

Rapport POO

# Projet Messenger

---

BOULOU Valentin

GAUMART Siméon

5 février 2020

---

# Sommaire

|   |           |
|---|-----------|
| <b>Guide d'administration</b>                         | <b>2</b>  |
| Spécification de l'application :                      | 2         |
| Spécification de la base de données centrale :        | 2         |
| Spécification du serveur de présence :                | 2         |
| Guide de déploiement de la base de données centrale : | 3         |
| Guide de déploiement du serveur de présence :         | 3         |
| Guide de déploiement de l'application :               | 4         |
| <b>Guide d'utilisation</b>                            | <b>5</b>  |
| <b>Campagnes de tests</b>                             | <b>12</b> |
| Tests unitaires :                                     | 12        |
| Tests de l'application :                              | 12        |
| <b>Choix de conception</b>                            | <b>13</b> |
| Modèle :  | 13        |
| Base de données :                                     | 13        |
| Serveur de présence :                                 | 14        |
| Interconnexion réseau :                               | 14        |
| Gestion du multithreading :                           | 15        |
| <b>Evolutions possibles</b>                           | <b>16</b> |
| Serveur de présence :                                 | 16        |
| Interface graphique :                                 | 16        |

## Guide d'administration

Cette partie décrit comment déployer et installer l'application.

### Spécification de l'application :

- **Nom** : MessengerApp.jar
- **Taille** : 7,5 Mo
- **Ports utilisés** : [6666-6669]
- **Version de la JDK** : Java SE Development Kit 11.0.5+10-LTS
- **Commande d'exécution** : java -jar MessengerApp.jar
- **Librairies** :
  - mysql-connector.jar
  - sqlite-jdbc-3.21.0.jar

### Spécification de la base de données centrale :

- **Nom du serveur** : srv-bdens.insa-toulouse.fr
- **IP du serveur (Dernier accès : 05/01/20)** : 10.10.40.6
- **Nom de la base de données** : tpervlet\_13
- **Mot de passe** : La1yah4k
- **Commande de connexion** : mysql -h srv-bdens.insa-toulouse.fr -D tpervlet\_13 -u tpervlet\_13 -p
- **Version du client MySQL** : Ver 14.14 Distrib 5.7.29, for Linux (x86\_64) using EditLine wrapper
- **Version du serveur** : 5.7.29-0ubuntu0.16.04.1
- **Database engine** : innnoDB version 5.7.29

### Spécification du serveur de présence :

- **Nom du serveur** : srv-bdens.insa-toulouse.fr
- **IP du serveur (Dernier accès : 05/01/20)** : 10.1.5.2
- **Version du serveur** : Apache Tomcat/9.0.16 (Ubuntu)
- **Version de la JVM** : 11.0.4+11-post-Ubuntu-1ubuntu218.04.3
- **Adresse du servlet** : hrv-gei-tomcat.insa-toulouse.fr/Messenger/
- **Librairie utilisée pour générer le servlet** :
  - servlet-api-tomcat9.jar
  - Common.jar (librairie créée pour ce projet)

## Guide de déploiement de la base de données centrale :

Une fois le serveur mySQL déployé, il faut créer une base de données du nom de tpervlet\_13 et un compte éponyme avec le code La1yah4k.

Si vous désirez changer l'un de ces paramètres, vous devrez modifier les champs suivants dans le fichier Application.DBCentrale:

```
20 public class DBCentrale {  
21     private static String login = "tpervlet_13";  
22     private static String pswd = "La1yah4k";  
23     private static String URL = "jdbc:mysql://srv-bdens.insa-toulouse.fr:3306/"+login;
```

*Champs relatif à la connexion la db centrale*

Il n'y a aucune manipulation à faire ensuite. Les tables sont gérées automatiquement par les différents agents.

Pour voir si le serveur fonctionne, simplement se connecter via un client SQL avec la commande ci-dessus.

