INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de projet

POO : Logiciel de clavardage

Rédigé par :

SIMARD Yohan – VERGNET Arnaud

– Version du 14 février 2021 –

Table des matières

Introduction

Clavardator est un système de communication textuel, permettant aussi le partage de fichiers. Il a été réalisé de novembre 2020 à février 2021 par Yohan SIMARD et Arnaud VERGNET à l'aide de Java 11 et JavaFX 11.

Ce document présente deux manuels d'utilisation : Un premier destiné à l'utilisateur final, expliquant les démarches à suivre pour commencer une discussion, et un autre destiné au technicien, expliquant les étapes d'installation. Ensuite, les choix de conception et les tests de validation sont explicités en détail.

Le logiciel Clavardator est compatible sur toutes les plate-formes bureau majeures, à savoir Linux, MacOS et Windows, à condition d'installer au minimum Java 11. Il est capable de fonctionner sur réseau local simple, ou sur des réseaux plus complexes grâce à un serveur de présence centralisé. Pour des raisons techniques, le logiciel n'a été testé que sur Linux.

La source de ce logiciel est disponible sur le serveur git étudiant.

1 Manuel d'utilisation

Manuel destiné à l'utilisateur. Pour toute information concernant l'installation sur poste et le déploiement du serveur de présence, merci de vous référer à la section Manuel de déploiement.

Pour lancer le logiciel, merci de vous référer aux instructions données par votre technicien.

1.1 Premier lancement

Lors du premier lancement, le logiciel vous demandera un nom d'utilisateur unique. Si votre sélection est déjà utilisée par un autre utilisateur, il vous sera demandé de changer. Vous aurez toujours la possibilité de le changer plus tard.

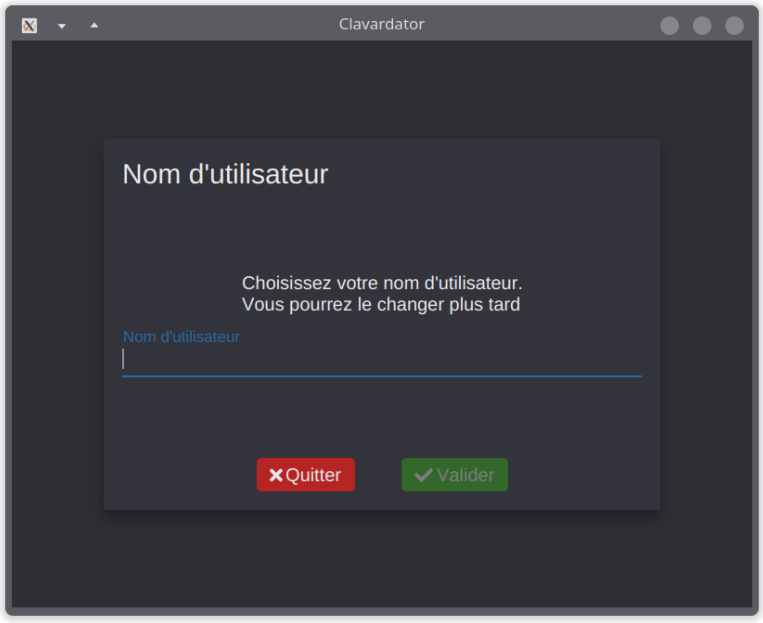


FIGURE 1 – Page de connexion

1.2 Prise en main de l'interface

Après avoir choisi votre nom d'utilisateur, vous arriverez sur la page principale de l'application. Cette page est découpée en trois parties. Tout en haut, vous trouverez une barre d'outils. À gauche se trouve la liste des utilisateurs actifs, et à droite se trouve la fenêtre de chat (vide pour l'instant).

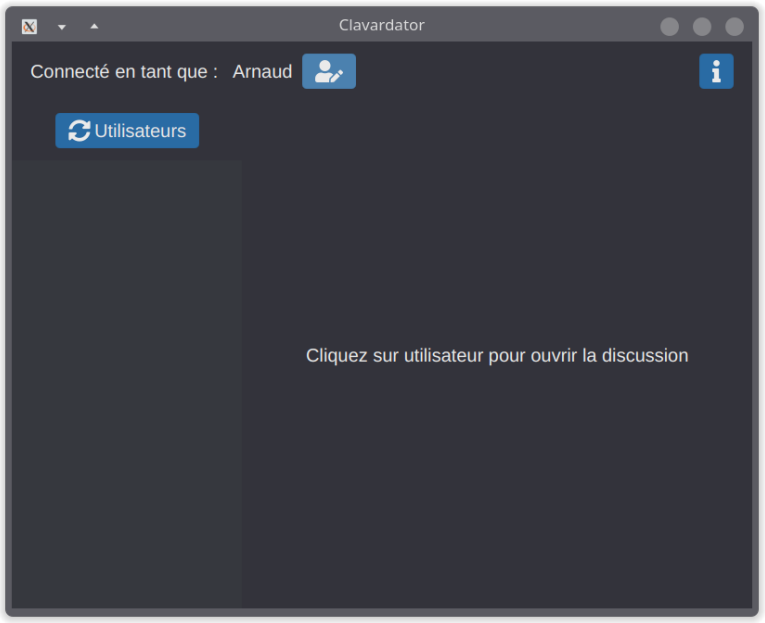


FIGURE 2 – Page principale

1.3 Voir les utilisateurs

Comme expliqué plus haut, la liste des utilisateurs se trouve à gauche de l'écran principal. Cette liste affiche tous les utilisateurs actifs (avec une pastille verte), et les inactifs avec qui vous avez déjà discuté (pastille grise). Cette liste est mise à jour dès qu'un utilisateur se connecte ou déconnecte du réseau. Si un utilisateur n'apparaît pas dans la liste, appuyez sur le bouton rafraîchir.

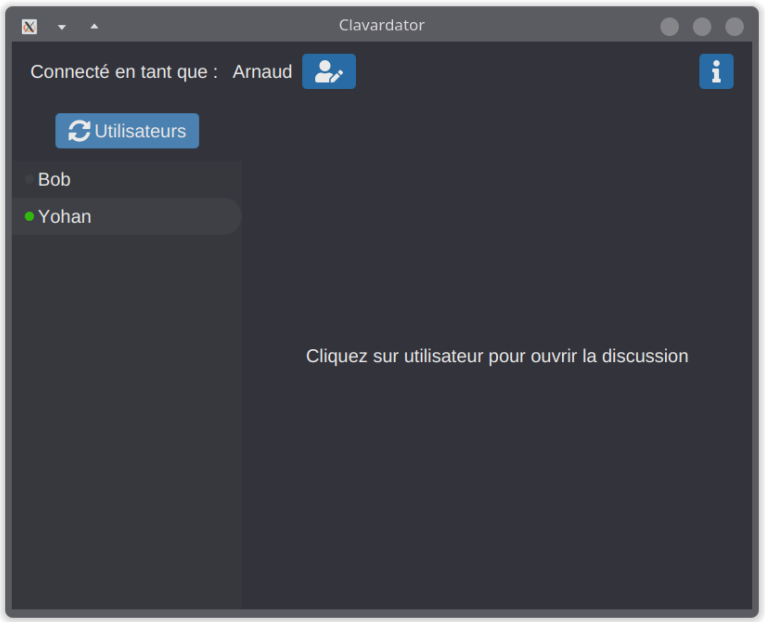


FIGURE 3 – Liste des utilisateurs

1.4 Démarrer une session de chat

Pour commencer une discussion, il suffit de cliquer sur le nom d'un utilisateur actif. La partie droite de l'écran affichera alors une page de discussion.

