

Rapport Projet Clavardage

Programmation Orientée Objet

25 janvier 2022

SIXT Romain
PASTOR Emmanuel

Dépôt Git du projet :

<https://github.com/emmanuel-pastor/chat-system>

Gestion de projet Jira :

<https://simple-smart-apps.atlassian.net/jira/software/projects/CS/boards/1/backlog>

Introduction

Le projet Chat-System a pour but de créer une **application de clavardage** qui sera déployée sur des **ordinateurs d'un même réseau local**. Cela permettrait à deux utilisateurs connectés en même temps de pouvoir échanger des messages textuels. Afin de créer cette application, nous devons passer par les indispensables phases d'analyse du cahier des charges et de conception. La phase de conception et l'élaboration des divers diagrammes a plusieurs vocations. La première est de nous assurer d'avoir saisi toutes les subtilités du cahier des charges. A savoir toutefois que les diagrammes ci-dessous se concentrent uniquement sur l'architecture peer-to-peer de l'application.

C'est-à-dire celle ne **considérant qu'un réseau local** et pas d'utilisateurs distants devant passer par un serveur de présence. La deuxième vocation de ces diagrammes est bien entendu de nous permettre de poser les premières bases de notre conception. Une fois les diagrammes bien établis, nous aurons toutes les cartes en main pour mettre en place le squelette de l'application et les principales classes du projet. Ce n'est qu'une fois tout cela effectué que l'implémentation peut être commencée de manière efficace.

Dans un premier temps nous détaillerons la **conception de notre application**. Dans un second temps nous détaillerons les procédures utilisées pour réaliser notre application. Dans un troisième temps, nous dresserons un manuel utilisateur pour permettre à toute personne d'utiliser correctement toutes les fonctionnalités de l'application.

Concernant la **gestion de notre projet**, nous avons utilisé **Jira** pour gérer notre progression dans le temps et nous répartir les tâches sous forme de sprints et **Git et Github** pour pouvoir implémenter de nouvelles fonctionnalités de manière parallèle de façon collaborative. Aussi, pour développer le projet, nous avons utilisé l'**IDE IntelliJ IDEA**, avec **Gradle** pour la gestion des dépendances, car nous étions déjà familiers avec ces outils là.

I. Conception du projet

Les diagrammes sont également disponibles dans le repository github dans le dossier Modeling pour une meilleure lisibilité