## Guide de déploiement du serveur de présence :

Une fois le serveur TOMCAT9 déployé, il suffit d'insérer le fichier Messenger.war fournit par l'interface manager de TOMCAT. Si celle-ci n'est pas installé, il suffit d'insérer le fichier Messenger dans le répertoire WEB racine de TOMCAT. Il faudra alors relancer TOMCAT pour que la modification soit prise en compte.

Si vous désirez changer l'adresse du serveur, vous devrez modifier le champ suivant dans le fichier Application.InternalSocket :

```
48  
49 public class InternalSocket {  
50     protected static final String PresenceServer = "https://srv-gei-tomcat.insa-toulouse.fr/Messenger/PresenceServer";
```

*Champs relatif à la connexion au servlet*

Il n'y a aucune autre manipulation à faire. Le servlet se met automatiquement à jour avec les différents agents.

Pour voir si le servlet fonctionne, taper l'URL suivant :

<https://srv-gei-tomcat.insa-toulouse.fr/Messenger/PresenceServer>

Il retourne la liste des utilisateurs connectés.



## Guide de déploiement de l'application :

Pour déployer l'application, il faut que la machine sur laquelle est exécuté le fichier MessengerApp.jar possède au moins l'environnement JRE 11, disponible sur le site d'ORACLE.

Un fois installé, il suffit de lancer la commande **java -jar MessengerApp.jar** et l'application se lance. Il est possible de copier le fichier jar n'importe où.

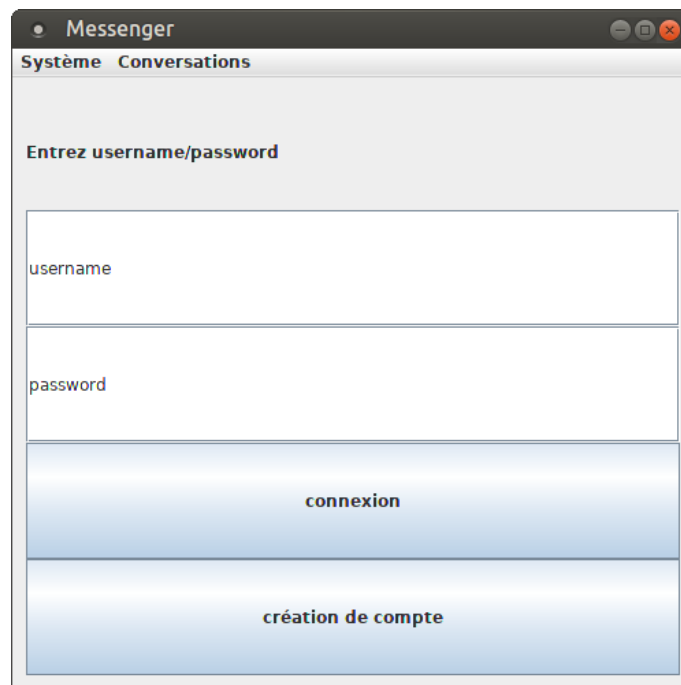
## Guide d'utilisation

Dans cette partie nous allons expliquer comment utiliser notre application une fois installée correctement.

L'application comporte 5 pages différentes ainsi qu'un menu composé de deux sous-menu :

- une page de connexion [1]
- une page de création de compte [2]
- une page de changement de pseudo [3]
- une page des utilisateurs connectés et déconnectés dont il existe une conversation [4]
- une page conversation [5]
- un sous-menu système [6]
- un sous-menu conversations [7]

À l'ouverture de l'application nous arrivons sur la **page de connexion** [1] :

The image shows a window titled "Messenger" with a dark title bar. Below the title bar is a menu bar with "Système" and "Conversations". The main content area has a light gray background. At the top, it says "Entrez username/password". Below this are two text input fields: the first is labeled "username" and the second is labeled "password". At the bottom, there are two large, light blue buttons with white text. The top button is labeled "connexion" and the bottom button is labeled "création de compte".