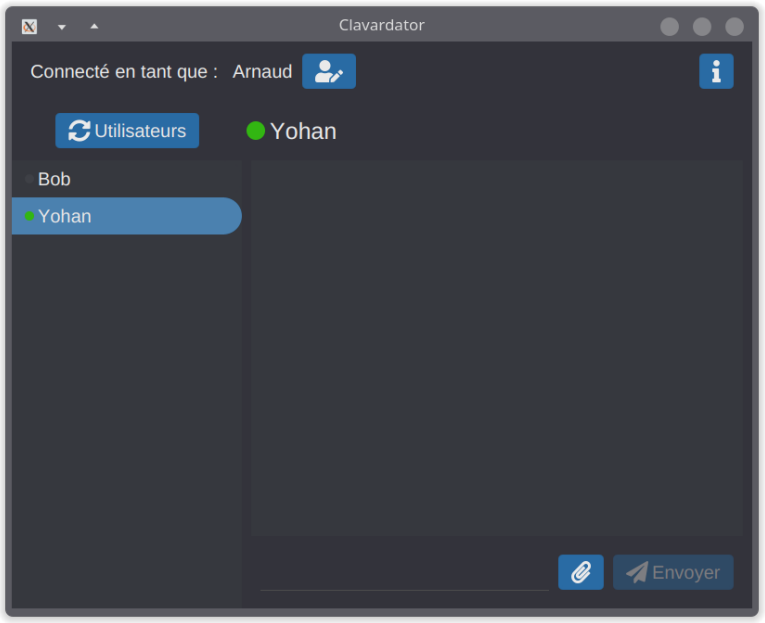


FIGURE 4 – Démarrage d'une session de clavardage

1.4.1 Envoyer et recevoir du texte

Pour envoyer un message, il suffit d'écrire le texte dans le champ en bas de l'écran, et d'appuyer sur la touche "Entrer" ou le bouton "Envoyer". Votre message apparaîtra alors dans la liste en bleu. Quand vous recevez un message, il apparaîtra dans la liste en gris.

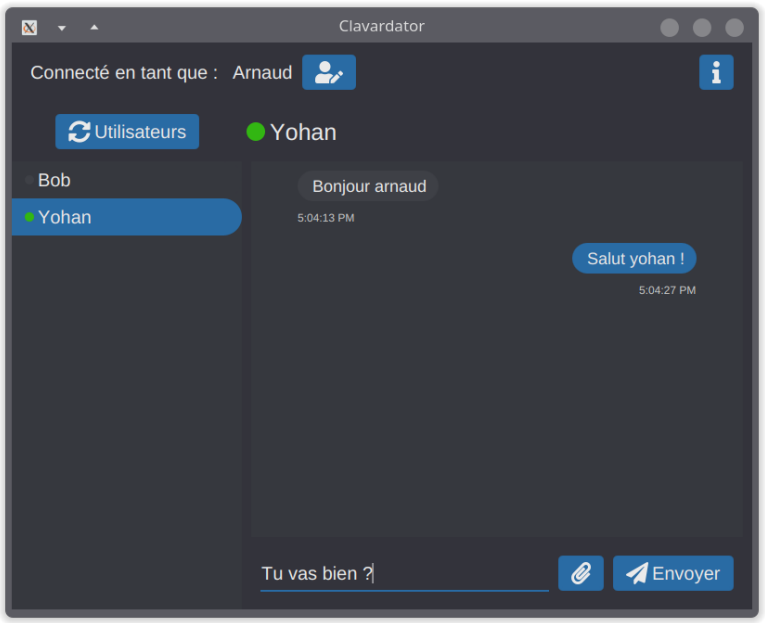


FIGURE 5 – Envoie et réception de messages textuels

1.4.2 Envoyer et recevoir des fichiers

Pour envoyer un fichier, cliquez sur le bouton en forme de trombone, à gauche du bouton "Envoyer", puis sélectionnez le fichier à envoyer. Vous pouvez entrer un message pour accompagner votre fichier. Quand vous êtes prêts, envoyez votre message de la même façon que du texte. Quand vous recevez un fichier, une icône de trombone apparaîtra à côté du message, un clic sur ce message suffit pour ouvrir le fichier.

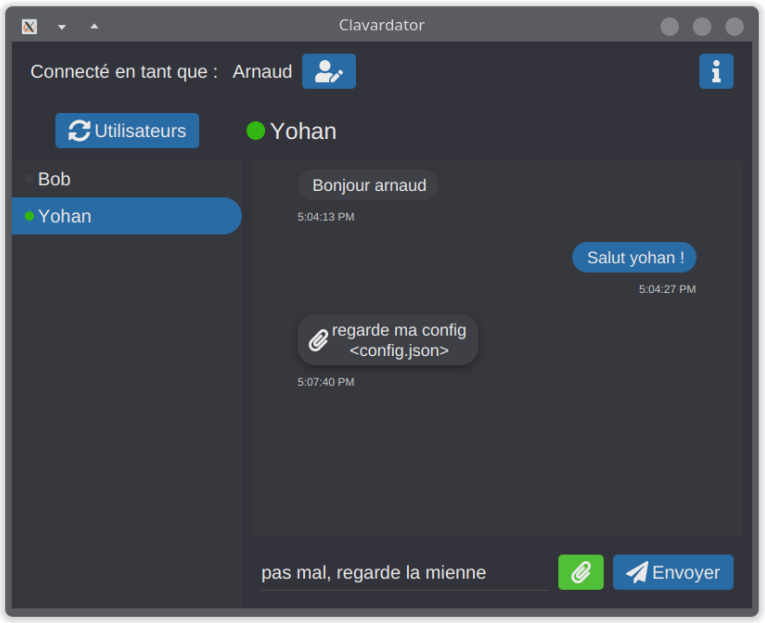


FIGURE 6 – Envoie et réception de fichiers

1.5 Voir d'anciens messages

Pour voir l'historique d'une conversation, cliquez sur l'utilisateur souhaité dans la liste (peu importe s'il est connecté ou non), puis remontez dans la liste des messages.

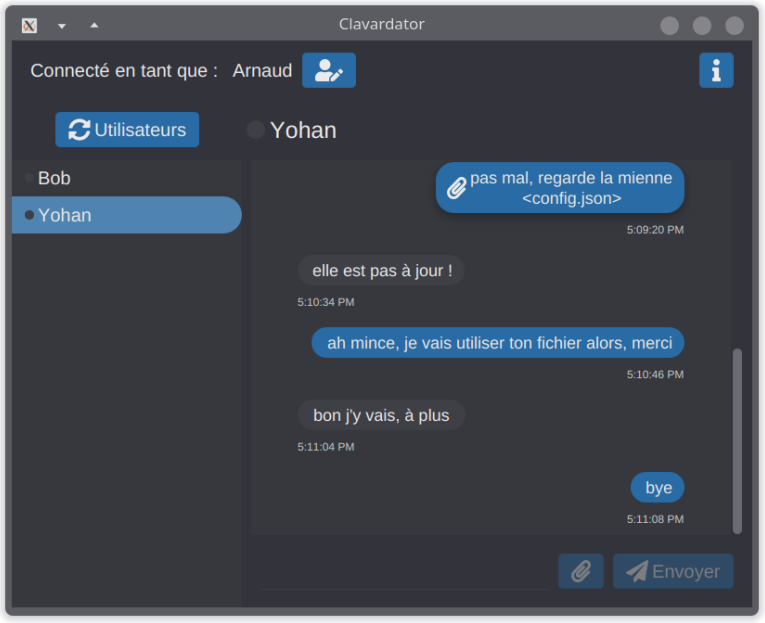


FIGURE 7 – Historique d'une conversation

1.6 Changer de nom d'utilisateur

Vous pouvez changer de nom d'utilisateur à tout moment, à condition qu'il ne soit pas déjà utilisé par une autre personne. Pour changer, cliquez sur le bouton "Modifier le nom d'utilisateur" dans la barre d'outils, puis entrez votre nom dans la fenêtre qui s'ouvre.