I.1 Diagrammes de cas d'utilisation

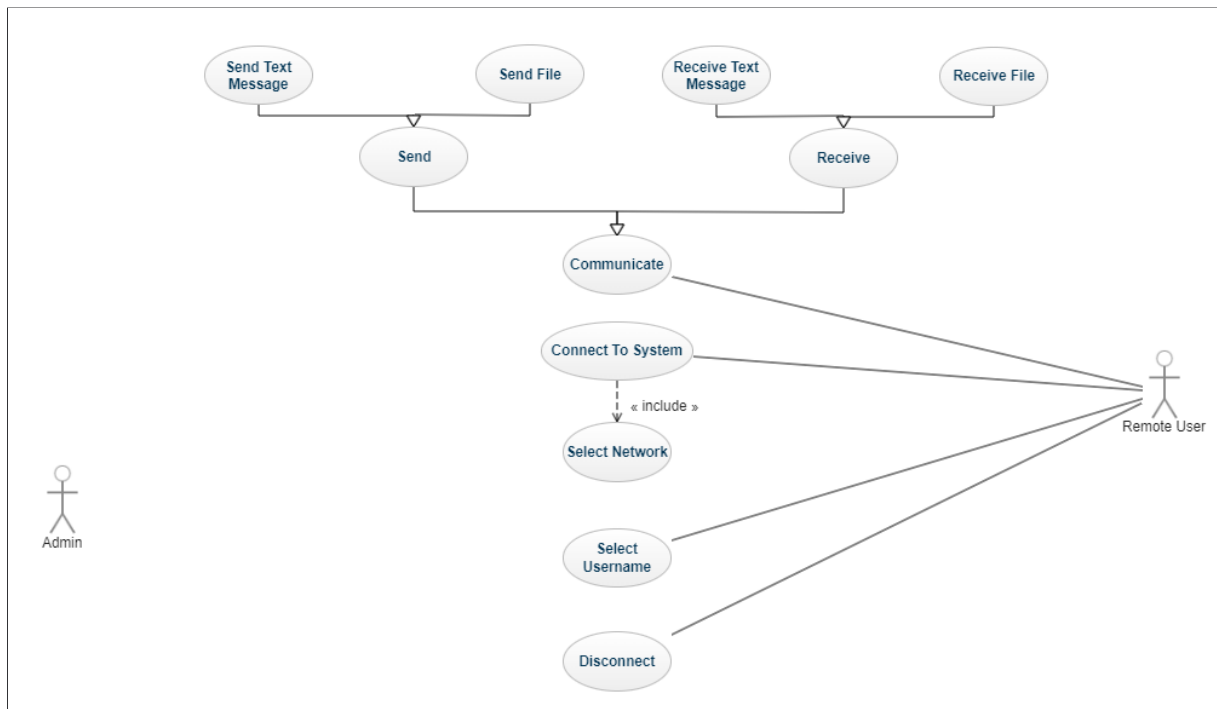


Figure 1 - Diagrammes de cas d'utilisation de Chat System

I.2 Diagrammes de classe

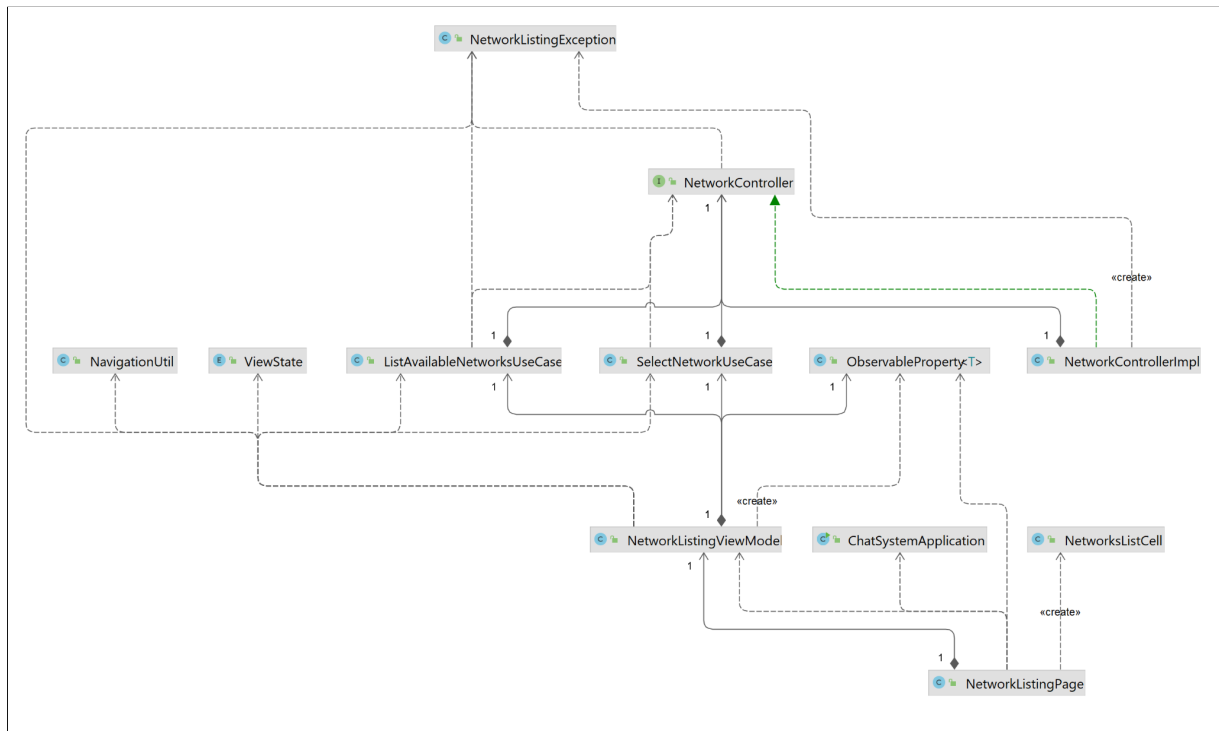


Figure 2 - Diagramme de classe : fonctionnalité de sélection du réseau

Le diagramme de classe complet est disponible sur le repository github et ne serait pas lisible sur le rapport. Chaque fonctionnalité a un diagramme de classe similaire à celui ci-dessus étant donné que chaque fonctionnalité suit l'architecture que l'on s'est fixée.

I.3 Diagrammes de séquences

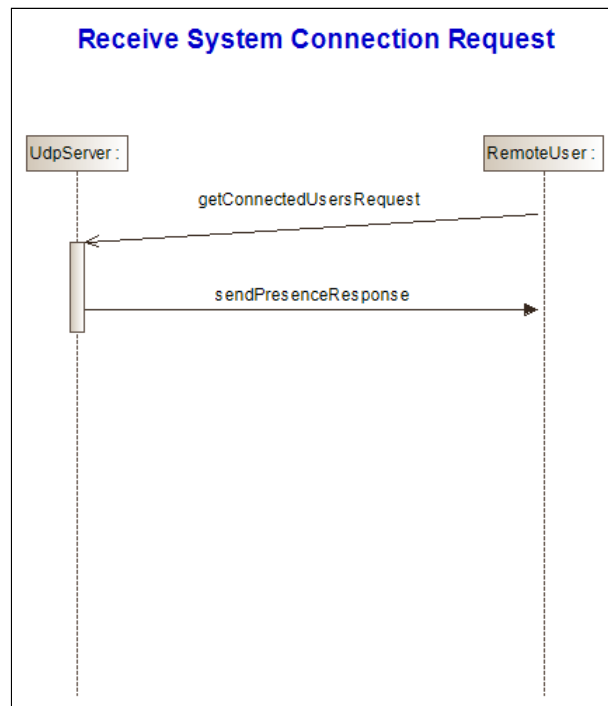


Figure 3 - Diagramme de séquence : réception d'une requête de connexion au système

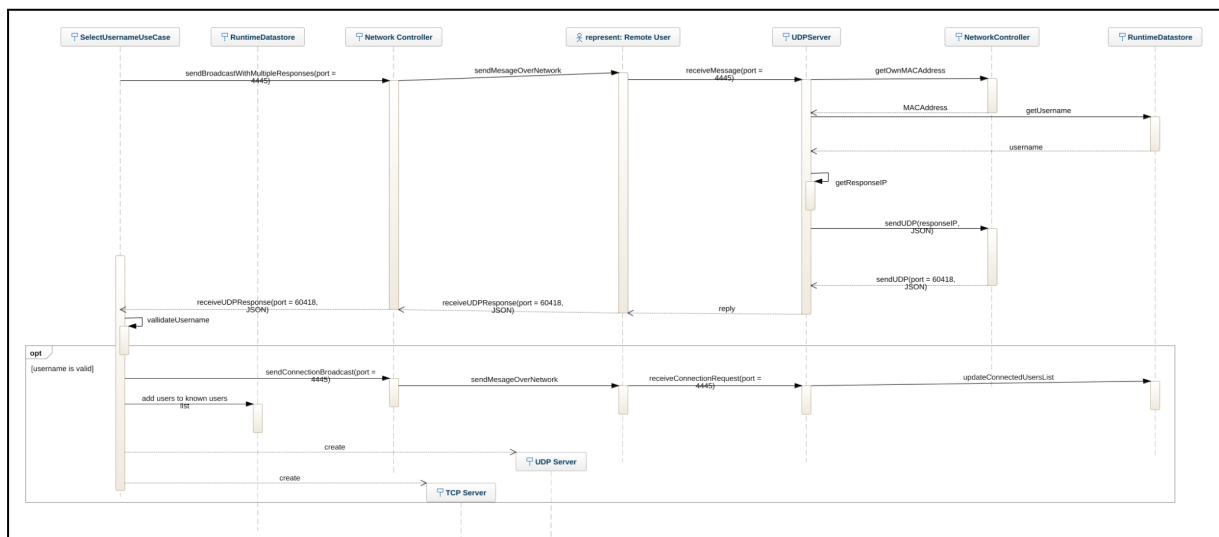


Figure 4 - Diagramme de séquence : validation du pseudonyme et connexion au système