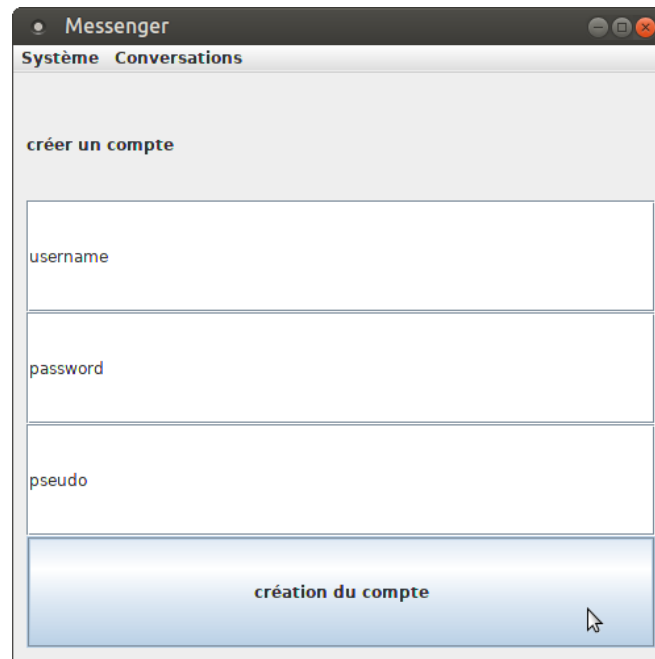
Pour se connecter, il suffit d'entrer son username et son password dans les champs correspondants, puis de cliquer sur le bouton "connexion" (ou d'appuyer sur Entrer).

Si le compte existe, nous arrivons sur la page [4].

Si le binôme username password ne correspond à aucun compte existant, la connexion est refusée et un message d'erreur "**erreur de connexion**" apparaît à la place de "Entrez username/password"

Dans ce cas, il faut créer un compte.

Pour créer un compte, il suffit de cliquer sur le bouton “création de compte” qui nous amène sur la **page de création de compte** [2] :

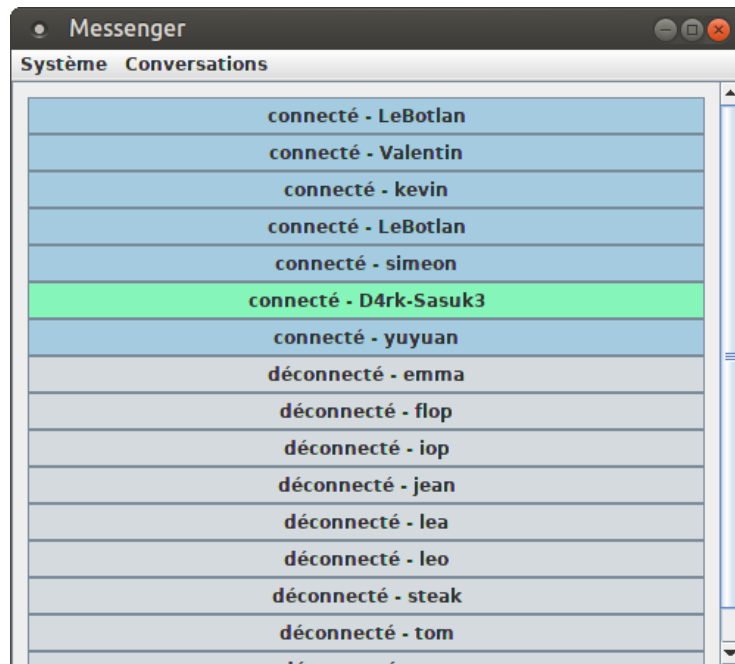


Entrez username, password et pseudo dans les champs correspondant. Cliquez ensuite sur le bouton “création de compte” (ou Entrer).

Si le compte est déjà existant (username et/ou pseudo existent déjà pour un autre compte), un message d’erreur rouge apparaît à la place de “créer un compte” (par exemple : “**erreur: pseudo déjà existant**”).