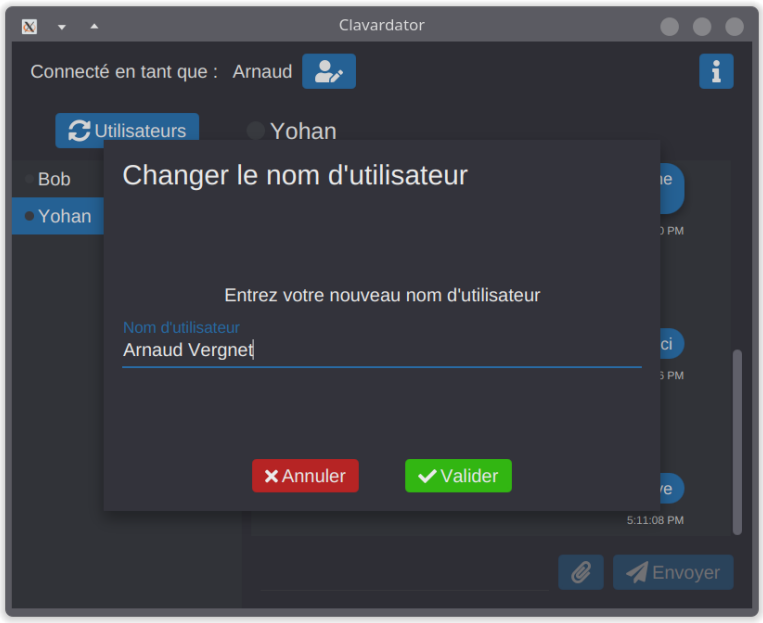


FIGURE 8 – Changement de nom d'utilisateur

1.7 Sauvegarder ses messages

Tous les messages sont sauvegardés dans le fichier clavardator.db, à côté de l'exécutable. Copiez ce fichier pour sauvegarder vos messages. Si vous ne savez pas où est situé ce fichier, merci de demander à votre technicien. Pour restaurer vos messages (suite à un changement de poste par exemple), remplacez le fichier présent sur la machine de destination par votre version copiée.

2 Manuel de déploiement

Manuel destiné à l'administrateur poste. Pour toute information concernant l'utilisation et la prise en main du logiciel, merci de vous référer à la section précédente *Manuel d'utilisation*.

Le système est composé de deux logiciels : le logiciel sur poste et le serveur de présence. Le serveur de présence est optionnel si vous êtes sur réseau local supportant le broadcast, mais devient obligatoire dans les autres cas.

2.1 Prérequis

Peu importe vos choix (réseau local seul ou avec serveur de présence), les étapes suivantes sont nécessaires. Nous ne fournissons pas de binaires du logiciels, mais les sources sont disponibles sur le serveur git étudiant.

2.1.1 Installation des dépendances

Toutes les machines accueillant le logiciel ou le serveur doivent disposer de Java 11 minimum. Merci de l'installer en suivant les informations officielles.

Ensuite, il est nécessaire de préparer une machine pour compiler les logiciels. **Les instructions suivantes supposent que vous utilisez un système Ubuntu**. Sur cette machine, installez le JDK openJDK 11 et git avec la commande suivante.

```
sudo apt install openjdk-11-jdk git
```

2.1.2 Préparation des binaires

Une fois les dépendances installées, clonez le projet sur votre machine en utilisant **HTTPS** (SSH étant désactivé). Pour cela, utilisez la commande suivante dans le dossier de votre choix :

```
git clone https://git.etud.insa-toulouse.fr/vergnet/clavardator.git
```

Un dossier `clavardator` sera créé. Déplacez vous à l'intérieur de ce dossier, puis compilez les logiciels à l'aide de Gradle.

Pour le logiciel client :

```
./gradlew client:build
```

Le binaire compilé se trouve dans `client/build/libs/client-0.0.1-all.jar`. **Ne pas utiliser `client-0.0.1.jar`, car ce fichier n'inclut pas toutes les dépendances.**

Pour le logiciel serveur :

```
./gradlew server:build
```

Le binaire compilé se trouve dans `server/build/libs/server-0.0.1-all.jar`. **Ne pas utiliser `server-0.0.1.jar`, car ce fichier n'inclut pas toutes les dépendances.**

2.2 Logiciel client

2.2.1 Installation

Récupérez le logiciel compilé à l'étape précédente et placez le sur la machine cible, si possible dans un dossier dédié accessible en écriture pour permettre au logiciel de générer ses propres fichiers (base de donnée et fichiers reçus).

Créez ensuite un script pour lancer le logiciel avec la commande suivante :

```
java -jar client-0.0.1-all.jar
```

Les utilisateurs n'auront plus qu'a exécuter ce script pour lancer le logiciel client.

2.2.2 Configuration

La configuration du client se fait par poste à l'aide d'un fichier JSON. Ce fichier doit se trouver dans le même dossier que l'exécutable et s'appeler `config.json`. Voici un exemple de configuration

```
{
  "serveur": {
    "actif": 1,
    "uri": "localhost",
    "type": "INSA",
    "ports": {
      "presence": 35650,
      "proxy": 35750
    }
  },
  "local": {
    "actif": 1,
    "port": 31598
  }
}
```

Ce fichier contient deux objets : `serveur` pour configurer la connexion au serveur de présence, et `local` pour configurer la connexion au réseau local. Si la clé actif d'un objet est à 0, alors toutes les autres clés de l'objet sont ignorées.

La table suivante donne des informations pour chaque clé ainsi que leur valeur par défaut.

Serveur		
Clé	Description	Défaut
actif	Active ou désactive la connexion au serveur de présence.	0
uri	Le chemin vers le serveur de présence. Cela peut être une adresse ip ou une url. Si ce champ est laissé vide, le serveur ne sera pas activé.	/
type	Le type de serveur de présence. Le système est créé de façon à pouvoir facilement ajouter de nouvelles implémentations de serveur de présence. Pour le moment, seul le type INSA est valide.	INSA
ports		
presence	Le port utilisé pour se connecter au serveur de présence.	35650
proxy	Le port utilisé pour se connecter au serveur de proxy.	35750

Local		
Clé	Description	Défaut
actif	Active ou désactive la connexion au réseau local.	1
port	Le port à utiliser pour se connecter aux autres instances sur le réseau local.	31598

2.3 Serveur de présence

2.3.1 Installation

Récupérez le logiciel compilé à l'étape *Préparation des binaires* et placez le sur la machine cible, dans le dossier de votre choix.

Pour lancer le serveur, exécutez la commande suivante :

```
java -jar server-0.0.1-all.jar
```

2.3.2 Configuration

Le logiciel serveur s'utilise en ligne de commande, plusieurs arguments sont autorisés :

-h, --help Affiche l'aide

-v, --version Affiche la version

-x <port>, --proxy <port> Spécifie le port à utiliser pour le proxy. Valeur par défaut : 35750.

-p <port>, --presence <port> Spécifie le port à utiliser pour le serveur de présence. Valeur par défaut : 35650.

Par exemple, pour lancer le serveur sur les ports 37000 et 38000 :

```
java -jar server-0.0.1-all.jar --proxy 37000 --presence 38000
```

3 Conception

3.1 Choix des technologies

Nous avons décidé d'utiliser Java 11 car c'était la version LTS lors de la conception de ce logiciel. Pour la compilation, nous avons adopté Gradle, car cet outil permet de gérer facilement les dépendances et nous a permis de découper notre projet en sous projets. Grâce à sa simplicité d'utilisation, le déploiement à été grandement facilité.

Pour permettre un plus grande indépendance des logiciels, nous avons opté pour un système avec base de donnée décentralisée. Chaque poste possède son propre fichier de base de donnée qu'il peut récupérer à tout moment.