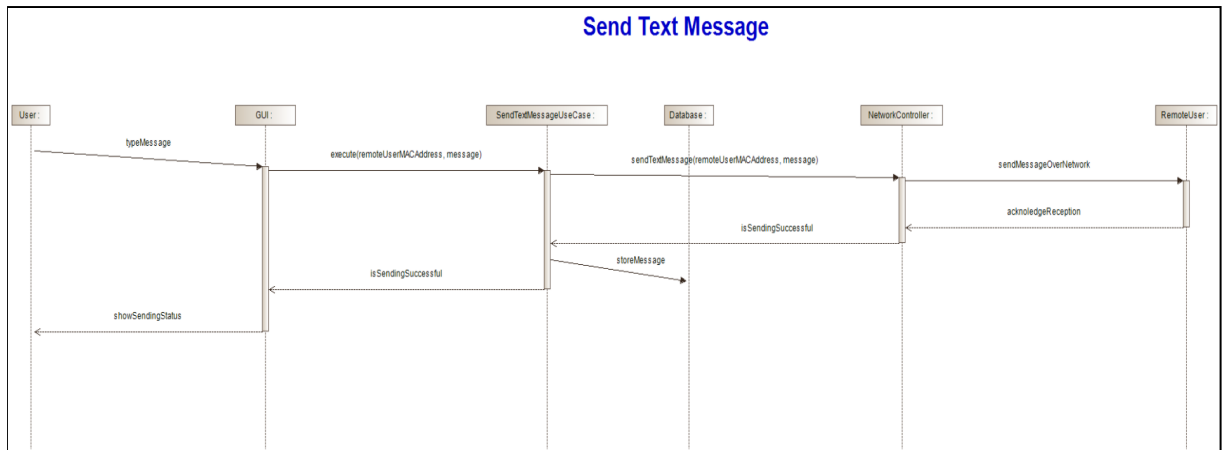


Figure 5 - Diagramme de séquence : envois de message

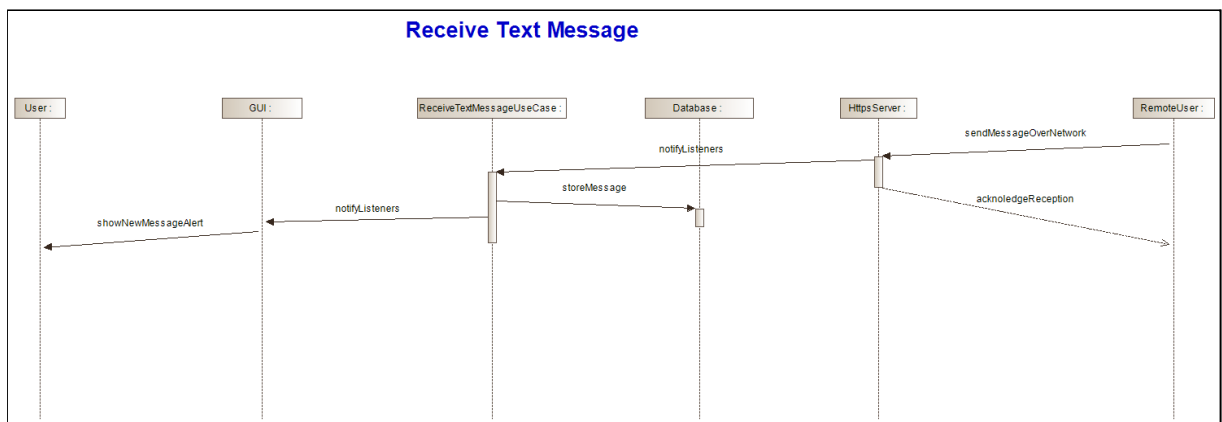


Figure 6 - Diagramme de séquence : réception de message

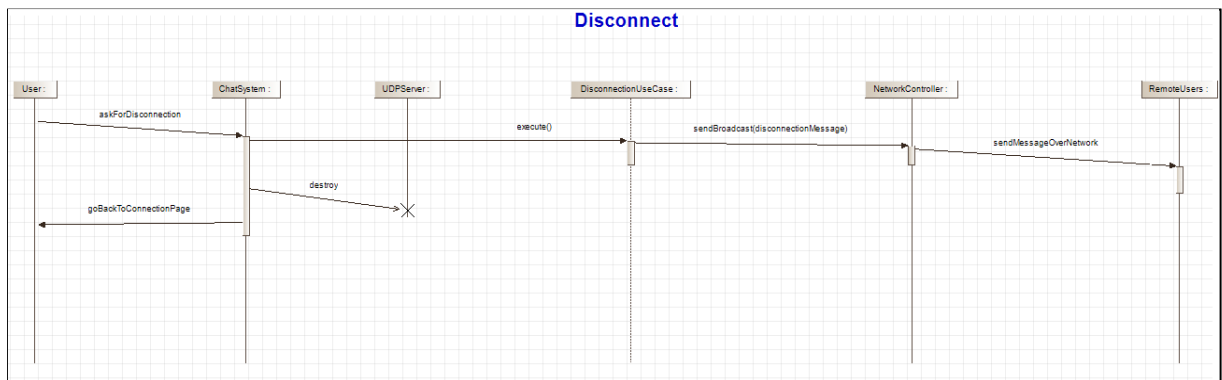


Figure 7 - Diagramme de séquence : déconnexion de l'utilisateur

II. Choix d'implémentation du projet

II.1 Intégration continue et gestion de projet

À propos de la gestion de projet nous avons choisi d'utiliser **Jira** afin de structurer l'avancement du projet sous forme de **sprints**. Chaque sprint dure une semaine et nous avons convenu que chacun devait apporter une nouvelle fonctionnalité (feature) à notre projet: récupération des interfaces réseaux, choix du pseudonyme, connexion, enregistrement dans la base de donnée, envois et écoute des messages.