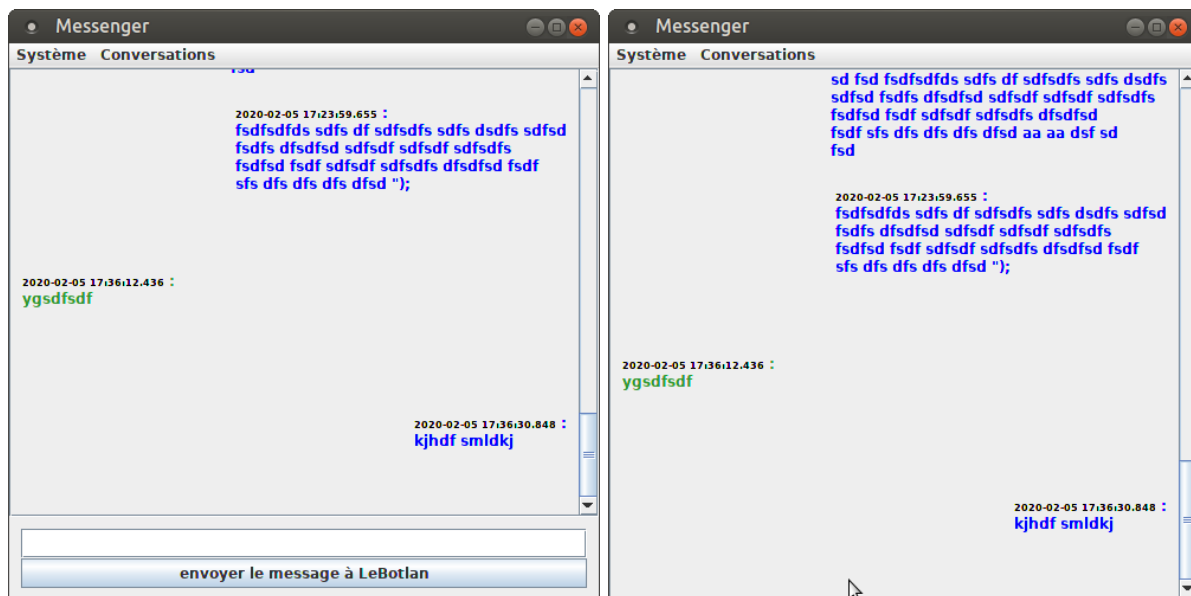
Si la création de compte a bien été effectuée, on revient automatiquement sur la page [1] avec un message “**création du compte de VotrePseudo réussie**”.

On peut enfin se connecter et on arrive sur la **page des utilisateurs connectés et déconnectés dont il existe une conversation** [4] :



Sur cette page, on peut voir le pseudo des utilisateurs connectés (en **bleu** avec marqué connecté devant), et le pseudo des utilisateurs déconnectés avec lesquels nous avons déjà eu une conversation sur ce compte (en **gris** avec marqué déconnecté devant). Les pseudos en **vert** sont ceux des utilisateurs qui nous ont envoyé un message depuis notre connexion, qui n'a pas encore été lu.

Pour voir la conversation avec un autre utilisateur, il suffit de cliquer sur le pseudo dans cette liste. On arrive sur la **page conversation** [5] suite à cette action :

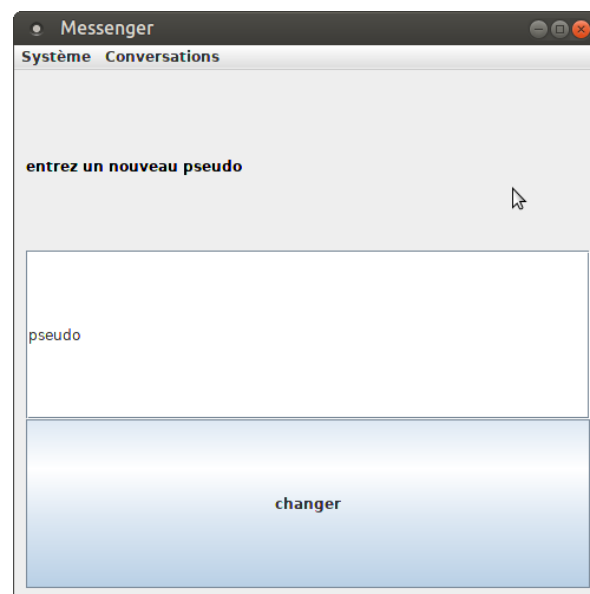




Si la personne est connectée, on arrive sur la page de gauche, et il est possible de lui envoyer des messages en écrivant dans la zone blanche et en cliquant sur “envoyer le message à SonPseudo” (ou Entrer). Si la personne est déconnectée il n’est pas possible de lui envoyer un message, on arrive sur la page de droite.

