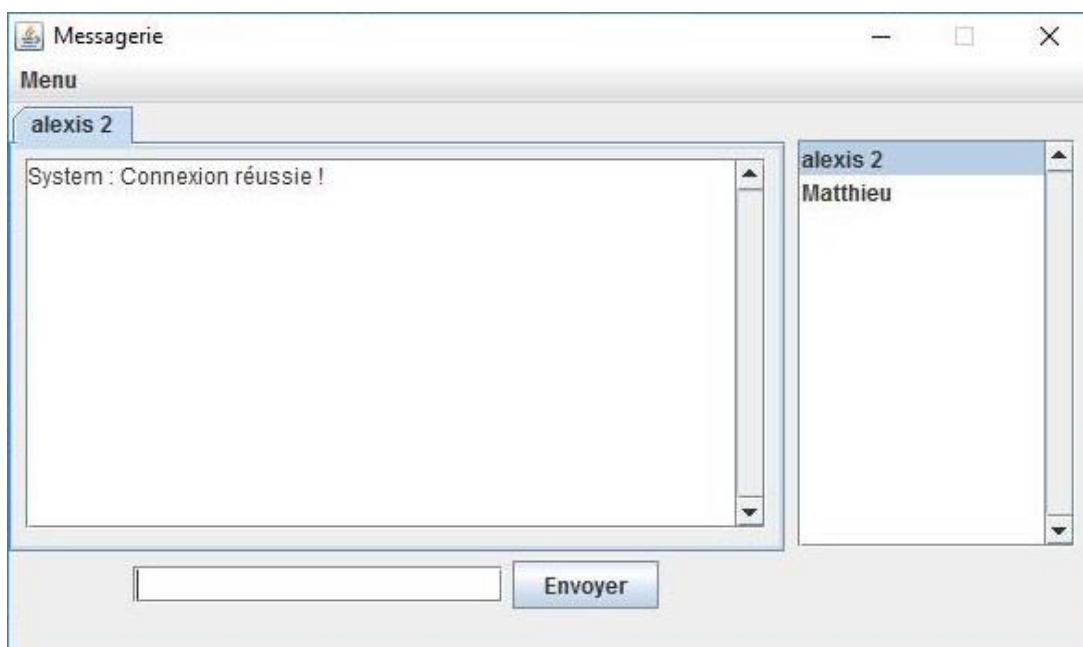
Annexe 3 – Pop-up IP serveur



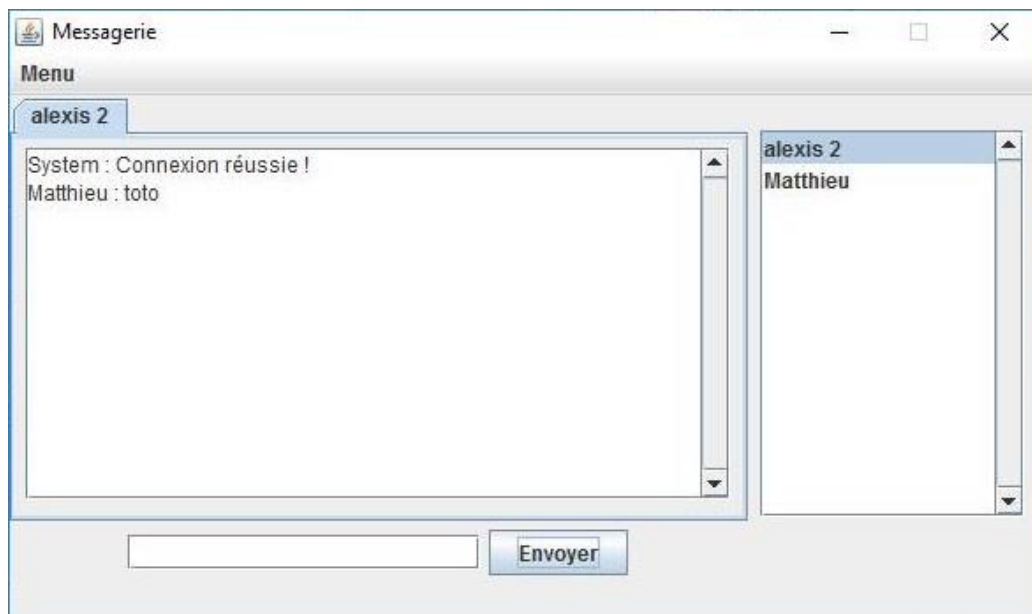
Annexe 4 – Connexion réussie



Annexe 5 – Liste des utilisateurs connectés

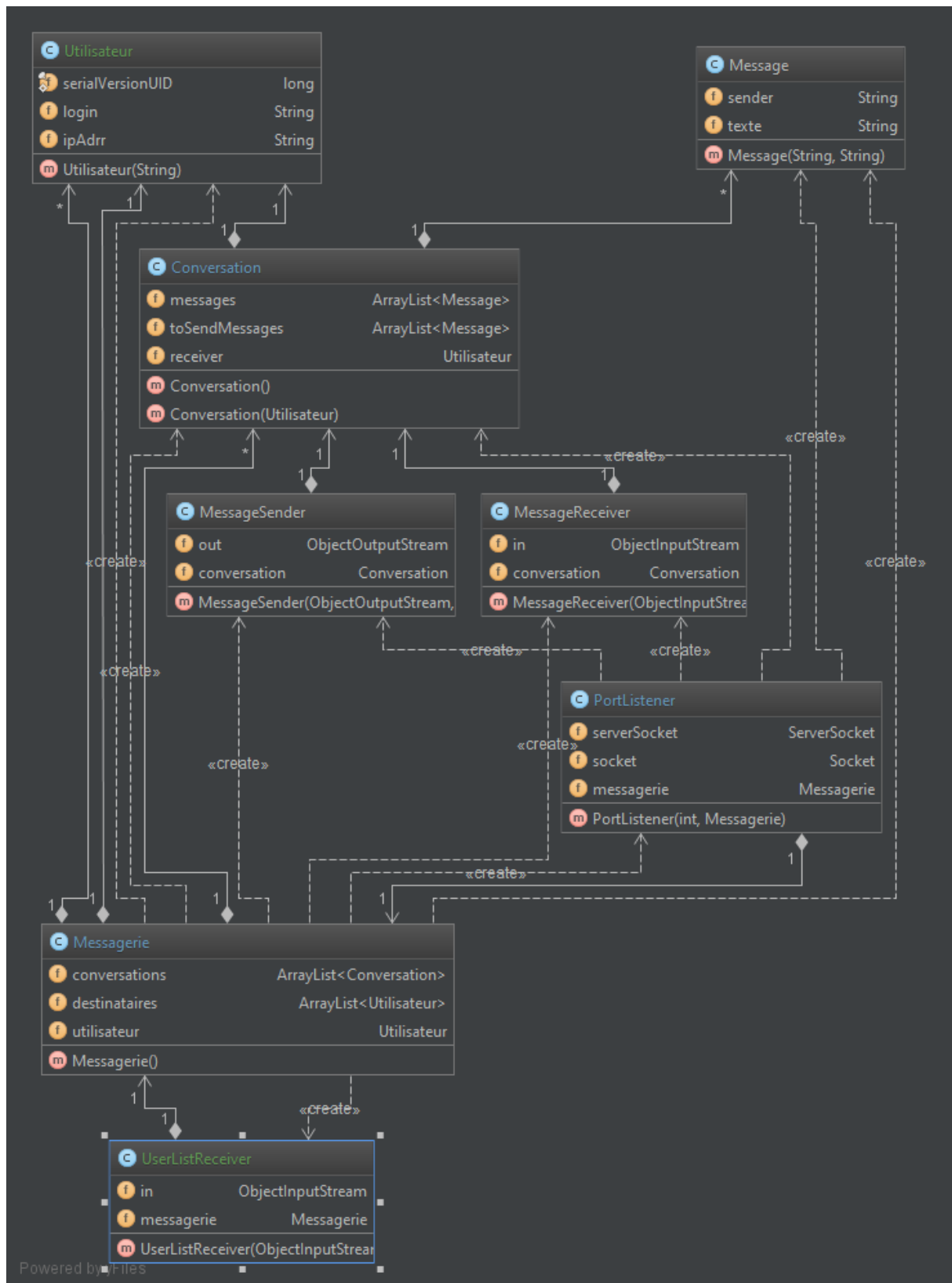


Annexe 6 – Connexion utilisateur réussie



Annexe 7 – Envoi de message





Annexe 8 – Diagramme de classe

## Annexe 9 – Explications des algorithmes (partie 1)

```
package model;

import ...

public class Messagerie {

    private ArrayList<Conversation> conversations;
    private ArrayList<Utilisateur> destinataires;
    private Utilisateur utilisateur;

    public Messagerie() {...}

    private void waitForConnectionOnPort(int port) { new Thread(new PortListener(port, this)).start(); }

    public void connectTo(Utilisateur destinataire) {
        try {
            Socket socket = new Socket(destinataire.getIpAdrr(), 2042);
            Conversation conversation = new Conversation(destinataire);
            conversation.addMessage(new Message("System", "Connexion réussie !"));
            conversations.add(conversation);

            ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
            out.flush();
            ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

            out.writeObject(this.utilisateur); // On envoie à notre destinataires nos informations

            new Thread(new MessageReceiver(in, conversation)).start();
            new Thread(new MessageSender(out, conversation)).start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

La fonction ci-dessus est utilisée pour démarrer une conversation avec un utilisateur distant.

Cet utilisateur possède un nom et une adresse IP. On utilise donc cette dernière pour créer une connexion grâce à l'API Socket puis on crée la nouvelle conversation correspondante que l'on ajoute à notre liste de conversations.