En parallèle nous avons utilisé **Git** pour la gestion des versions de notre projet et l'ajout de fonctionnalités pas à pas pour chaque sprint. Ainsi nous avons créé une **branche dev** contenant tous les commits de fonctionnalités achevés à chaque fin de sprint et une **branche sprint** (ex: CS-18_Connect_to_a_connected_user) correspondant à notre branche de travail sur le sprint associé à la fonctionnalité souhaitée. Une fois le sprint fini et la fonctionnalité développée, nous faisons un merge de la branche sprint dans la branche dev et nous supprimons la branche sprint. La branche master sert à faire apparaître seulement les versions majeures du logiciel.

De plus, nous avons mis en place **Jenkins** pour toute la partie **automatisation des tests** et des builds de notre projet sachant que nous avons utilisé **Gradle** comme build tool. Nous avons relié Jenkins à notre dépôt Git afin qu'à chaque push effectué sur le dépôt, le lancement des tests et des builds se fasse automatiquement.

II.2 Architecture

Pour l'implémentation de notre projet de système de clavardage nous avons opté pour une architecture appelée **Clean Architecture**.

Cette architecture identifie trois couches distinctes :

- **La couche données**: On y retrouve les sources de données locales ou distantes et les classes permettant d'accéder à ces données.
- **La couche domaine**: Cette couche contient les use case et tout traitement qui peut être fait sur les données. Chaque use case possède sa propre classe en suivant le Command Pattern.
- **La couche présentation** : C'est ici que l'on s'occupe d'afficher les données à l'écran. Pour chaque page de l'application, on y retrouve deux classes: la page elle-même (View) et le "contrôleur" associé à la page (ViewModel). Le viewmodel

s'occupe d'exécuter les use case appropriés selon les évènements qui lui sont remontés par la vue et de modifier ses états internes qui sont mis à disposition de la vue. La vue observe l'état du viewmodel pour mettre à jour son affichage en fonction et lui remonte les évènements qui surviennent (clics, raccourcis clavier etc...).

II.3 Dependency Injection

Nous avons appliqué le principe de Dependency Injection (DI) dans notre projet afin de nous faciliter les tests et le développement. En effet, grâce à la DI et au polymorphisme, nous pouvons très facilement remplacer nos sources de données par des sources de test lors de nos tests. Nous avons utilisé le framework **Guice** qui est très simple d'utilisation. Il nous a permis de facilement injecter toutes les dépendances parfois nombreuses.

II.4 Base de donnée et stockage des messages

Nous utilisons une **base de données SQLite** afin de stocker les messages envoyés entre les utilisateurs. La base de données est décentralisée, contenue dans un simple fichier local et propre à l'application. Nous avons choisi SQLite pour sa facilité d'installation, d'utilisation, d'accès (pas de dépendance à un serveur), sa portabilité et sa rapidité.

II.5 Interface graphique utilisateur (GUI)

Le choix de l'interface graphique s'est porté sur la bibliothèque **JavaFX** que nous maîtrisons plutôt que Swing pour pallier les défauts de ce dernier et profiter du logiciel **SceneBuilder** pour concevoir nos interfaces rapidement.

II.6 Observation des données

Afin de nous faciliter la tâche, nous avons créé une classe **ObservableProperty** qui nous permet d'observer n'importe quel objet de manière thread safe. Cela rend la tâche plus facile lorsque l'on souhaite observer les données du viewmodel depuis la vue. Cela nous évite de devoir implémenter la classe **Observable** de Java pour chaque objet que l'on souhaite observer et permet à plusieurs classes d'observer le même objet.

II.7 Connexion et récupération des utilisateurs connectés

La demande de connexion et la récupération des utilisateurs connectés se fait lors de la validation du choix de pseudonyme dans la fenêtre associée. Nous avons opté pour un

envoi de messages sous le format JSON qui permet de transmettre plusieurs informations : choix du pseudonyme, et l'adresse MAC source au sein d'un seul message, facile à manipuler. L'**envoi des données de connexion** se fait en **mode broadcast** pour une question de facilité d'envoi sachant que les Qualité de Service (QoS) demandés ne concernent pas l'ordonnancement. La réception du côté serveur (utilisateur distant) d'un fichier de connexion au format JSON est ainsi traité et une réponse au format JSON est envoyée à l'utilisateur initial en **mode UDP** afin de l'informer sur le pseudonyme de l'utilisateur distant. De ce fait, l'utilisateur initial reçoit un fichier de connexion JSON de tous les utilisateurs présents ainsi que leur pseudonyme, il est alors facile de faire la liste des utilisateurs connectés et de vérifier que le nom d'utilisateur choisi est disponible.

II.8 Envois de messages entre utilisateurs

L'envoi de message se fait par **utilisation du protocole TCP** afin d'assurer au minimum ordre et fiabilité des paquets. Une phase de connexion est d'abord établie entre les utilisateurs connectés. Une fois effectuée, l'envoi de messages peut se faire point à point. Chaque utilisateur pouvant recevoir plusieurs messages venant d'utilisateurs différents à n'importe quel moment. Nous avons aussi choisi de laisser les connexions ouvertes lorsque l'on change d'utilisateurs avec lequel on souhaite communiquer, cela nous permet de pouvoir continuer à recevoir des messages des autres utilisateurs connectés.

II.9 Re-synchronisation de l'application avec l'état du réseau

Nous avons conçu le système de sorte à ce que si un envoi de paquet UDP se perd (lors de la connexion ou de la déconnexion), l'application continue à marcher correctement et l'état de connexion des utilisateurs reste correct. Par exemple, si un utilisateur ferme la fenêtre de l'application sans avoir cliqué sur "Disconnect", tous les utilisateurs qui étaient en discussion avec lui sont immédiatement informés. Autre exemple, si un premier utilisateur voit qu'un second est connecté sur le réseau mais pas l'inverse, alors ce premier utilisateur peut envoyer un message au second, et ce message apparaîtra bien chez le deuxième utilisateur. Pour le deuxième utilisateur, son correspondant sera désormais marqué comme connecté et il pourra lui envoyer des messages.

II.10 Test du projet

Nous avons testé notre projet à plusieurs niveaux. Tout d'abord, nous avons fait quelques tests unitaires, mais trop peu par manque de temps. Nous avons principalement fait des tests end-to-end de l'application en la faisant fonctionner sur

des machines Linux et Windows et en effectuant différents scénarios utilisateur. De plus, le covid nous ayant empêché de tester ensemble sur la fin, Emmanuel a simulé le serveur TCP et UDP dans un programme python afin de le faire fonctionner sur un Raspberry Pi et de pouvoir continuer à tester l'application avec un utilisateur distant.

III. Manuel Utilisateur

III.1 Routine de lancement de l'application

- ❖ Accédez au projet (lien page 1).
- ❖ Se positionner sur la branche **master** (par défaut).
- ❖ Téléchargez le projet

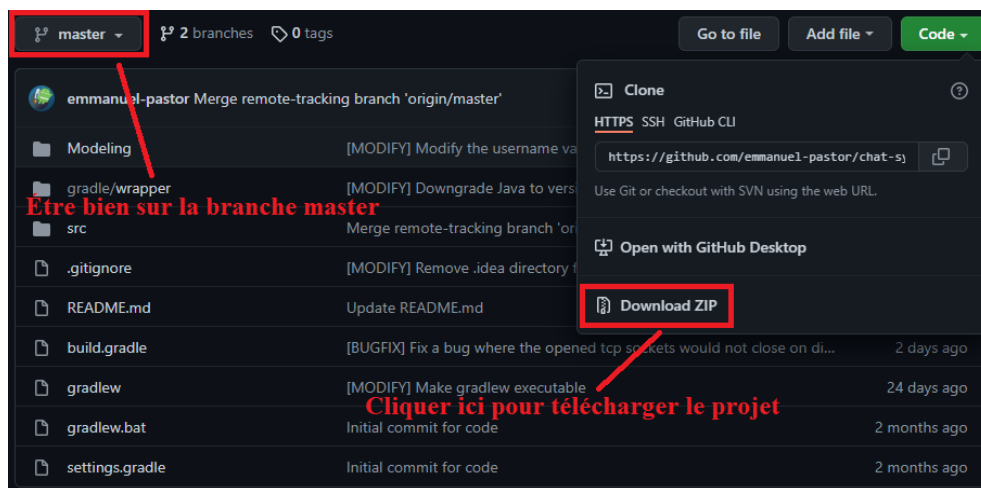
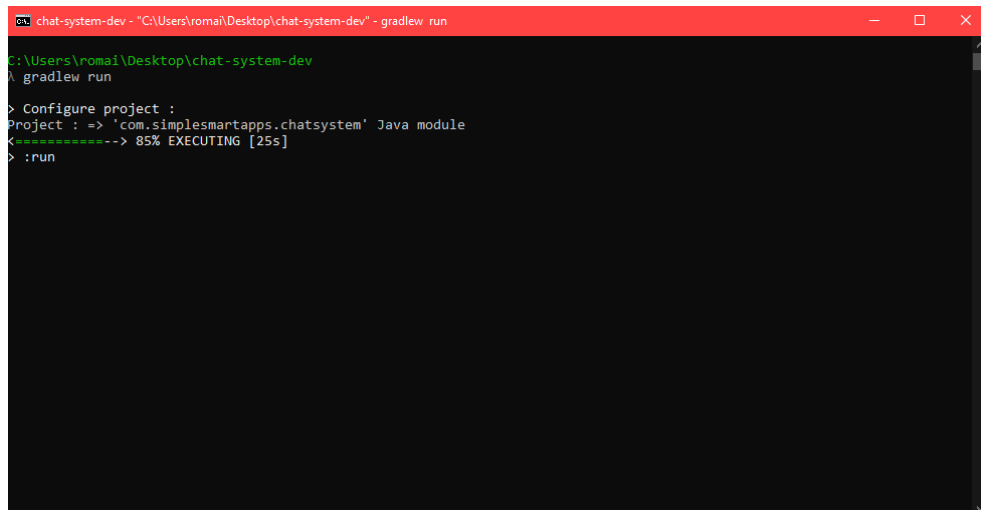


Figure 8 - Téléchargement du projet sur la branche dev

- ❖ Décompressez le dossier zip
- ❖ Téléchargez et installez le JDK Java, version 11. (Faire attention à ce que la variable d'environnement système JAVA_HOME soit configurée sur cette version de Java).
- ❖ Dans un terminal ouvert dans le répertoire racine du projet :
 - sous Linux, tapez : `./gradlew run`
 - sous Windows, tapez : `gradlew run`
 - sous Mac OS, tapez: `./gradlew run` (logiciel non testé sous Mac OS)



```
chat-system-dev - "C:\Users\romai\Desktop\chat-system-dev" - gradlew run
C:\Users\romai\Desktop\chat-system-dev
\ gradlew run
> Configure project :
Project : => 'com.simplesmartapps.chatsystem' Java module
<-----> 85% EXECUTING [25s]
> :run
```

Figure 9 - Lancement du projet par CLI (command line interface)

L'application s'ouvre après quelques secondes, après compilation sous Gradle.

III.2 Choix de l'interface réseau

Au lancement de l'application, une fenêtre renseignant la liste des interfaces réseau disponible s'affiche. Il s'agit des interfaces réseau actuellement connectées, que votre ordinateur possède (cartes réseaux). Il vous est demandé de cliquer (clic gauche de la souris) sur l'interface correspondant au réseau auquel vous souhaitez vous connecter dans la liste. Un choix d'interface réseau différent implique un réseau de communication différent. Un bouton situé au-dessus de la liste est présent afin de rafraîchir la liste des interfaces disponibles.

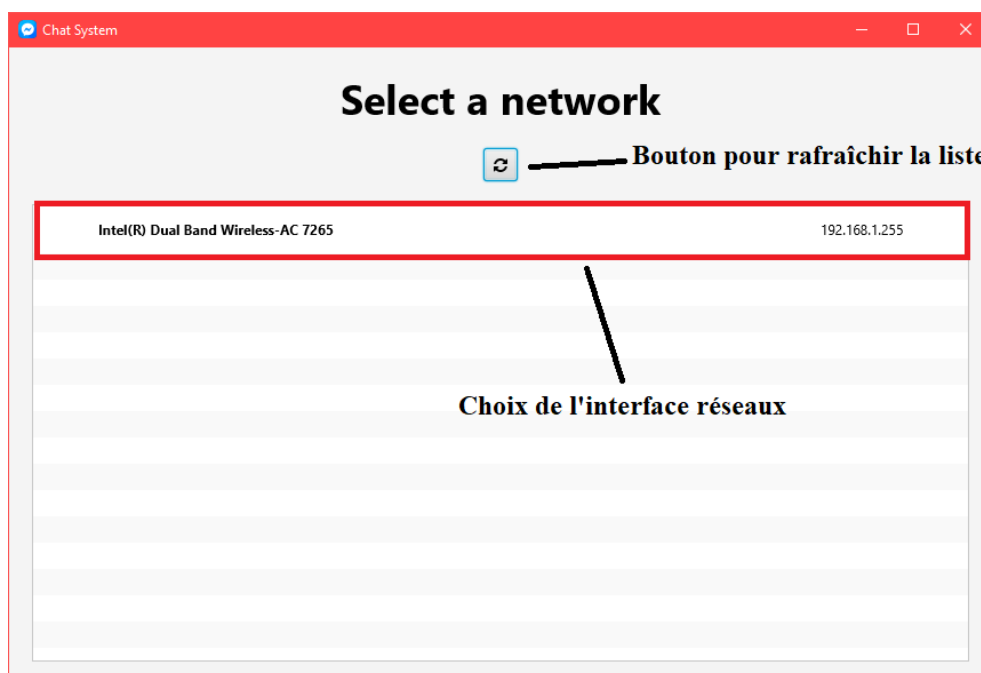


Figure 10 - Écran de sélection de l'interface réseau

III.3 Choix du pseudonyme et connexion

Une fois l'interface réseau choisie, vous serez redirigé vers la fenêtre de choix du pseudonyme et de connexion. Il vous est demandé de rentrer votre pseudonyme afin de finaliser votre connexion. Pour le valider il vous suffit d'appuyer sur le bouton "Submit" ou bien d'utiliser le raccourci clavier "Entrée". Au bout d'un laps de temps défini à 1 seconde vous serez redirigé vers la page de messagerie principale si votre pseudonyme n'est pas déjà utilisé sur le réseau. Il est possible de revenir à la sélection des interfaces réseaux en appuyant sur le bouton en haut à gauche de l'application. Le pseudonyme doit contenir **moins de 20 caractères** pour être acceptable.

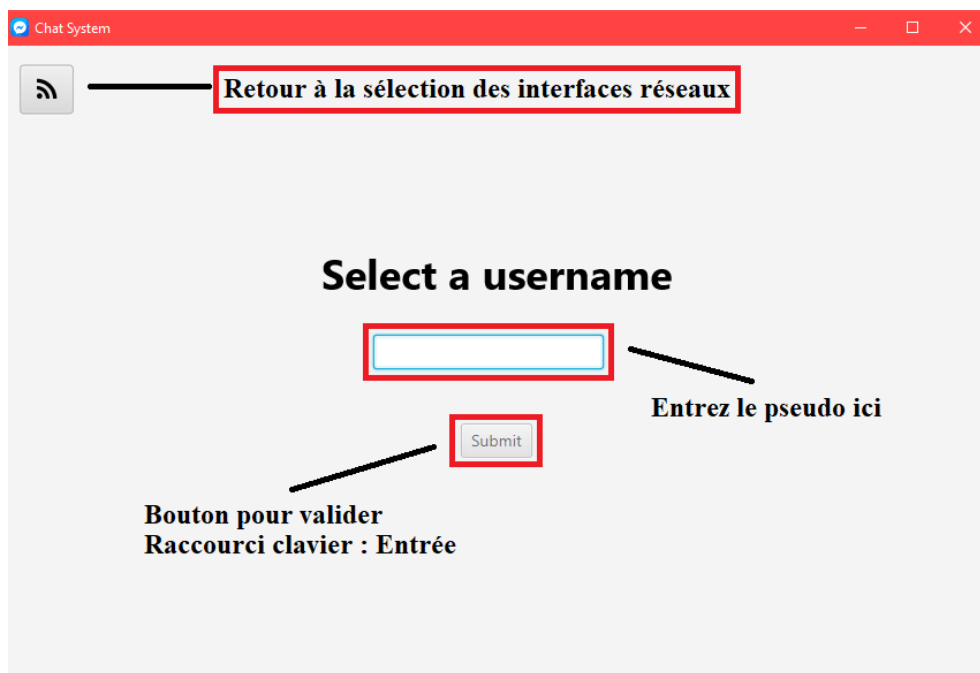


Figure 11 - Écran de choix de pseudonyme et connexion

Dans le cas où le pseudonyme est déjà utilisé, vous recevrez un message d'erreur. Celui-ci indiquera que le pseudonyme est déjà utilisé. Vous aurez la possibilité de rentrer un nouveau pseudonyme par la suite.

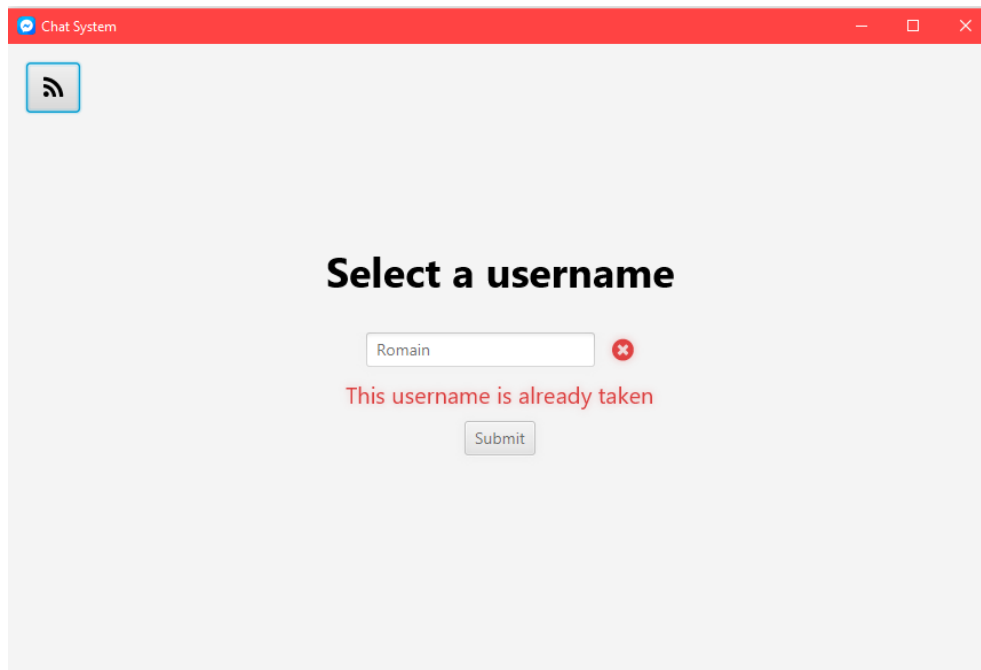


Figure 12 - Écran d'erreur relatif au pseudonyme

III.4 Écran principal

Une fois le pseudonyme et la connexion validée vous serez redirigée sur l'écran du menu principal de l'application. Un bouton rouge "Disconnect" est présent en haut à gauche afin de vous déconnecter de la session à n'importe quel moment.

Votre pseudonyme est aussi renseigné et il est possible de le modifier à tout moment en cliquant sur le bouton à droite de votre pseudonyme. Il vous sera demandé d'entrer votre nouveau pseudonyme ou d'annuler la saisie en appuyant sur le raccourci clavier "Echap".

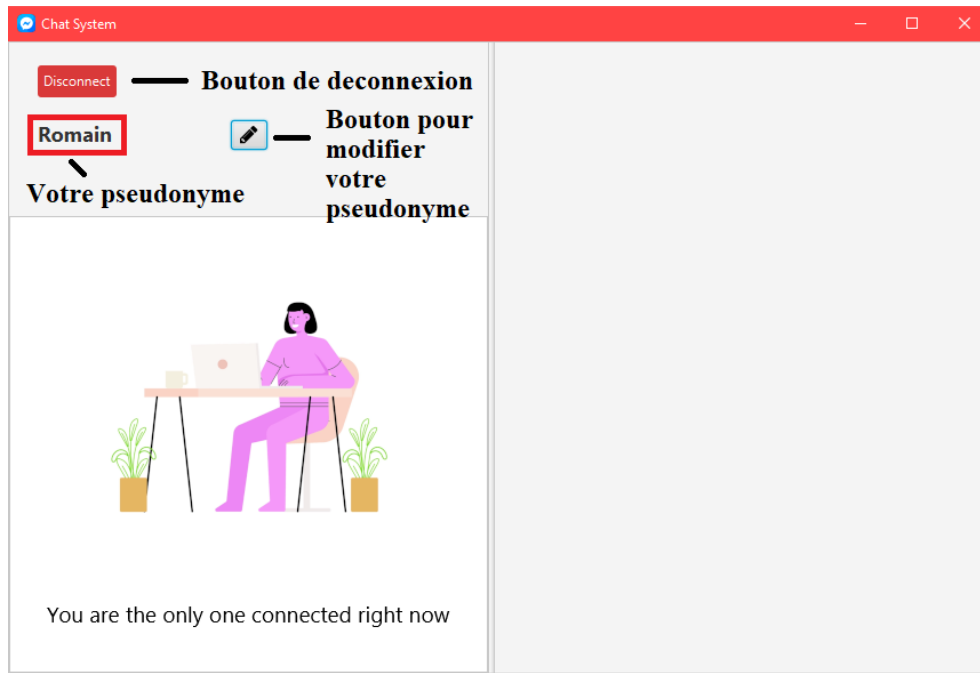


Figure 13 - Écran principal

III.4 Conversation et historique des messages

La liste des personnes ayant été connectées à l'application sur le réseau sont disponibles dans la liste en dessous et le panneau de droite permet de renseigner la conversation existante avec la personne sélectionnée dans la liste précédente. Si vous êtes la seule personne connectée sur le réseau cette liste sera vide et le panneau de droite sera alors vide.

Lorsqu'un utilisateur est connecté à l'application, il est alors possible de démarrer ou reprendre une conversation. Pour cela il suffit de cliquer sur la personne souhaitée dans la liste des utilisateurs sur le panneau de gauche de l'application et le panneau de droite affichera automatiquement la conversation avec les messages précédents ou une conversation vierge si aucun message n'a déjà été envoyé.

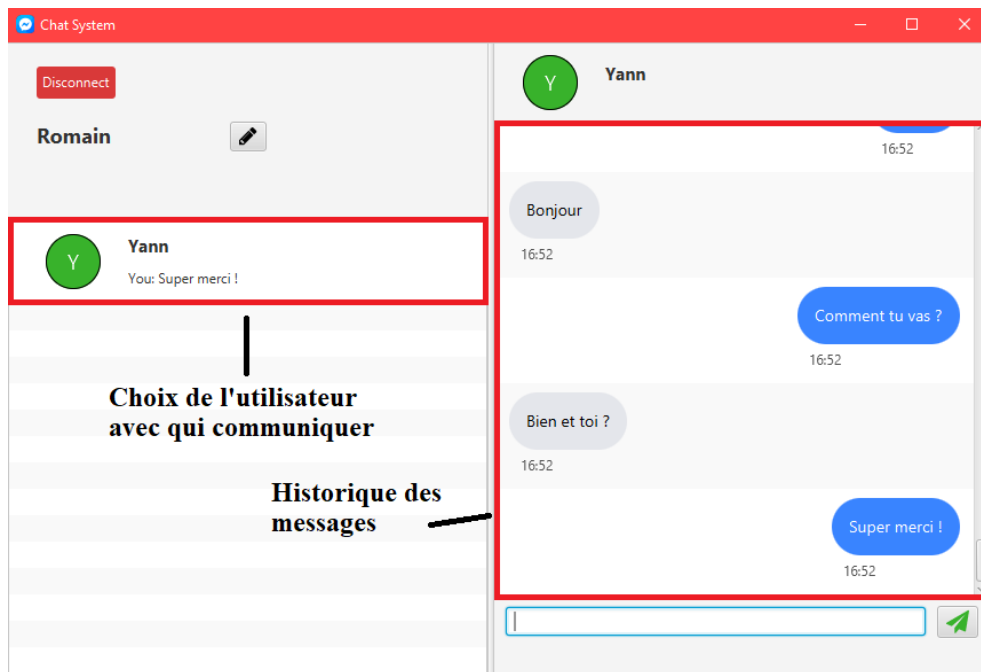


Figure 14 - Écran principal : sélection d'un utilisateur

Pour envoyer un message à la personne sélectionnée, il suffit d'écrire le message dans la zone de texte prévue à cet effet en dessous de l'historique des messages et d'appuyer sur le bouton d'envoi à droite ou le raccourci clavier "Entrée".