3.1.1 Client

Pour réaliser l'interface graphique du logiciel client, nous avons préféré JavaFx 11 à Swing. En effet JavaFx (JFX) permet de définir l'interface dans des fichiers xml, et d'y associer une classe Java, le contrôleur. Il est aussi possible d'utiliser du CSS pour appliquer un style aux composants. Nous avons donc bénéficié de plus de souplesse lors de la création de notre interface que si nous avions utilisé Swing. En contrepartie, le logiciel résultant est plus lourd, car JavaFx n'est plus inclus dans le JDK et suit son propre rythme de dévelopement, il a donc fallu l'inclure dans le jar.

Nous avons aussi utilisé la bibliothèque JFoenix pour utiliser des composants JFX material design, et Ikonli pour pouvoir utiliser les icônes FontAwesome. L'application est ainsi plus moderne et plus facile à utiliser.

3.1.2 Serveur

Nous n'avons pas jugé pertinent l'utilisation d'un serveur HTTP tel qu'un Servlet pour la partie serveur de ce projet. En effet, toutes les communications réseau étaient jusqu'alors réalisées par sérialisation d'objets Java. Il aurait donc fallu écrire des méthodes pour transformer ces objets en JSON, et inversement. De plus, il aurait été complexe de notifier les utilisateurs en HTTP depuis le serveur de présence, car le serveur ne peut que répondre à une requête.

Il nous a donc semblé bien plus simple de faire un serveur en Java utilisant des sockets purs, que ce soit pour la partie présence ou proxy. Ainsi, chaque utilisateur possède une connexion TCP pour le serveur de présence, et une autre pour le proxy, établissant alors une communication duplex avec chaque système.

3.2 Découpage en projets

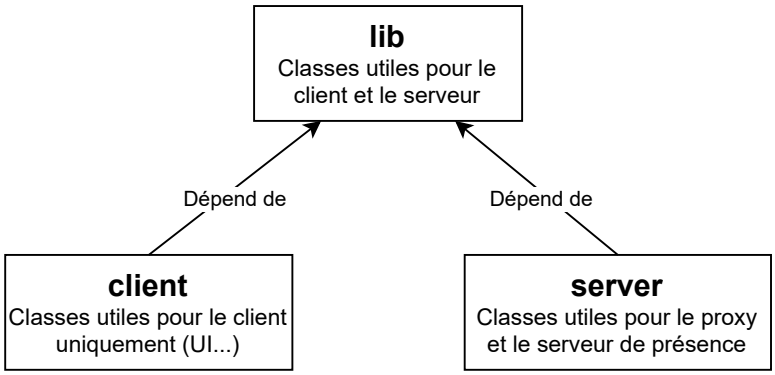


FIGURE 9 – Schéma de dépendance du projet

Afin de réutiliser notre code existant pour le serveur sans pour autant le dupliquer, nous avons créé une bibliothèque contenant toutes les classes utiles à la fois au client et au serveur. Nous avons ensuite ajouté cette bibliothèque en tant que dépendance du client et du serveur (figure 9), grâce à Gradle.

Nous avons ainsi pu utiliser les mêmes classes pour la communication TCP, les utilisateurs et les messages entre le serveur et le logiciel.

Le projet `lib` contient aussi une classe `Log` permettant d'afficher des informations, des warnings et des erreurs dans la console, avec plusieurs niveaux de verbose configurables.

3.3 Communication et identification

Chaque utilisateur est identifié par un UUID, dont l'unicité est quasi-garantie. Il est créé lors du premier démarrage de l'application, puis est stocké dans la base de données. Tous les messages, liste d'utilisateurs, etc. utilisent ces UUID pour désigner les utilisateurs.

Comparé à une identification reposant sur l'adresse IP, il est ici possible de sauvegarder son fichier de base de donnée et de le transmettre sur un autre poste : Nous serons encore considéré comme le même utilisateur par les autres logiciels. Il est aussi possible de communiquer depuis un ordinateur possédant plusieurs cartes réseau.

Réseau Local Le protocole de connexion sur réseau local est relativement simple : chaque utilisateur est connecté à tous les autres (figure 10a). Pour cela, le logiciel envoie un broadcast UDP, auquel les autres machines répondent. Il se connecte ensuite en TCP à ceux qui ont répondu, et les id et les pseudos des deux parties sont échangés.

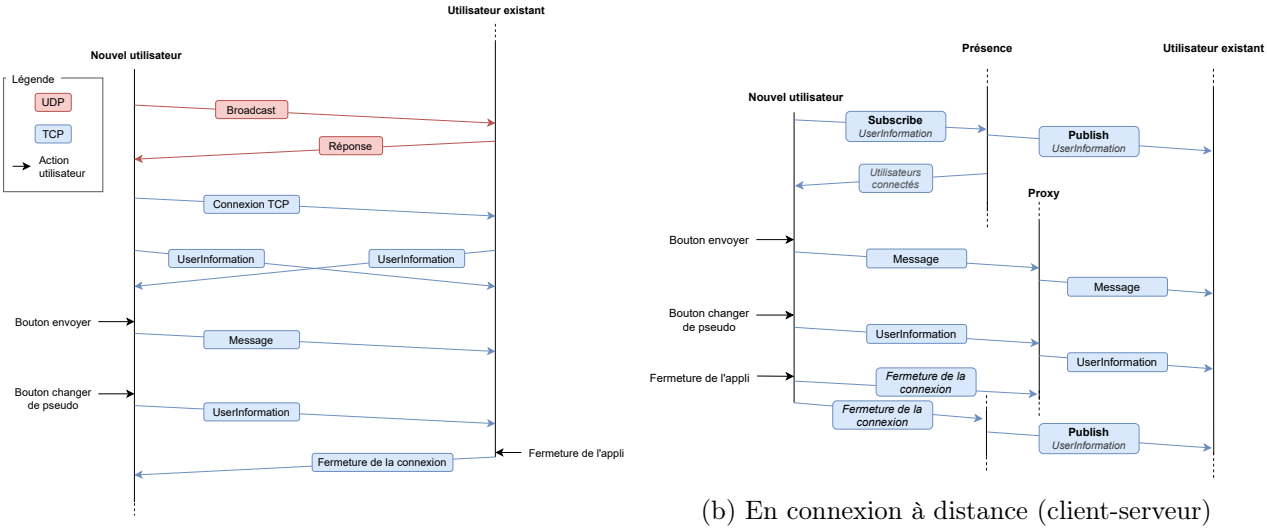
Enfin, des messages peuvent être envoyés jusqu'à ce que la connexion soit fermée, signe de la déconnexion de l'un des utilisateurs. Un message est une classe sérialisée, que le destinataire peut facilement reconstruire. Il est possible d'envoyer un message text, un message avec fichier, ou son profil utilisateur (pour indiquer un changement de nom par exemple).

Serveur de Présence En mode hors réseau local, un serveur offrant deux services sert de relais : le serveur de présence et le proxy (figure 10b).

Le premier sert à stocker la liste des utilisateurs actifs et à notifier ces derniers lors de la connexion, déconnexion ou du changement de nom d'un utilisateur.

Le deuxième, le proxy, s'occupe de transférer les messages à leur destinataire respectif. Le proxy utilisant la même classe de communication TCP que le client, la communication avec proxy ou avec un autre utilisateur fonctionne de la même manière.

Le serveur est caché à l'utilisateur : aucune indication graphique ne se présente à l'utilisateur au sujet de ce serveur. La communication est identique pour l'utilisateur qu'il soit connecté sur un réseau local, au serveur, ou aux deux.