La seconde partie de cette fonction sert à récupérer les objets représentant les flux d'entrée et de sortie de notre socket que nous passons alors en paramètre des deux Threads d'émission et de réception de message. (Voir détail ci-dessous)

```

package model;

import ...

public class MessageReceiver implements Runnable {

    private ObjectInputStream in;
    private Conversation conversation;

    public MessageReceiver(ObjectInputStream in, Conversation conversation) {...}

    @Override
    public void run() {
        while (true) {
            try {
                conversation.addMessage((Message) in.readObject());
            } catch (IOException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
}

```

La classe `MessageReceiver` ci-dessus implémente l'interface `Runnable` qui nous a servi précédemment l'exécuter dans un nouveau `Thread`.

On peut voir que le `run()` de cette classe est relativement simple : la fonction `readObject()` est relative à l'API `Socket` et est bloquante tant que rien n'est détecté dans le flux d'entrée (ici notre objet `in`).

A la réception d'un message on "cast" ce dernier en objet `Message` et on l'ajoute à la conversation correspondante.

## Annexe 9 – Explications des algorithmes (partie 3)

```
package model;

import ...

public class MessageSender implements Runnable {

    private ObjectOutputStream out;
    private Conversation conversation;

    public MessageSender(ObjectOutputStream out, Conversation conversation) {...}

    @Override
    public void run() {
        while (true) {
            for(int i = 0; i<conversation.getToSendMessages().size(); i++) {
                try {
                    out.writeObject(conversation.getToSendMessages().get(i));
                    out.flush();
                    conversation.addMessage(conversation.getToSendMessages().get(i));
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            conversation.getToSendMessages().clear();