Le dernier message envoyé sera visible dans la liste des utilisateurs en dessous du pseudonyme de la personne avec qui vous avez conversé, en plus d'apparaître dans la liste des messages à droite.

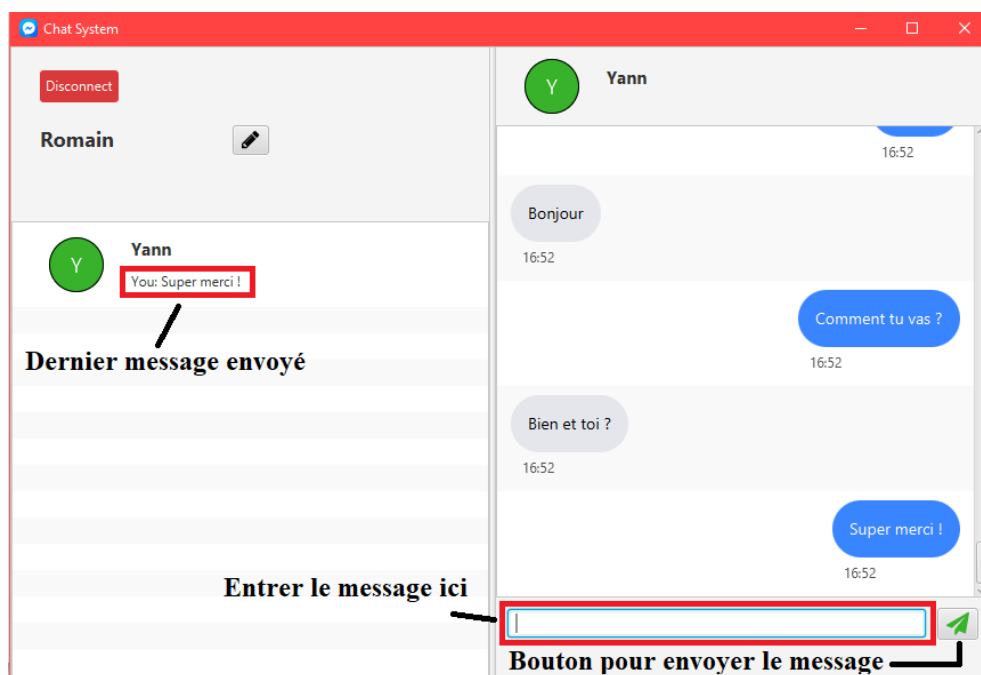


Figure 15 - Écran principal : envoi d'un message

Les personnes connectées sont repérables dans la liste par un cercle vert rempli par la première lettre de leur pseudonyme, et celles déconnectées possèdent un cercle grisé. Il n'est pas possible d'envoyer un message à une personne déconnectée mais seulement de voir l'historique des messages envoyés.

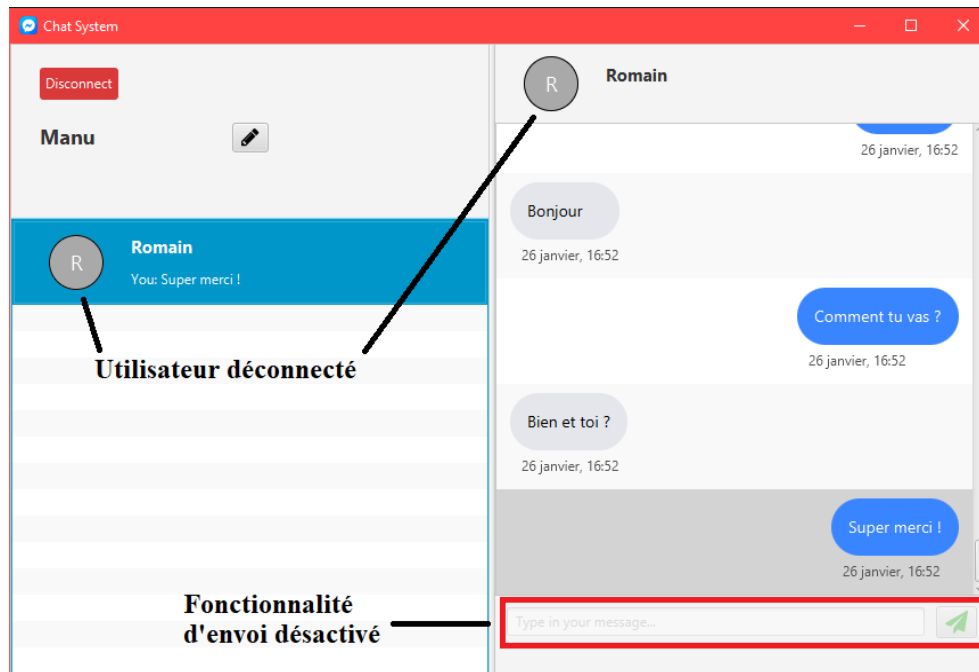


Figure 16 - Écran principal : désactivation d'envoi de message & utilisateur déconnecté

Conclusion

Pour conclure sur notre projet, nous sommes heureux d'avoir eu la chance de travailler sur un tel projet dans le cadre de nos études. Cela a été pour nous l'occasion de travailler de manière concrète sur l'intégralité d'une application que nous avons nous-mêmes conçue.

En terme de montée en compétences, nous avons pu nous améliorer en gestion de projet et nous former à de nouveaux outils tels que JavaFX. Nous regrettons cependant que le contexte sanitaire ne nous ait pas permis de développer l'application dans les meilleures conditions possibles. Cela nous a handicapés en matière de temps et de moyens, ce qui a impacté notre projet sur sa dernière ligne droite.

Certaines améliorations sur notre projet seraient possibles, comme l'ajout d'envois de fichiers non textuels entre deux utilisateurs, des messages vocaux et un système de personnalisation de compte avec un choix de photo de profil, et l'ajout d'une liste de contacts favoris.