(a) En réseau local (peer-to-peer)

(b) En connexion à distance (client-serveur)

FIGURE 10 – Protocoles réseau dans les deux modes de connexion

3.4 Base de données

Nous utilisons un système de bases de données *sqlite* décentralisé pour stocker les messages et les utilisateurs. Elle est automatiquement créée sur chaque poste au premier démarrage de l'application dans le fichier `clavardator.db`.

Cette base de données est composée de trois tables (voir figure 11) :

— La table `user` stocke les pseudos des utilisateurs avec lesquels on a déjà été connecté au moins une fois.

— La table `message` stocke les messages envoyés et reçus. Elle contient la date d'envoi, l'expéditeur et le destinataire identifiés par leur id, le texte du message, et éventuellement le chemin vers le fichier si le message en contient.

— La table `current_user` stocke le pseudo et l'id de l'utilisateur qui utilise cette instance de Clavardator.

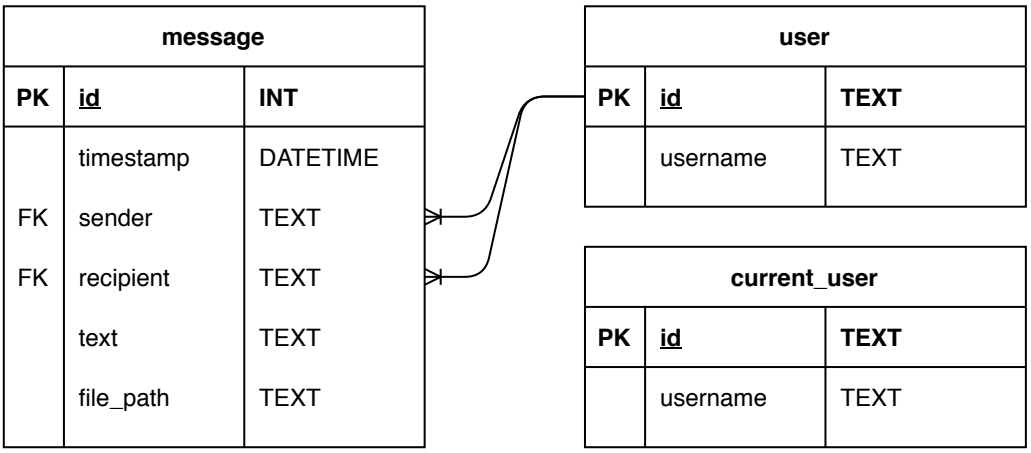


FIGURE 11 – Structure de la base de données

Il est possible de copier le fichier `clavardator.db` et de l'envoyer sur un autre poste pour importer son profil et tous ses messages. Un système de sauvegarde est alors facile à implémenter.

3.5 Architecture MVC

3.5.1 Packages

L'organisation de notre projet est un reflet du design MVC. En effet, nous pouvons distinguer 2 grandes parties pour l'application client :

— Le package `fr.insa.clavardator.client.ui`, qui ne contient que les contrôleurs des vues contenues dans `src/main/resources`.

— Le package `fr.insa.clavardator.client`, qui, si l'on exclut le package précédent, contient seulement le modèle de l'application.

Ainsi, chaque classe du package `ui` est associé à un fichier `fmxl`, et ces classes utilisent celles du modèle pour transmettre les actions de l'utilisateur sur l'interface.

Le logiciel serveur n'acceptant que des paramètres en ligne de commande, il ne possède pas d'interface graphique et un modèle MVC n'était donc pas nécessaire.

3.5.2 Diagramme use case

Après avoir analysé le cahier des charges, nous avons pu isoler les fonctionnalités les plus importantes dans le use case présent sur la figure 12.

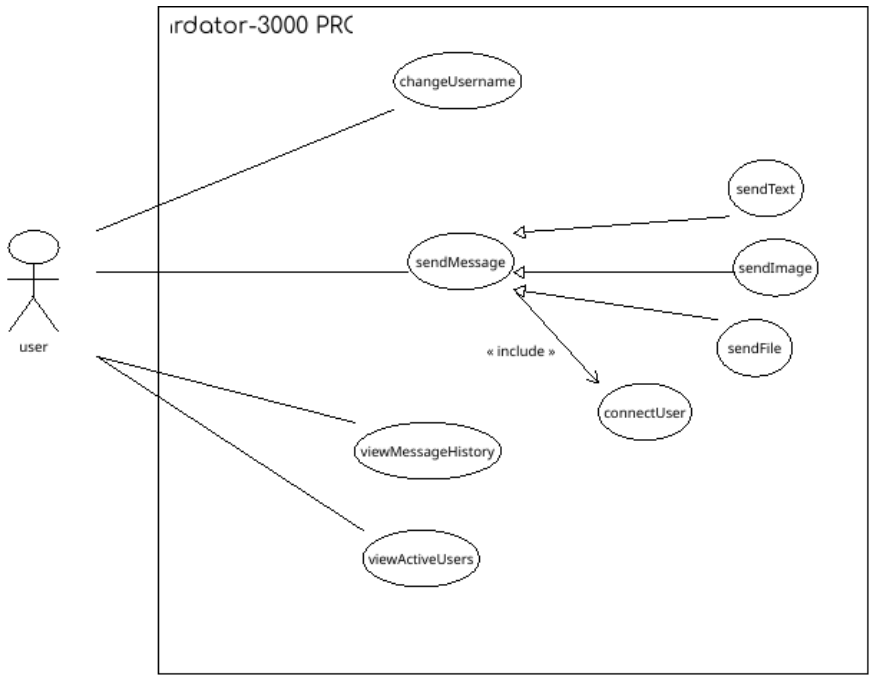


FIGURE 12 – Diagramme use case

3.5.3 Les classes principales du backend

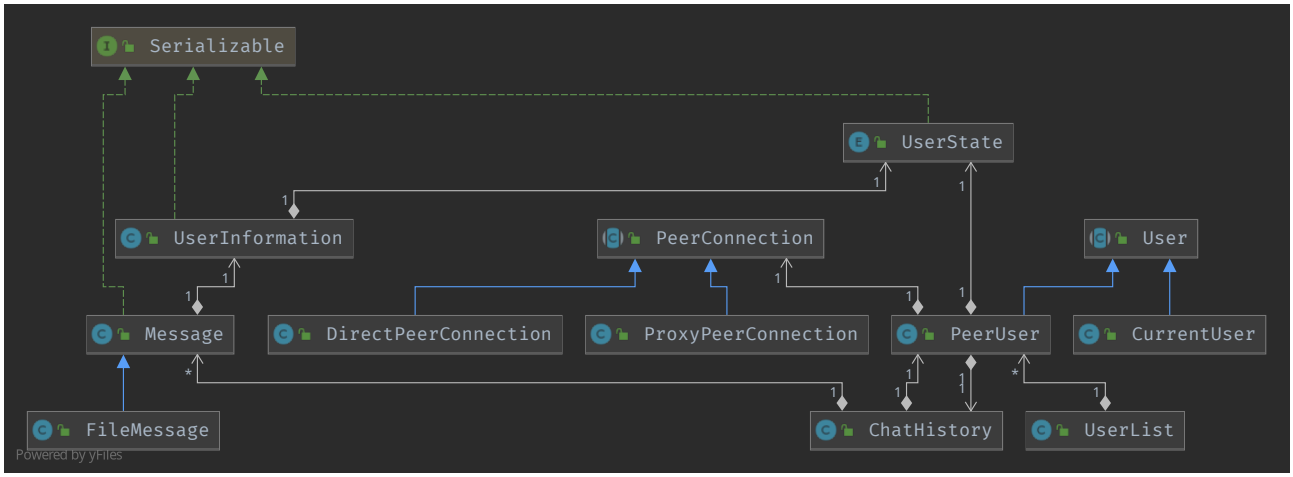


FIGURE 13 – Diagramme UML des principales classes du backend

Les `Users` La classe `User` est une classe abstraite dont héritent `CurrentUser` (qui représente l'utilisateur qui est sur cette machine) et `PeerUser` (qui représentent tous les autres utilisateurs). Ces derniers contiennent un `ChatHistory`, qui gère l'historique de la conversation, ainsi qu'une `PeerConnection`, qui gère la connexion réseau.