        }
    }
}
```

---

De la même façon que la classe précédente, l’envoi des messages s’effectue également dans un Thread différent.

Lorsqu’un utilisateur envoie un message, ce dernier est placé dans une liste de message “en attente” qui se trouve dans la conversation associée.

Le Thread d’envoi se contente donc de lire dans cette liste afin d’écrire les message un à un dans le flux sortant grâce à la méthode *writeObject()*. Après avoir envoyé tous les messages en attentes on efface la liste dans la conversation.

On notera que la méthode *getToSendMessages()* est synchronisée avec celle qui ajoute les messages en attentes à notre liste. Cette méthode restera bloquante tant qu’il n’y a pas de nouveaux messages à envoyer.

## Annexe 9 – Explications des algorithmes (partie 4)

```
package model;

import ...

public class PortListener implements Runnable {

    private ServerSocket serverSocket;
    private Socket socket;
    private Messagerie messagerie;

    public PortListener(int port, Messagerie messagerie) {...}

    @Override
    public void run() {
        while (true) {
            try {
                socket = serverSocket.accept(); // On attend d'être contacté par un correspondant
                Conversation conversation = new Conversation();
                conversation.addMessage(new Message("System", "Un utilisateur vient de se connecter"));

                ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStream());
                out.flush();
                ObjectInputStream in = new ObjectInputStream(socket.getInputStream());

                try {
                    conversation.setReceiver((Utilisateur)in.readObject()); // On récupère les infos envoyées par notre correspondant
                } catch (ClassNotFoundException e) {
                    e.printStackTrace();
                }
                messagerie.addConversation(conversation);