De plus il est possible de voir l’historique de la conversation (en scrollant vers le haut pour voir les messages les plus anciens) et chaque message est daté. Les messages **bleus** correspondent aux messages que nous avons envoyés, et les messages **verts** aux messages que notre correspondant a envoyé.

Il est aussi possible de changer le pseudo de son compte grâce à la **page de changement de pseudo** [3] (accessible via les sous-menu Système [6] -> modifier le pseudo) :



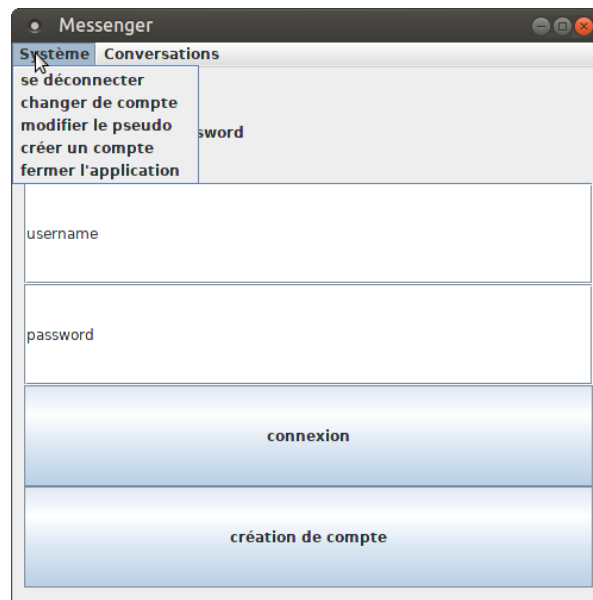
Pour changer son pseudo, il suffit d’entrer le nouveau pseudo dans le champ correspondant puis de cliquer sur “changer” (ou Entrer).

Si le pseudo est déjà utilisé, un message d’erreur “**erreur: pseudo déjà existant**” apparaît à la place de “entrez un nouveau pseudo”.

Si le changement de pseudo a bien été effectué, un message “**changement du pseudo réussi**” apparaît à la place de “entrez un nouveau pseudo”.

Il est aussi possible d’afficher la plupart des pages grâce aux sous-menus disponibles sur n’importe quelle page :

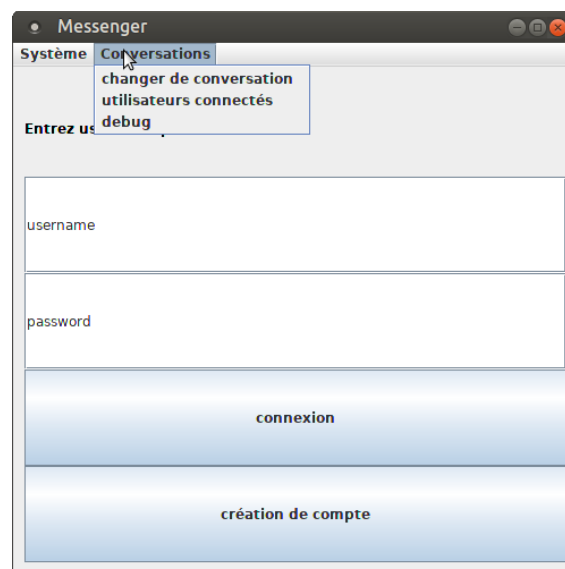
Le **sous-menu Systèmes** [6] :



qui est composé des items :