Les informations observables des `Users` Les `Users` contiennent un `username` et un `state` qui sont observables grâce à un `PropertyChangeSupport`, l'implémentation java du pattern Observer. L'attribut `state` permet de voir si un utilisateur est actuellement connecté ou non.

Les `PeerConnections` La classe `PeerConnection` est abstraite, car les méthodes pour envoyer et recevoir des messages doivent être différentes dans le cas d'une connexion directe et d'une connexion par proxy.

Les `Serializables` Les classes `Messages`, `UserInformation` et `UserState` implémentent l'interface `Serializable`, car on envoie ces objets directement aux autres utilisateurs pour communiquer un nouveau message, un nouveau nom d'utilisateur ou une déconnexion.

3.5.4 Le serveur

Côté serveur L'architecture du serveur est très simple : une classe Main s'occupe de démarrer le proxy et le serveur de présence, les deux étant par la suite complètement indépendants.

Côté client Pour permettre d'ajouter de nouvelles implémentations de serveur, nous avons opté pour un design pattern Factory. La classe `PresenceFactory` s'occupe de créer l'objet instanciant la classe demandée, suivant le type `ServerType` donné. Cet objet s'occupe de créer à son tour le proxy compatible. Ces classes de serveur de présence et de proxy implémentent respectivement les interfaces `Presence` et `Proxy`, qui sont utilisées dans le reste de l'application.

Ajouter une nouvelle implémentation de serveur est ainsi très simple : il suffit de créer les classes de présence et de proxy, les faire implémenter les bonnes interfaces, et de les ajouter à la factory. Il sera alors possible d'utiliser ce nouveau serveur en changeant le type de serveur dans le fichier config.

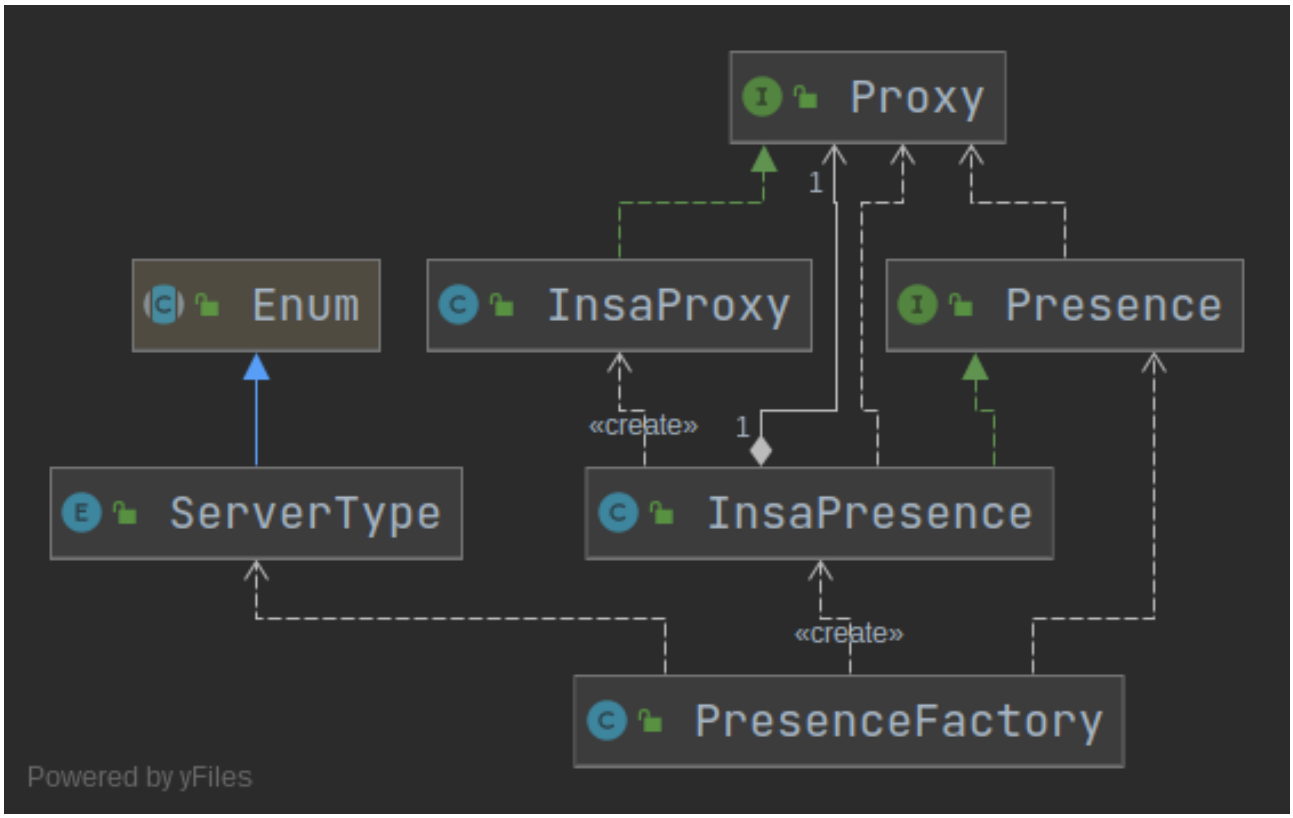


FIGURE 14 – Diagramme UML des classes pour la connexion au serveur

3.5.5 Diagrammes de séquence

Comme le montrent les diagrammes de séquence ci-dessous, chaque action utilisateur est transmise à la classe appropriée du backend, qui s'occupe d'envoyer les données, de les stocker dans la base de données, etc. Une fois toutes ces actions terminées, la valeur d'un attribut observable est modifiée (nom d'utilisateur, liste de messages...). Ceci déclenche la mise à jour de l'interface, car elle est abonnée aux changements de ces attributs.

Les méthodes telles que `send()` ou `discoverActiveUsers()` prennent en argument deux fonctions de callback, appelées en cas de succès ou d'échec. Nous ne les avons pas représentées sur les diagrammes pour éviter de surcharger ces derniers.