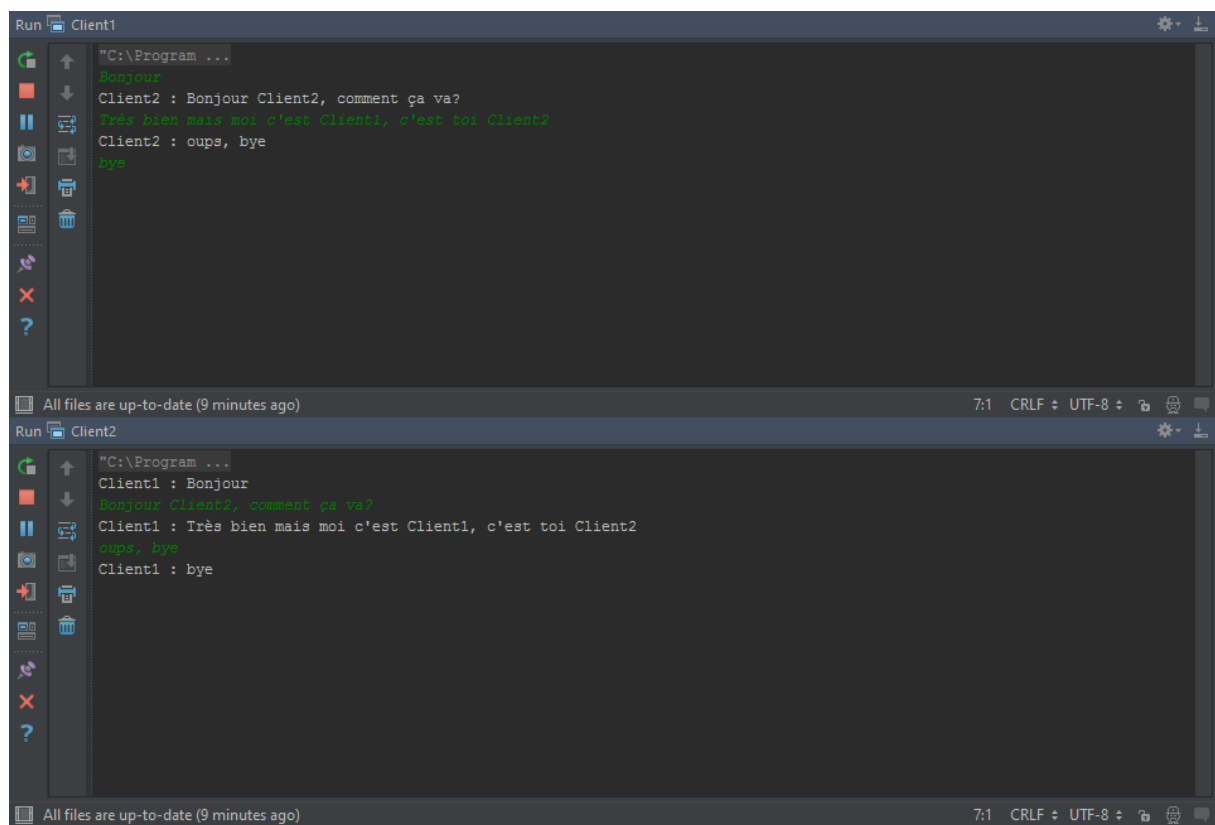
                new Thread(new MessageReceiver(in, conversation)).start();
                new Thread(new MessageSender(out, conversation)).start();

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Comme nous l'avons vu précédemment, on peut se connecter à un utilisateur distant afin de commencer une nouvelle conversation, pourvu que celui-ci écoute et comprenne les informations qu'on lui envoie.

C'est exactement le rôle de cette classe qui gère les connexions entrantes. Le code est le même que celui de la méthode *connectTo()* si ce n'est que le socket est créé après que la méthode *accept()*, qui est bloquante, ait reçu une connexion sur le port d'écoute.

On notera également que ce traitement se trouve dans une boucle qui va gérer à tour de rôle chacune des connexions entrante, d'où l'importance une fois de plus de l'effectuer dans un Thread indépendant.



The image shows two terminal windows side-by-side, representing the execution of a chat application. The top window, titled 'Client1', shows the following text: 'C:\Program ...', 'Bonjour', 'Client2 : Bonjour Client2, comment ça va?', 'Très bien mais moi c'est Client1, c'est toi Client2', 'Client2 : oups, bye', and 'bye'. The bottom window, titled 'Client2', shows: 'C:\Program ...', 'Client1 : Bonjour', 'Bonjour Client2, comment ça va?', 'Client1 : Très bien mais moi c'est Client1, c'est toi Client2', 'oups, bye', and 'Client1 : bye'. Both windows have a status bar at the bottom indicating 'All files are up-to-date (9 minutes ago)' and encoding settings '7:1 CRLF UTF-8'.