- se déconnecter : qui permet de se déconnecter si on l'est pas et d'aller sur la page de connexion [1]
- changer de compte : qui fait comme se déconnecter
- modifier le pseudo : qui permet d'aller sur la page de changement de pseudo [3] mais seulement si on est connecté
- créer un compte : qui permet d'aller sur la page de création de compte [2] en se déconnectant si on ne l'est pas déjà
- fermer l'application : qui permet de fermer l'application

et le **sous-menu Conversations [7]** :





qui est composé des items :

- changer de conversation : qui permet d'afficher la page des utilisateurs connectés et déconnectés [4] si on est connecté
- utilisateurs connectés : qui fait comme changer de conversation (*on fera pop une nouvelle pas dans une prochaine version*)
- debug : qui permet à l'utilisateur admin d'afficher toute la base de donnée locale dans le terminal.

## Campagnes de tests

### Tests unitaires :

**Toutes** les fonctions ont été testées de manière isolée dans un main subsidiaire qui a été supprimé au moment de produire la version finale de l'application. Cela a permis de d'isoler et corriger des comportements anormaux.

Pour vérifier le bon fonctionnement des composants réseaux tel que le serveur WEB ou les threads réseaux, nous avons eu recours à des fonctions auxiliaires créées pour le besoin. Certaines d'entre elles sont disponibles dans le dossier TestReseau.

### Tests de l'application :

Nous avons testé l'application avec plusieurs agents et testé toutes les fonctionnalités.

Avec l'application résultante, il ne nous semble pas possible de créer un comportement anormal autrement qu'en modifiant manuellement les données contenues les base de données ou le serveur WEB.

Cette application **N'EST PAS sécurisée**, toutes les informations sont disponibles en clair et rien n'a été fait dans l'objectif de fournir de la sécurité à l'utilisateur autre que la vérification l'intégrité des données.

## Choix de conception

Cette partie décrit les principaux éléments d'architecture de notre application.

### Modèle :

Notre application fonctionne sur le **modèle MVC** avec notamment un contrôleur et une interface graphique. Cela nous a permis de développer en parallèle ces deux éléments et de les fusionner à la fin du projet. Ainsi, chacun membre du binôme pouvait s'atteler de son côté à sa tâche.

### Base de données :

Notre application s'appuie sur une base de données hybride : une base de données distante constamment active et une base de données locale. En effet, le cahier des charges imposait que l'application puisse supporter 1000 sessions de clavardages simultanées [CdC-Bs-22]. Il paraît donc peu raisonnable de tout sauvegarder sur un serveur, cela pourrait vite surcharger celui-ci et rendrait notre application fortement dépendante de celui-ci. De plus, une base de données locale peut très vite poser un problème de cohérence de l'information entre agent ainsi qu'un manque flagrant d'ergonomie : un utilisateur perd ainsi tout historique des conversations passées s'il n'a plus accès à son poste de travail.

Nous avons donc opté pour une option hybride qui permet d'allier les qualités de centralisation fournit par un serveur de base de données et les qualités de scalabilité fournit par une base de données locale.

L'agent récupère ainsi les données via le serveur au début d'une session et les sauvegardes à la fin de celle-ci. Le reste du temps, tout est sauvegardé en locale.

Dans le but de minimiser la quantité d'information envoyé par le réseau, nous n'envoyons seulement l'information que l'une des bases de données ne possède pas. Par exemple, à la fin d'une session, seule l'information acquis modifier ajouter ou modifier pendant la session est envoyée au serveur.

### Serveur de présence :

Le serveur web de présence permet à un utilisateur qui n'est pas sur le même **réseau de broadcast** qu'un autre utilisateur de pouvoir communiquer avec lui. En effet, ce serveur tient à

jour une **liste des utilisateurs connectés** qu'il peut transmettre aux différents agents lorsqu'ils le désirent.

Comme le serveur ne peut, de lui-même, envoyer de l'information aux agents lorsque cette liste est modifiée, il faut que chaque agent vienne régulièrement faire des requêtes auprès du serveur afin de mettre à jour leur liste locale des utilisateurs connectés. C'est donc la régularité de cette requête qui détermine le temps de propagation de l'information auprès des utilisateurs dit "externes" de la connexion d'un utilisateur [CdC-Bs-25], du changement de pseudonyme [CdC-Bs-20], etc...

Tout comme avec le serveur de base de données, dans le but de **minimiser la quantité d'information envoyé** par le réseau, seuls les changements survenus entre 2 requêtes d'un agent lui sont envoyés.

## Interconnexion réseau :

Nous avons choisi de faire la découverte des agents locaux via un **broadcast UDP**. Cela permet d'atteindre les utilisateurs locaux efficacement. Nous avons choisi le broadcast UDP par rapport à l'IP multicast car, le but étant simplement d'atteindre les utilisateurs locaux, nous n'avons pas l'utilité que le paquet "passe" un routeur. En effet, la découverte des utilisateurs externes passe par le serveur de présence. Nous utilisons la même technologie pour signaler qu'un utilisateur change de pseudo.

L'envoi de message passe par un **socket TCP/IP**, car nous devons nous assurer que le message est correctement reçu par l'autre agent [CdC-Bs-30]. De plus, TCP nous assure que les messages seront reçus dans l'ordre, ce qui est grandement appréciable pour notre application. Cela évite de vérifier dynamiquement l'ordre des messages. Finalement, cela assure aussi que l'autre agent a reçu le message alors qu'il est encore actif, ce qui évite toute incohérence dans les bases de données locales des agents.

Pour terminer, la liaison entre les agents et le serveur WEB se fait via des requêtes **HTTP GET** et **HTTP POST**. Cela facilite le déploiement du serveur présence, qui est simplement un servlet qui peut simplement se greffer à un serveur WEB existant qui accepte les servlets JAVA tel qu'Apache TOMCAT.

## Gestion du multithreading :

Afin de profiter des processeurs multicœurs présent sur la plupart des PC actuels et du parallélisme qu'offre le **multithreading**. Notre application possède un thread main, 2 thread permanent et quelques threads éphémères.

Le thread **main** contient le contrôleur, il gère donc l'affichage, et l'envoi des messages.

Les deux threads **permanents** contiennent respectivement un serveur UDP et un serveur TCP. Ainsi leur fonctionnement n'altère pas celui du thread main et permet de tout de même être capable de réagir lorsque l'agent reçoit un message.

Les threads **éphémères** sont potentiellement créés lorsque le serveur TCP reçoit une connexion, le serveur crée alors un thread afin que celui-ci puisse gérer la communication et que le serveur puisse retourner dans un état d'écoute.

Il est aussi périodiquement nécessaire de créer un thread qui s'occupera de faire une requête auprès du serveur de présence. Ce thread contient donc un client temporaire HTTP.

## Evolutions possibles

Cette partie met en évidence des évolutions possibles de l'application si elle devait être déployée.

### Serveur de présence :

**L'une des causes principales de trafic superflue** sur le réseau venant de nos applications vient du fait les agents soient obligés de faire des requêtes au serveur de présence en permanence afin de se mettre à jour. Une future version de l'application pourrait faire en sorte que ce soit **le serveur de présence qui prenne l'initiative** de transmettre à tous les agents une information à chaque fois qu'il en reçoit une. Cela éviterai de surcharger inutilement le réseau avec ces requêtes qui ne servent la plupart du temps à rien, puisque le taux de rafraîchissement qui est de quelques secondes, imposés indirectement par le cahier des charges **[CdC-Bs-25]**, est bien supérieur comparé à la probabilité qu'un agent modifie la liste du serveur de présence.

### Interface graphique :

Dans le but d'apprendre comment utiliser Java SWING, nous n'avons pas souhaité utiliser de plug-in Designer (plug-in permettant de concevoir graphiquement l'interface graphique). Cela aurait sûrement permis d'obtenir une interface graphique plus esthétique et ergonomique, en passant par l'utilisation de template déjà conçue ou par l'utilisation du designer.