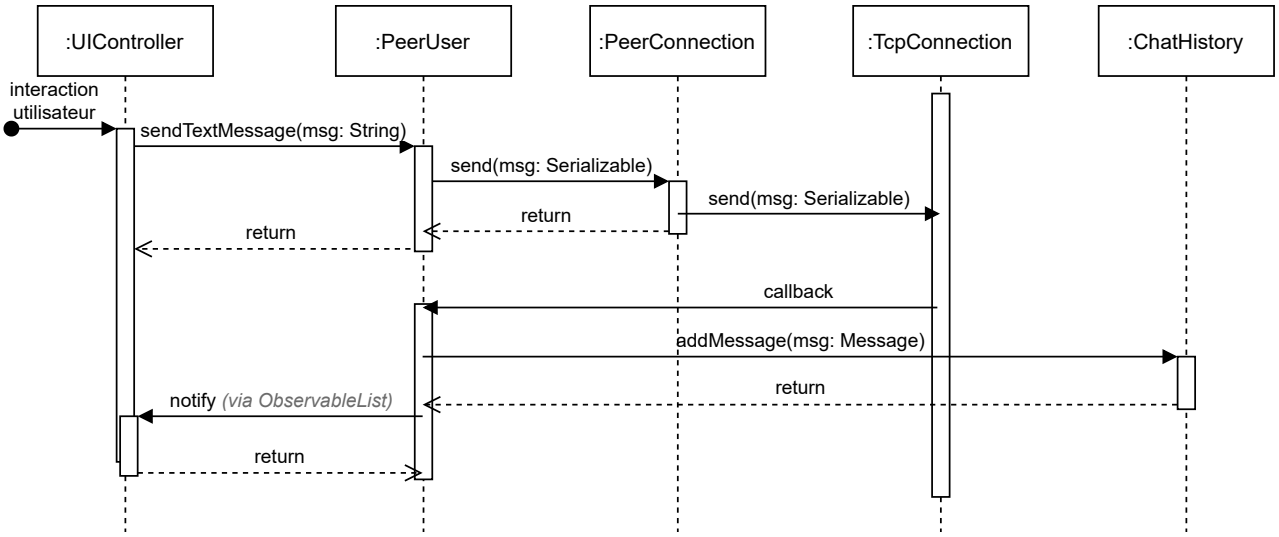


FIGURE 15 – Diagramme de séquence de l'envoi d'un message

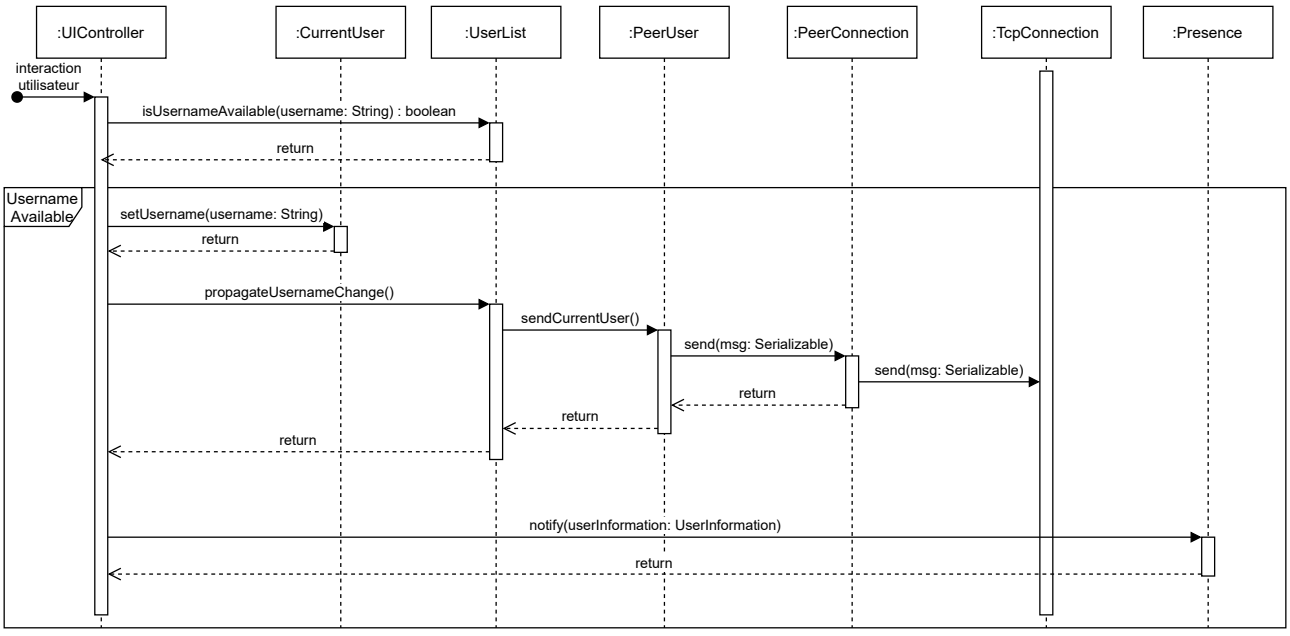


FIGURE 16 – Diagramme de séquence du changement de nom

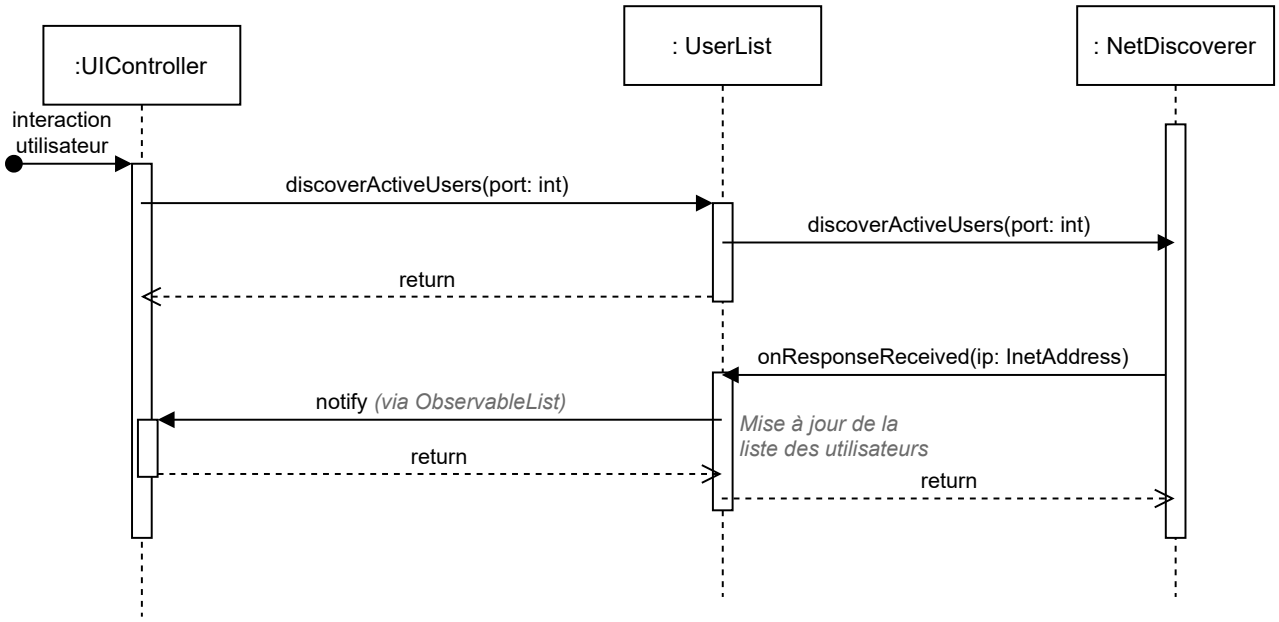


FIGURE 17 – Diagramme de séquence de la découverte des utilisateurs actifs

3.5.6 Frontend

Chaque classe du package `fr.insa.clavardator.client.ui` est un contrôleur d'interface. L'interface est définie dans plusieurs fichiers `fxml`. À chacun de ces fichiers `fxml` est associé un contrôleur, portant le même nom avec le mot `Controller` ajouté à la fin. Par exemple, le controleur du fichier `errorScreen.fxml` est `ErrorScreenController.java`.

Pour améliorer la modularité de l'interface, nous avons isolé chaque composant d'interface dans son propre couple FXML/Controller. que nous combinons pour créer des écrans complexes. Nous avons aussi regroupé les classes manipulant les mêmes éléments dans des packages pour simplifier l'organisation. Nous avons ainsi 3 sous packages interne à l'interface :

- `chat` pour tout ce qui touche à l'affichage de la fenêtre de messages.
- `dialogs` pour la manipulation de popups.
- `users` pour l'affichage de la liste des utilisateurs

chat Ce package permet de gérer la fenêtre de chat. Comme nous pouvons le voir dans la figure 18, le fonctionnement général est géré par la classe `ChatController`, qui délègue le fonctionnement précis de chaque composant (entête, pied de page, item de liste) au contrôleur associé.