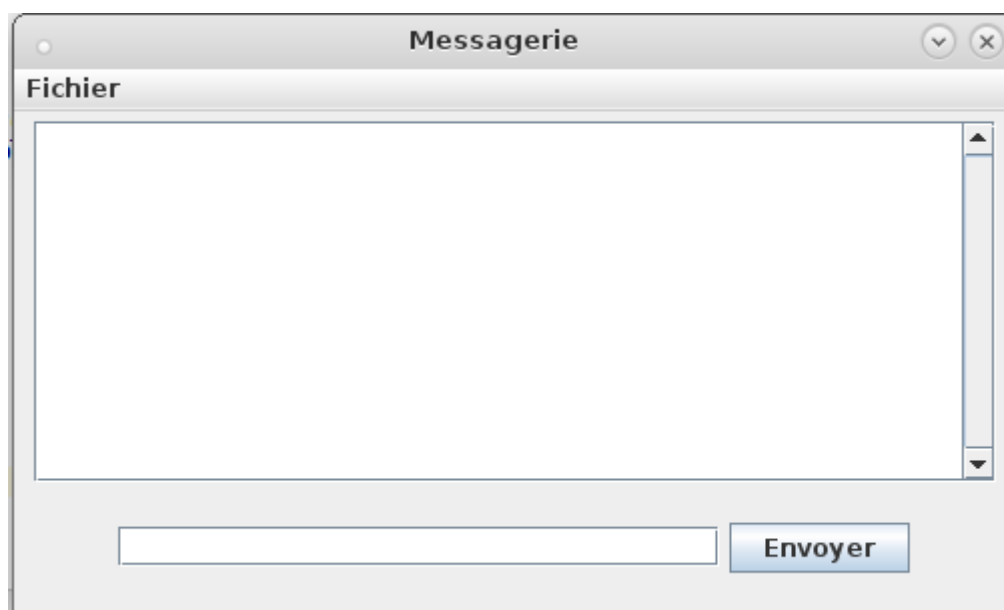
```
Run Client1
"C:\Program ..."
Bonjour
Client2 : Bonjour Client2, comment ça va?
Très bien mais moi c'est Client1, c'est toi Client2
Client2 : oups, bye
bye

All files are up-to-date (9 minutes ago) 7:1 CRLF UTF-8

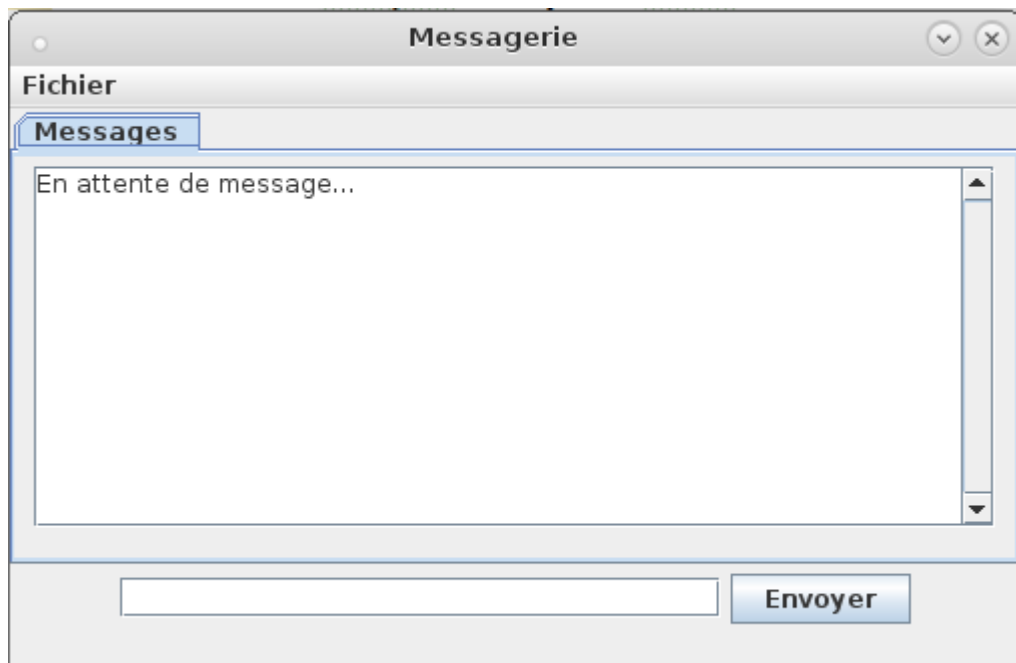
Run Client2
"C:\Program ..."
Client1 : Bonjour
Bonjour Client2, comment ça va?
Client1 : Très bien mais moi c'est Client1, c'est toi Client2
oups, bye
Client1 : bye

All files are up-to-date (9 minutes ago) 7:1 CRLF UTF-8
```

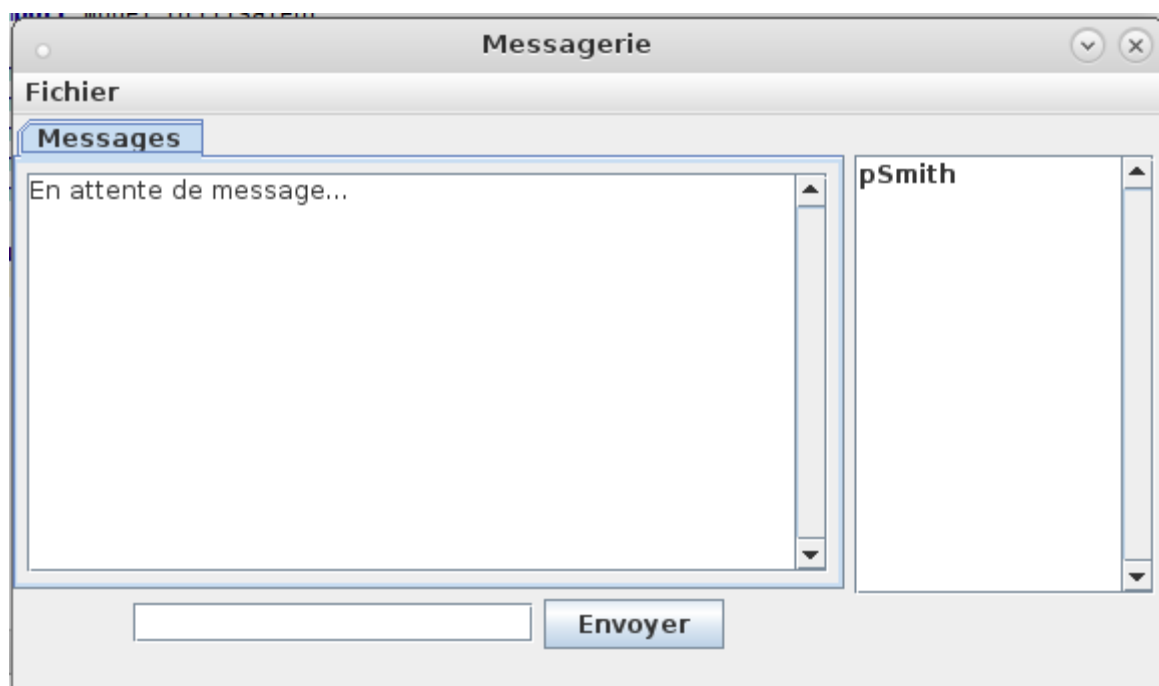
Annexe 10 – Version 1 de l’application



Annexe 11 – Version 2 de l’application : Ajout d’une interface graphique



Annexe 12 – Version 3 de l'application : Gestion de plusieurs conversations et onglet associé



Annexe 13 – Version 4 de l'application : Interaction avec le serveur et affichage de la liste des utilisateurs connectés

Le projet qui nous a été proposé a pour objectif principal de développer une messagerie instantanée pouvant servir sur le réseau interne d'une entreprise pour permettre d'améliorer la communication inter-membre. Le tout sécurisé par SSL afin de garantir l'intégrité des données.

Cette messagerie est développée en Java en faisant appel aux librairies javax.swing (interface graphique), java.net (application réseau, sockets), java.security (sécurité).

**Mots clés** : SSL, Messagerie instantanée, Client / Serveur, Socket, Java, Interface graphique

The main goal of our project is to create an instant messaging application, used by company on their local network, to upgrade the communication between its member. The integrity of data is preserved by SSL. This messaging is developed using Java, with libraries like javax.swing (graphic interface), java.net (network application, socket), java.security (security).

**Keywords** : SSL, Instant messaging, Client/server, Socket, Java, Graphic Interface