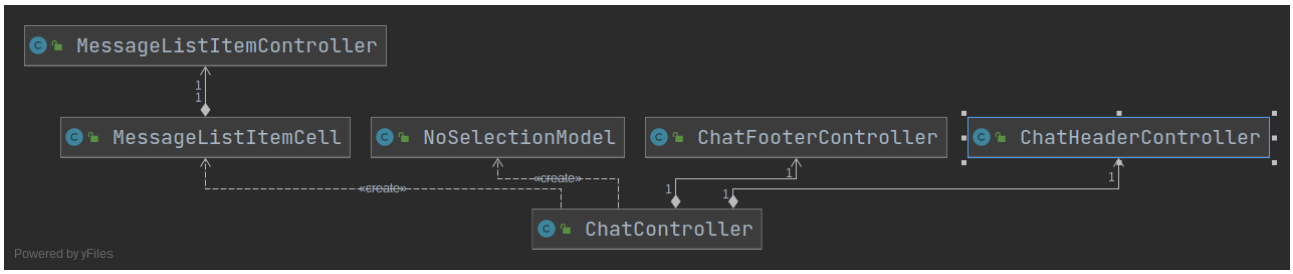


FIGURE 18 – Package `chat`

dialogs Ce package regroupe les classes permettant la gestion de popup par dessus la fenêtre principale. Toutes ces classes sont indépendantes, et ne dépendant pas d'autres éléments de l'interface.

users Ce package est responsable de la gestion de l'affichage des utilisateurs. Comme pour le package `chat`, une classe (`UserListController`) s'occupe de déléguer les tâches aux bons contrôleurs.

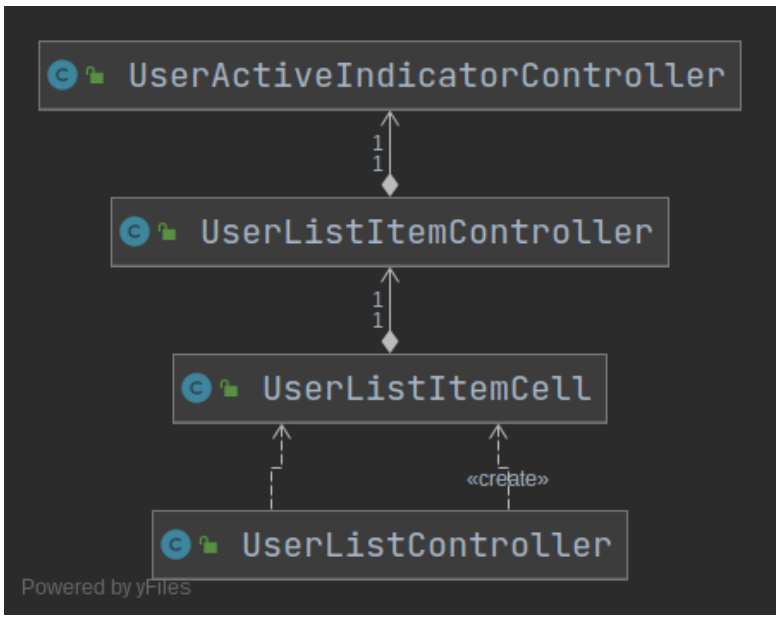


FIGURE 19 – Package `users`

Si nous prenons un peu plus de recul et que nous regardons le package `ui`, nous pouvons voir que la classe `MainController` est le contrôleur principal, s'occupant de gérer toute l'interface à l'aide des packages vus précédemment et de contrôleurs pour un écran d'erreur, la barre d'outils, et un écran de chargement.

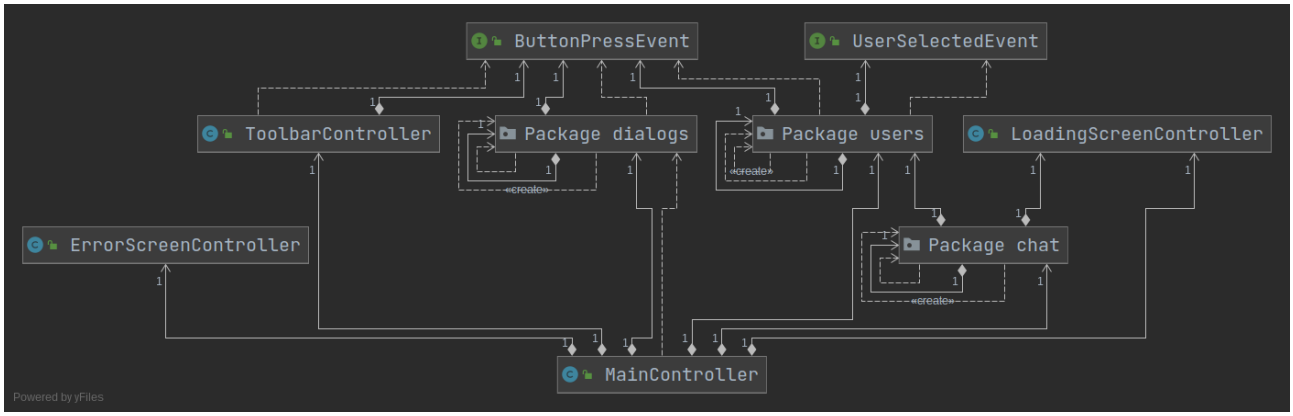


FIGURE 20 – Package `ui`

4 Tests et validation

4.1 Tests unitaires

Nous avons créé des tests unitaires pour les fonctions liées à la base de données. Ces tests étaient assez simples à écrire, et nous ont permis de débugguer plus rapidement les requêtes SQL que nous avions écrites.

Le reste du code était plus complexe à tester de cette manière car il reposait sur l'interface graphique ou le réseau.

4.2 Simulations

Nous ne pouvions pas tester toutes les fonctionnalités de l'application en mode peer-to-peer sur un même ordinateur, car les deux instances auraient essayé d'écouter sur les mêmes ports. Nous avons

donc seulement pu vérifier que l'application se lançait bien, que les broadcast étaient bien envoyés et reçus grâce à Wireshark, etc.

En mode client-serveur cependant, les tests étaient possibles sur un même ordinateur. Nous avons testé la connexion et la déconnexion, l'envoi de messages, l'envoi de fichiers, le changement de pseudo, et l'erreur lors de la connexion si le pseudo est déjà pris.

4.3 Conditions réelles

Nous avons aussi eu l'occasion de tester le programme en conditions réelles, avec plusieurs machines connectées sur un même réseau local. Nous avons alors effectué en peer-to-peer les mêmes tests que pour le mode client-serveur.

Nous avons aussi testé les deux modes à la fois, avec certains ordinateurs connectés sur le même réseau local et d'autres connectés depuis un autre réseau à travers le serveur.

Ces tests nous ont permis de corriger quelques bugs, et à présent tout a l'air de fonctionner.

Cependant, pour des raisons techniques, nous n'avons pas eu l'occasion de tester sur d'autres plateformes que Linux (Ubuntu 20.04). Théoriquement tout devrait fonctionner de la même manière, mais si ce n'est pas le cas, merci d'essayer sous Ubuntu.

Conclusion

Clavardator est ainsi un logiciel de discussion instantané respectant le cahier des charges fourni. Le système est capable d'authentifier des utilisateurs avec un identifiant et un nom unique, de découvrir et de dialoguer avec des utilisateurs internes ou externes au réseau, et d'afficher l'historique des conversations.

Grâce aux technologies choisies, le logiciel est ergonomique et simple d'utilisation. Le déploiement se fait en quelques minutes et peut être facilement intégré à des Pipelines de CI/CD (Continuous Integration / Continuous Delivery) comme Gitlab, Github ou Jenkin.

Le code utilise des patterns de conception connus, comme MVC, Observeurs, ou encore Factory, et il est facile d'ajouter de nouvelles implémentations de serveur.

INSA Toulouse
135, Avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Université Fédérale Toulouse Midi-Pyrénées

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE