

Rapport du système de clavardage

Table des matières

Introduction.....	2
I. Conception, architecture du système et choix technologiques.....	3
II. Procédure d'évaluation et tests.....	8
III. Procédure d'installation et de déploiement.....	8
IV. Manuel d'utilisation.....	9
V. Gestion de projet.....	13
VI. Processus de développement automatisé.....	13
Conclusion.....	14

Introduction

Ce rapport présente la conception et le développement du système de clavardage que nous avons réalisé dans le cadre des cours de Conception et de Programmation Orientée Objet. Il s'agit d'un système de communication utilisé par les employés d'une entreprise permettant l'échange de messages textuels et de fichiers.

Nous présenterons dans un premier temps la conception du système sous forme de diagrammes UML ainsi que l'architecture du système et nos choix technologiques. Puis dans un second temps, nous décrirons nos procédures d'évaluation et de tests.

Nous passerons ensuite à la gestion du projet et au processus de développement automatisé avant de fournir la procédure d'installation et de déploiement et le manuel d'utilisation simplifié de l'application.

I. Conception, architecture du système et choix technologiques

Nous avons commencé la conception du système par l'identification des acteurs et des cas d'utilisations à partir du cahier des charges.

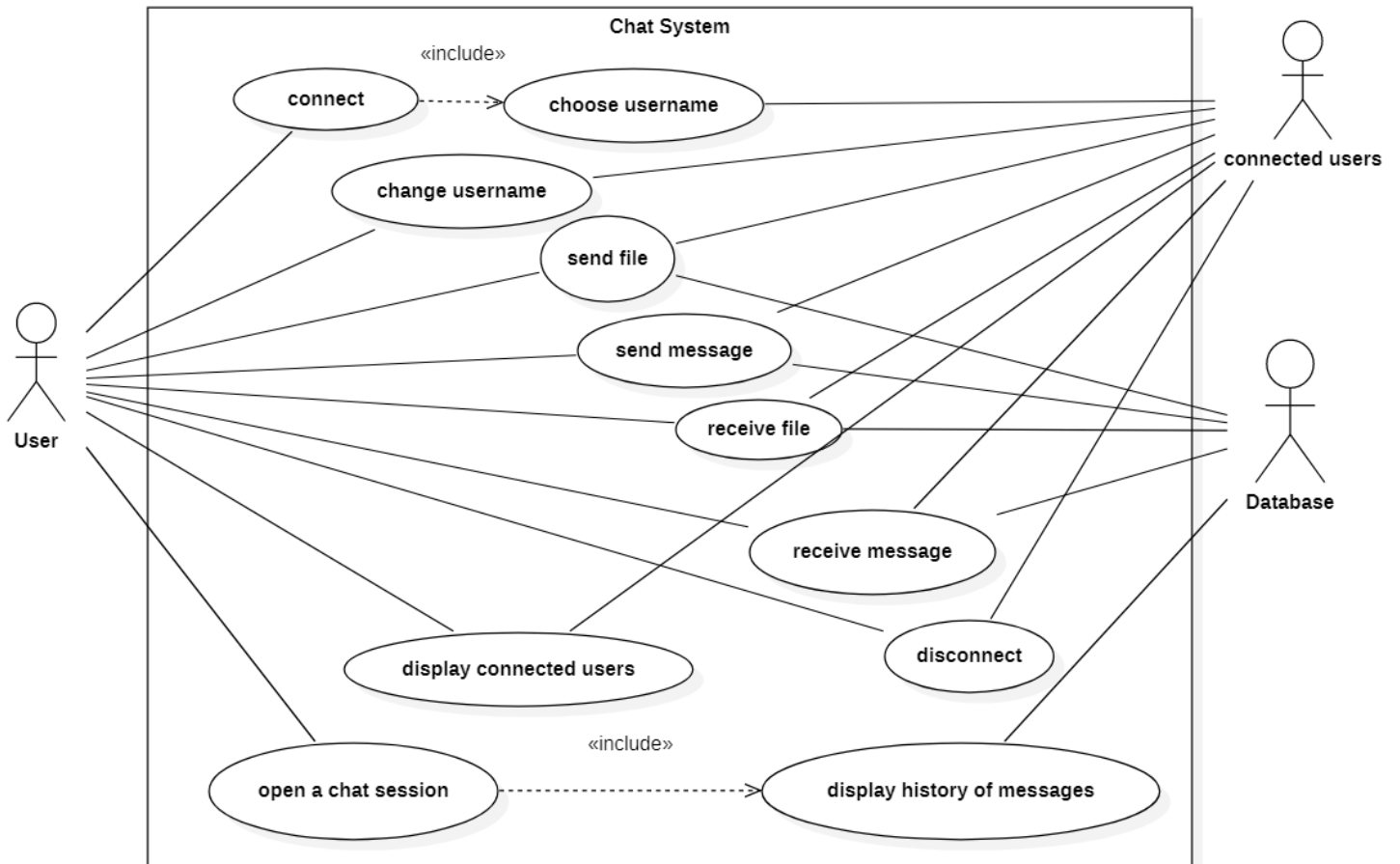


Figure I.1 Diagramme des cas d'utilisation du Chat System

À partir de ce diagramme des cas d'utilisation nous avons conçu différents diagrammes de séquences qui nous ont permis d'identifier plusieurs classes.

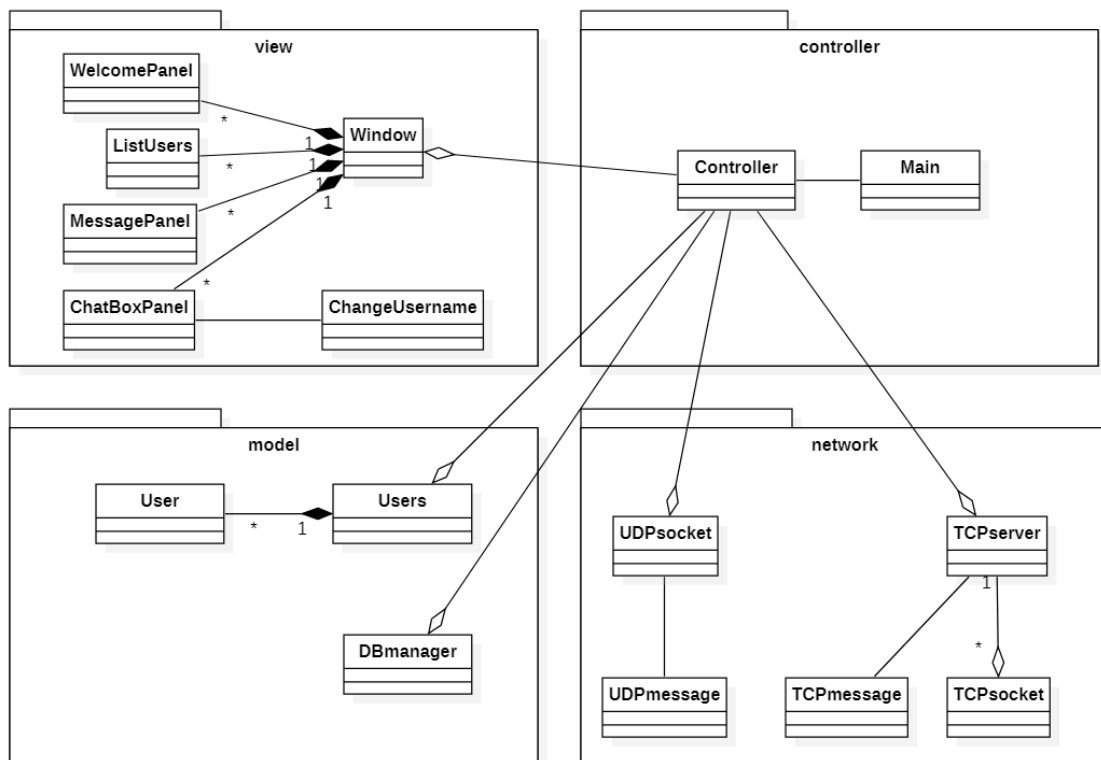


Figure 1.2 Diagramme de classe du Chat System

Pour plus de visibilité nous n'avons pas fait figurer les attributs et les méthodes des différentes classes sur ce diagramme. Par la suite, nous vous présenterons un diagramme de classe plus détaillé pour chaque package, sur lesquels nous avons fait figurer les attributs et méthodes essentielles des classes.

Concernant l'architecture du système, nous nous sommes appuyées sur le motif d'architecture logiciel MVC qui nous permet d'opérer des changements dans une des composantes (modèle, vue ou contrôleur) sans en affecter les deux autres. Ainsi, nous avons décidé de regrouper nos classes en 4 packages : model, view, controller et network.

I.1 Package network

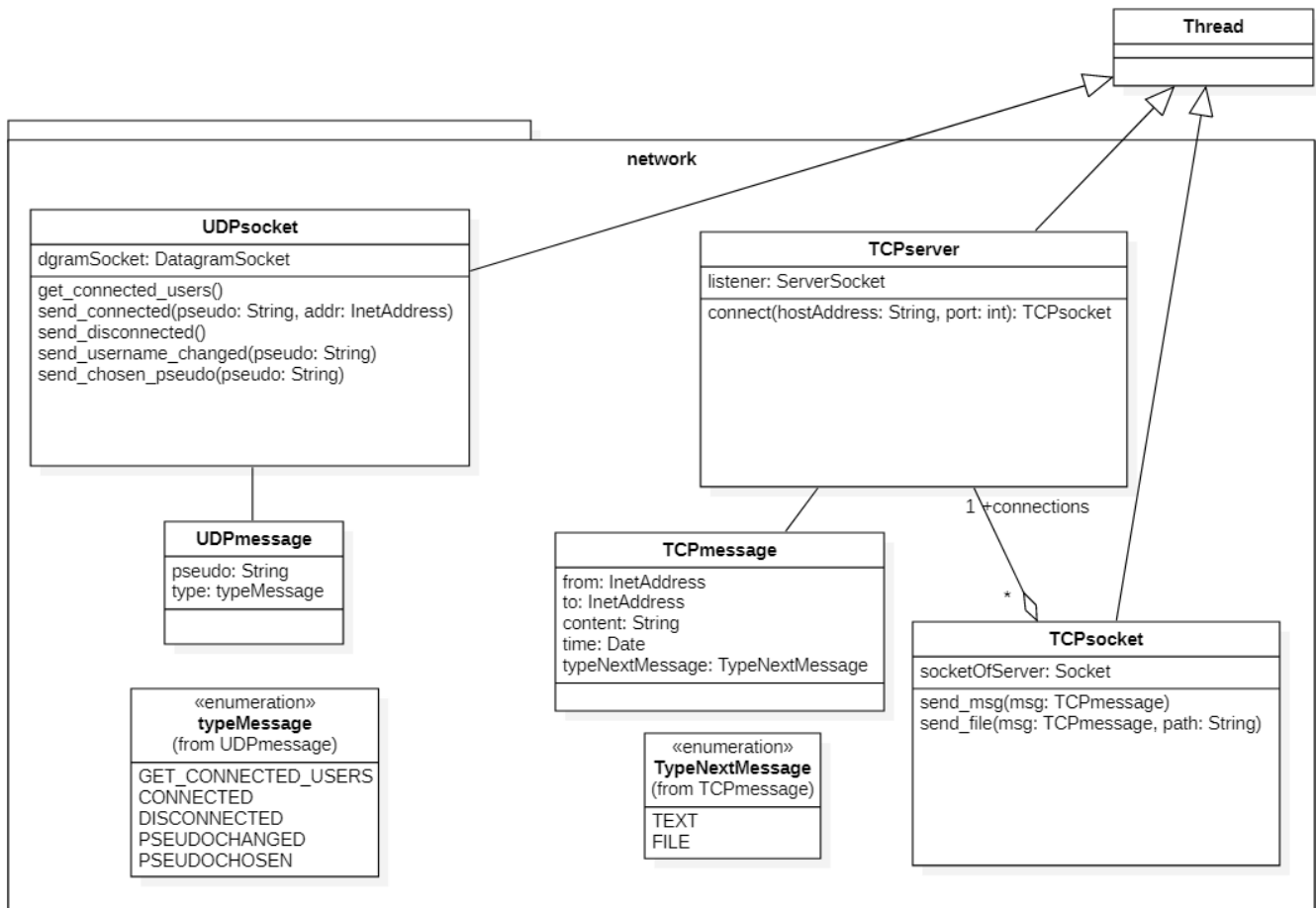


Figure I.1.1 Diagramme de classe du package network

Le package network est divisé en deux parties : la partie communication via UDP et celle via TCP.

Les messages envoyés par UDP nous permettent de découvrir la liste des utilisateurs connectés et de la mettre à jour au fil de l'exécution du programme. La classe **UDPsocket** met en œuvre un thread qui écoute continuellement sur son socket pour de nouveaux messages. De plus, elle fournit différentes méthodes permettant l'envoi de message sur ce même socket.

Les messages et fichiers envoyés entre deux utilisateurs sont gérés par TCP qui est un protocole de communication fiable contrairement à UDP. La classe **TCPserver** met en œuvre un thread qui attend continuellement pour de nouvelles connexions. Elle délègue chaque nouvelle connexion à une nouvelle instance de **TCPsocket**. Ainsi pour chaque session de clavardage entre deux utilisateurs un socket TCP est alloué.

I.2 Package model

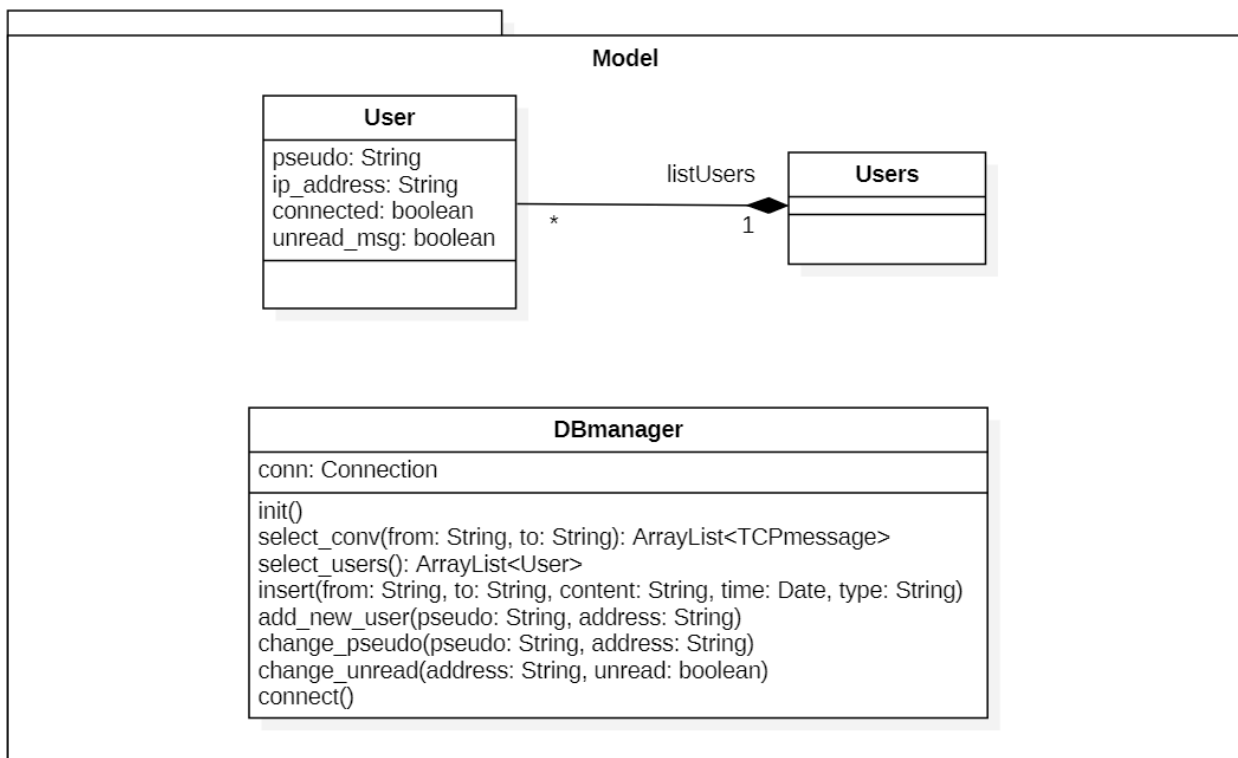


Figure I.2.1 Diagramme de classe du package model

Le package model contient la liste des utilisateurs ainsi que le manager permettant l'accès à la base de données.

La classe Users est une composition de la classe User, elle fournit différentes méthodes permettant la mise à jour de la liste des utilisateurs. Chaque utilisateur est identifié par son pseudo, son adresse IP et son statut de connexion. L'attribut unread_msg permet de savoir s'il l'utilisateur local possède oui ou non des messages non lu avec cet utilisateur.

La classe DBmanager implémente différentes méthodes permettant l'exécution de requêtes sur la base de données.

Dans cette base de données, nous stockons deux tables : la table contenant l'historique des messages et la table contenant l'ensemble des utilisateurs rencontrés jusqu'à présent. Cette dernière table permet de faire figurer dans la liste des utilisateurs des utilisateurs non connectés mais avec qui on aurait pu avoir des conversations antérieures. Ainsi, on peut toujours afficher l'historique des messages avec un utilisateur non connecté en le sélectionnant dans la liste.

Concernant l'implémentation de la base de données, nous avons choisi la base de données décentralisées SQLite. L'avantage de cette base de données est qu'elle est légère et ne requiert pas d'installation ou de configuration particulières. Il suffit d'intégrer la librairie sqlite-jdbc au projet.

I.3 Package view

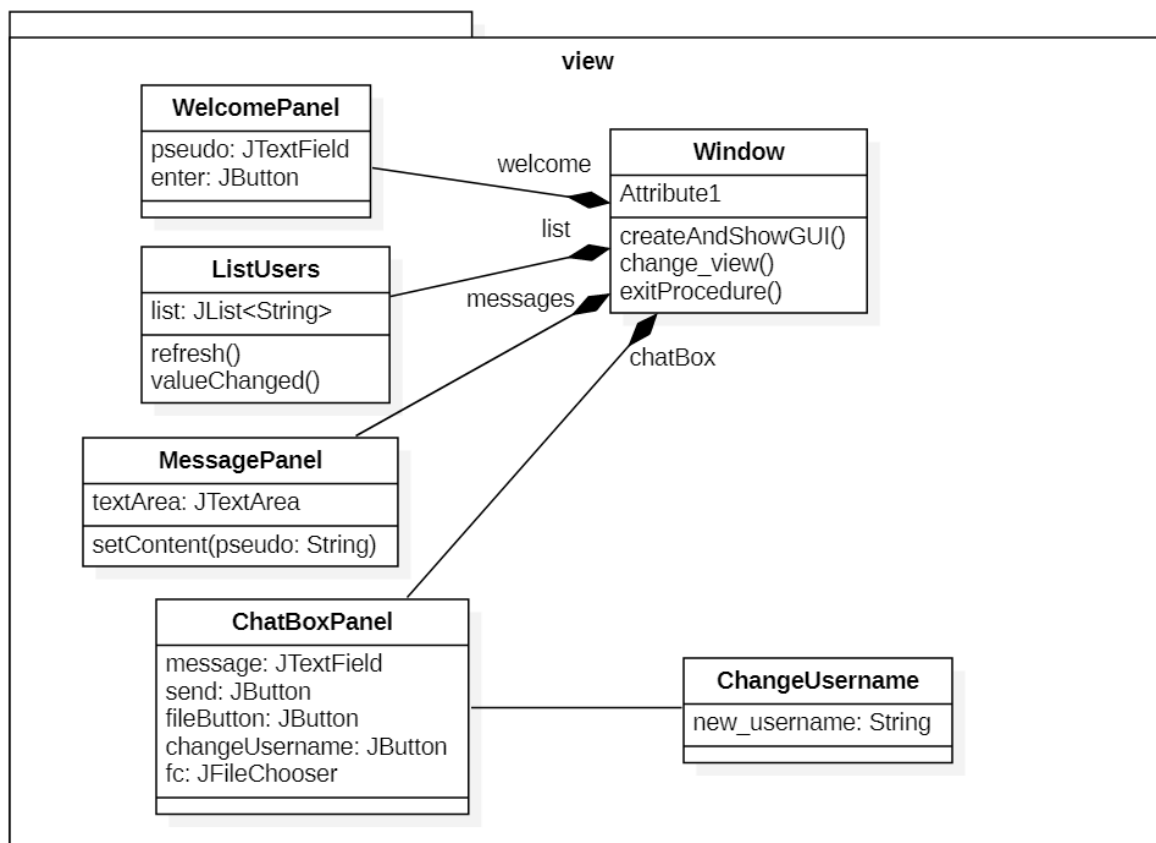


Figure I.3.1 Diagramme de classe du package view

L'interface graphique a été réalisée avec Java Swing. Par rapport à AWT, Java Swing est plus léger, il offre plus de composants et suit le motif MVC.

Nous avons utilisé la librairie open-source FlatLaf qui propose de nombreux thèmes pour les applications Java Swing. Les thèmes de la librairie FlatLaf sont plus modernes que les "look and feel" présents dans l'API de Java.

La classe Window représente la fenêtre principale, elle est composée de différents panels (WelcomePanel, ListConnectedUsers, MessagePanel et ChatBoxPanel). ChangeUsername est une fenêtre qui apparaît lorsque l'on clique sur le bouton "change username" du ChatBoxPanel.

I.4 Package controller

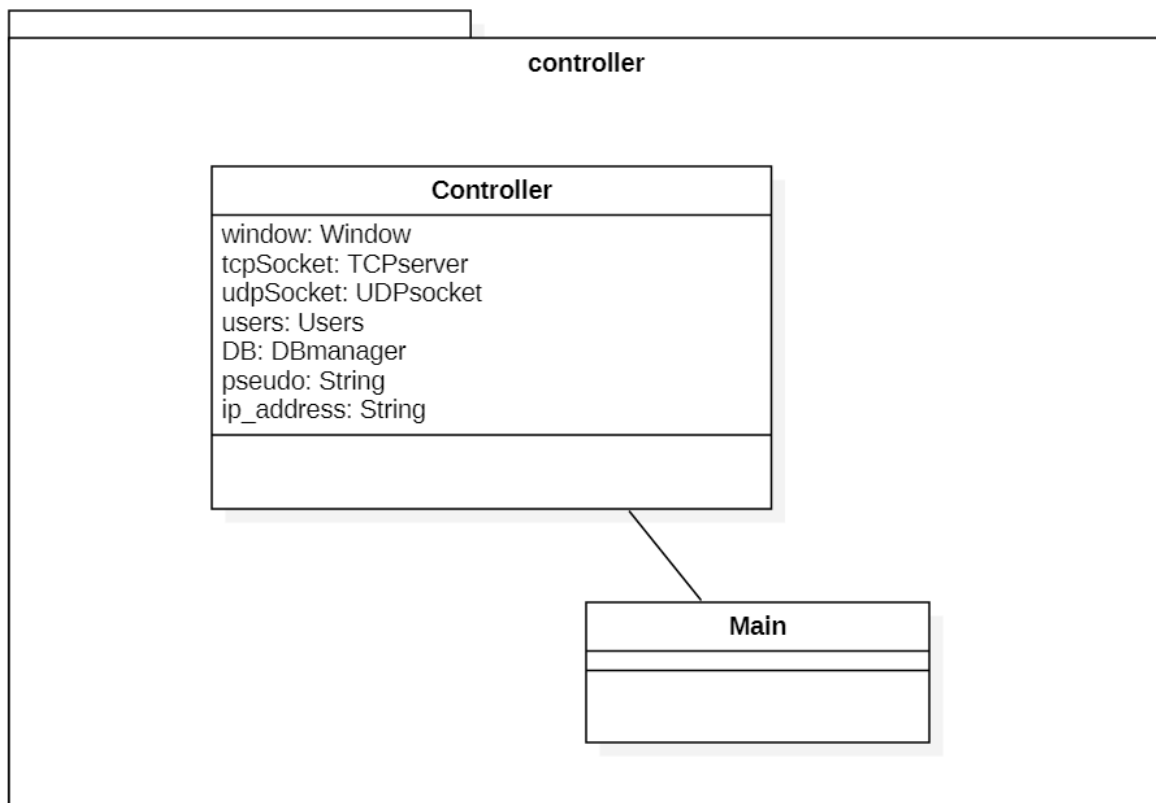


Figure I.4.1 Diagramme de classe du package view

Le package controller contient la classe Controller ainsi que la classe Main qui implémente la méthode *public static void main(String[] args)*. La classe Controller fait le lien entre la vue, le modèle et la partie réseau. Elle possède également le pseudo et l'adresse IP de l'utilisateur local.

II. Procédure d'évaluation et tests

Nous avons effectué des tests au début de l'implémentation pour la partie réseau et la partie base de données. Cela nous permet de vérifier le fonctionnement de code et corriger l'erreur à l'heure.

Nous avons également testé les différentes fonctionnalités de l'application en veillant à appliquer tous les cas possibles d'utilisation.

III. Procédure d'installation et de déploiement

Pour déployer l'application il suffit de télécharger le fichier .jar présent dans le répertoire git <https://github.com/Elise-Le-Roux/Chat-System.git> sur chacune des machines des utilisateurs. Nous avons supposé que chaque employé à sa propre machine avec une adresse IP appartenant au réseau de l'entreprise.

Une version de Java supérieure ou égale à 7 est requise pour l'exécution du fichier jar.

IV. Manuel d'utilisation

La commande pour lancer l'application est la suivante : **java -jar [chemin d'accès]/Chat-System.jar** .

Lorsque vous ouvrez l'application, une fenêtre avec la liste des utilisateurs à gauche et une zone pour entrer votre nom d'utilisateur au centre s'affiche.

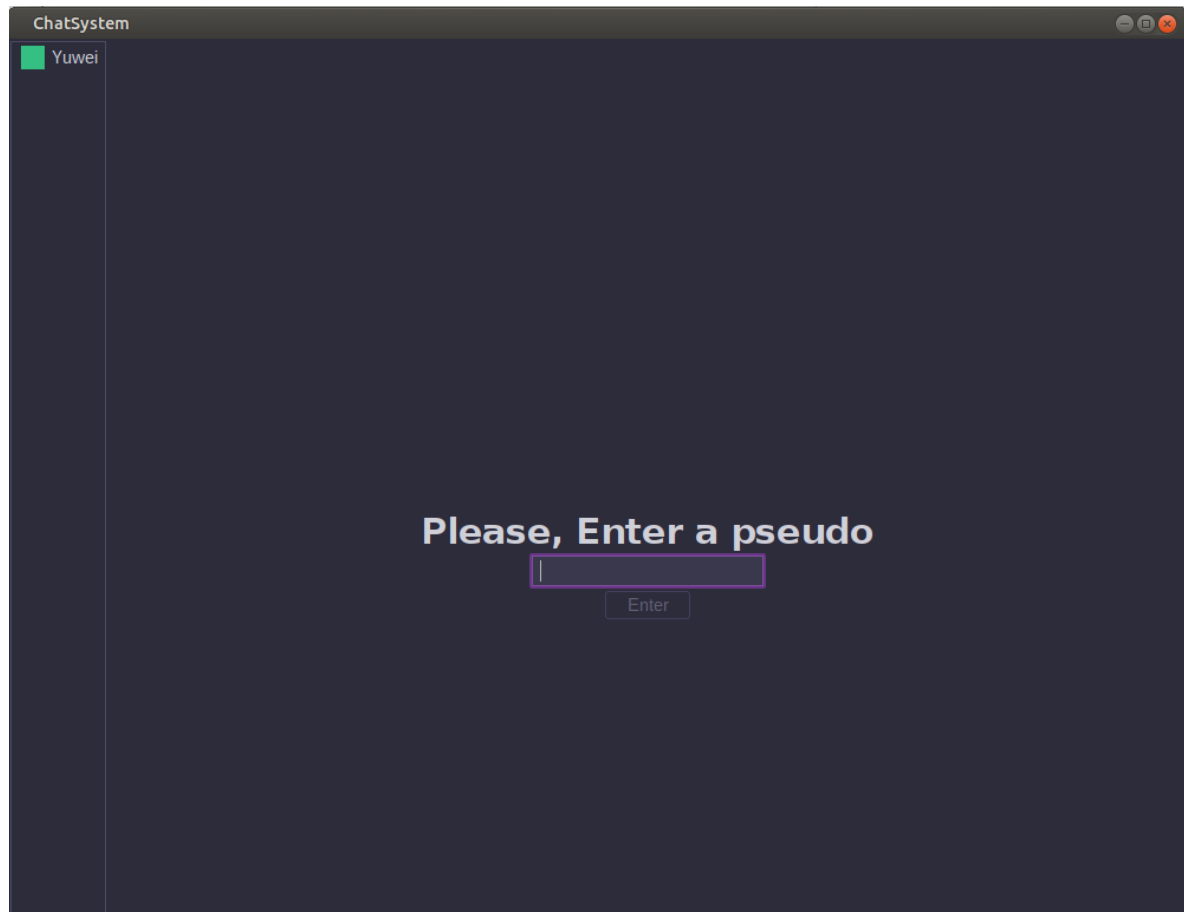


Figure IV.1 Fenêtre d'accueil de l'application

Vous devez entrer votre pseudonyme pour continuer. Il doit être différent des utilisateurs qui s'affiche sur la liste de gauche sinon un message d'erreur sera levé vous demandant d'entrer un nouveau pseudonyme.

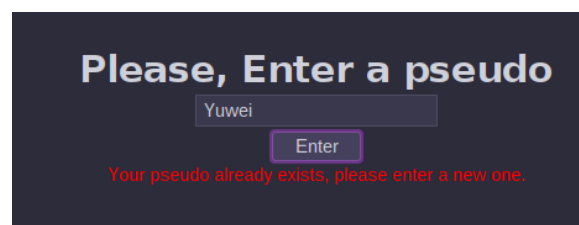


Figure IV.2 Message d'erreur lors du choix du pseudonyme

Une fois votre nom d'utilisateur choisi vous pouvez vous identifier sur la liste des utilisateurs par un carré vert perforé.

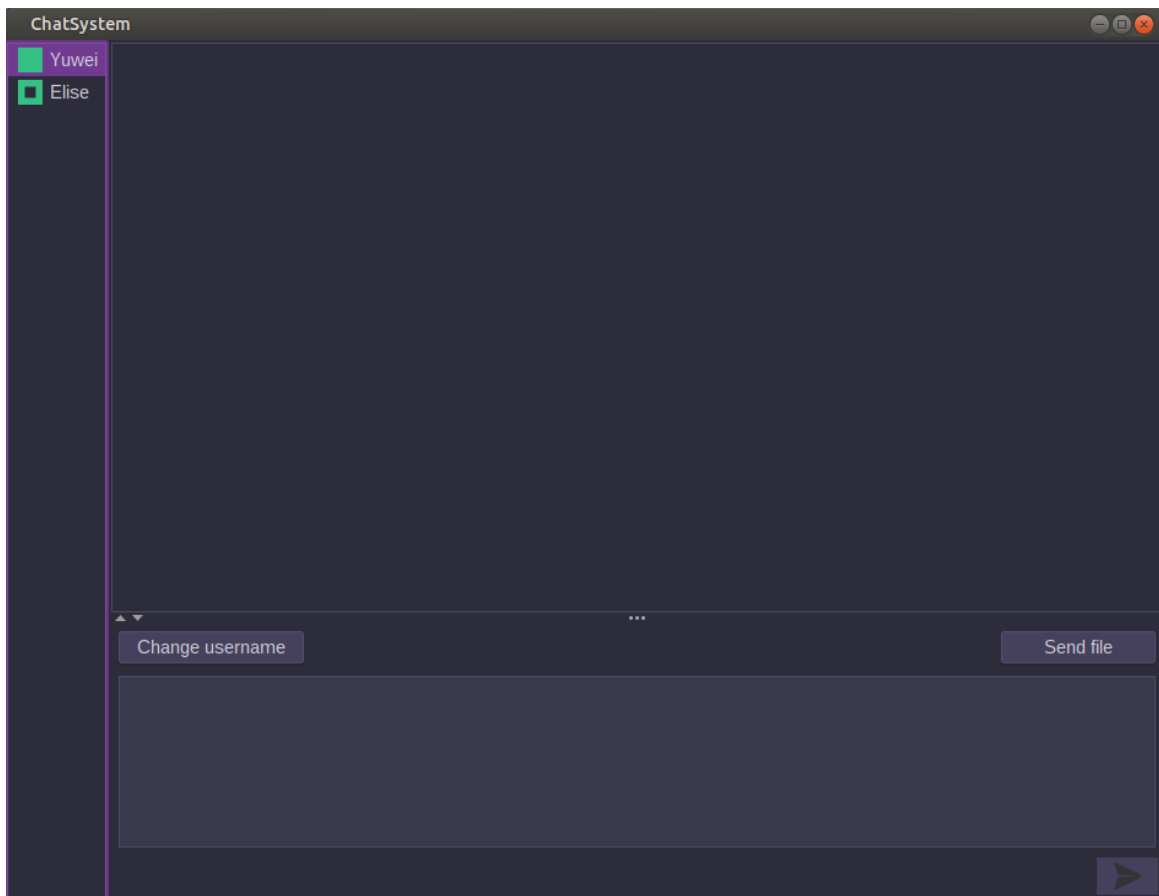







Figure IV.3 Fenêtre principale de l'application

Veillez vous référer à la légende ci-dessous pour la signification des différentes icônes utilisées.

-  Utilisateur connecté.
-  Utilisateur déconnecté.
-  Vous êtes connecté.
-  Vous êtes déconnecté. L'icône apparaît lorsque vous choisissez votre nom d'utilisateur et que vous vous êtes déjà connecté au moins une fois auparavant. Elle se situe alors à gauche de votre ancien nom d'utilisateur. Vous pouvez choisir d'entrer votre ancien pseudo ou un nouveau.
-  Message non lu (l'utilisateur peut-être connecté ou déconnecté). Cette icône disparaît en cliquant sur le pseudo pour faire apparaître l'historique des messages et ainsi lire le message.

Vous pouvez ensuite sélectionner un utilisateur présent dans la liste par un simple clic droit. Vous aurez alors accès à l'historique des messages. Dans le cas où l'utilisateur est connecté vous aurez également la possibilité de lui envoyer des messages ou des fichiers.

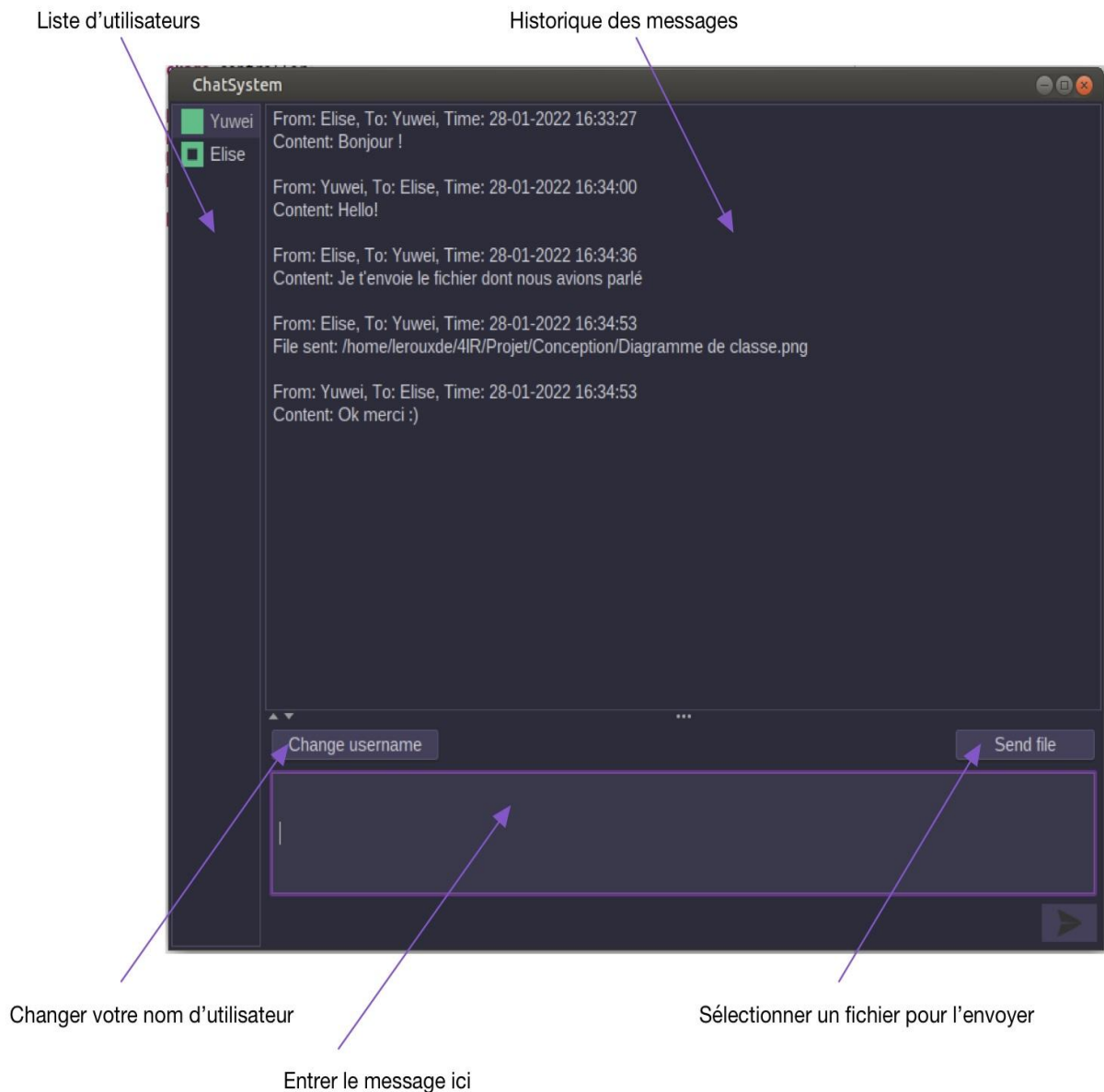


Figure IV.4 Conversation avec un utilisateur

Vous pouvez à tout moment choisir de changer de nom d'utilisateur. Lorsque vous cliquez sur le bouton "change username" une fenêtre apparaît pour entrer votre nouveau pseudonyme. Cliquez sur ok pour confirmer ou bien annulez.

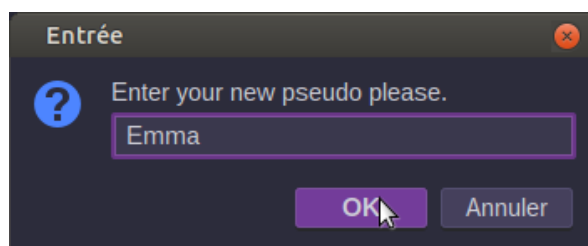


Figure IV.5 Fenêtre de changement de pseudonyme

Si le pseudonyme que vous avez entré est déjà utilisé, un message d'erreur apparaît vous demandant d'en choisir un autre.

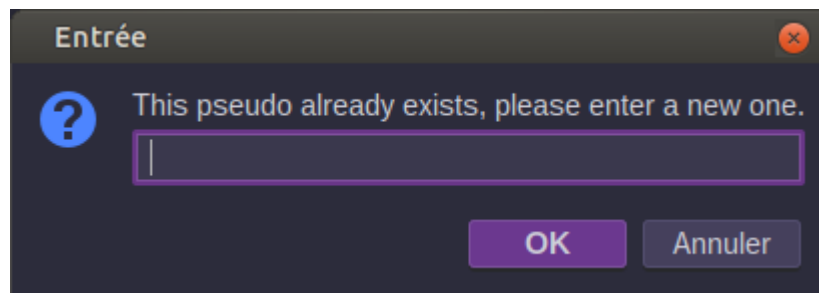


Figure IV.6 Message d'erreur lors du changement de pseudonyme

Lorsque vous voulez envoyer un fichier une fenêtre s'affiche vous permettant de sélectionner le fichier à envoyer. Pour confirmer l'envoi appuyez sur le bouton ouvrir.

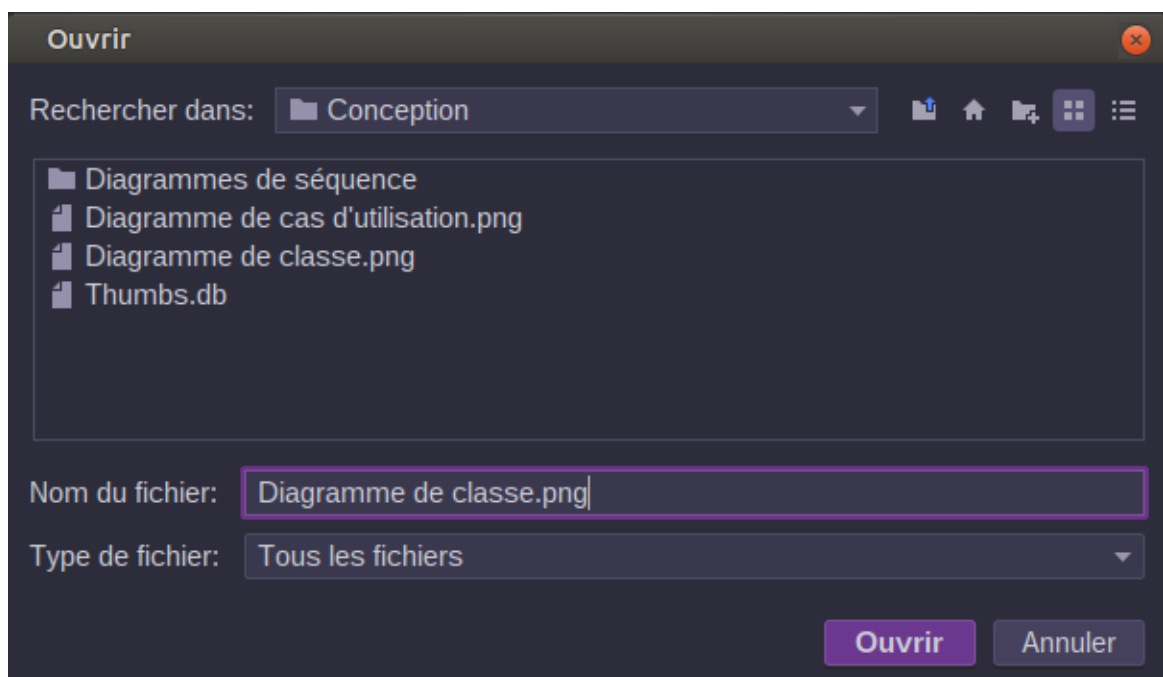


Figure IV.7 Fenêtre de sélection d'un fichier à envoyer

V. Gestion de projet

Pour l'organisation et la gestion du projet nous avons utilisé la méthode agile scrum. Nous avons utilisé pour cela l'outil JIRA. Nous avons commencé par définir les différentes "users stories" en vérifiant qu'elles soient indépendantes, négociables, estimables, suffisamment petites, testables et qu'elles apportent de la valeur. Puis nous les avons classées par ordre de priorité et nous les avons estimées en utilisant la méthode Poker. Ensuite nous avons décomposé les "users stories" en tâches. Chaque semaine nous définissions un nouveau sprint contenant diverses tâches à compléter en fonction de leur estimation et de leur ordre de priorité. Cette méthode nous a permis de nous concentrer chaque semaine sur des objectifs précis et de ne pas coder tout en même temps de manière désordonnée.

VI. Processus de développement automatisé

Nous avons utilisé l'outil de versionnage git. Le dépôt ne contient qu'une seule branche master car nous avons travaillé ensemble sur un seul et même poste de travail.

Pour l'automatisation de la compilation et de la génération de l'exécutable nous avons utilisé jenkins.

Nous avons défini trois jobs jenkins :

1- BuildChatSystem

Ce job exécute la commande **mvn clean compile**, ce qui nettoie le projet en supprimant les fichiers issus des précédents builds et le compile.

2- BuildGitChatSystem

Ce job exécute également la commande **mvn clean compile** mais elle compile les fichiers présents dans notre répertoire distant git et non pas sur la machine comme précédemment. C'est la scrutation de l'outil de gestion de version (git) toutes les minutes qui déclenche le build. Donc à chaque fois qu'il y a un nouveau push sur le git, le projet est compilé.

3- ChatSystemPackaging

Ce job exécute la commande **mvn clean package**.

Nous avons configuré Maven pour que cette commande entraîne la création d'un fichier jar et d'un fichier fat jar. Le fichier fat jar contient les dépendances du projet, ce qui permet, lors de l'exécution, de ne pas avoir à lister tous les fichiers jar dont notre application dépend dans le classpath.

Nous avons ensuite créé un pipeline entre le deuxième et le troisième job. Ce qui permet de lancer la création des fichiers jar si la compilation du projet a réussi.

Conclusion

Ce projet s'est révélé très enrichissant dans la mesure où il a permis une approche concrète du domaine du développement logiciel. Il nous a permis d'approfondir nos connaissances sur le langage Java (découverte de Swing, utilisation de sockets...), sur l'utilisation des bases de données ainsi que des outils utilisés fréquemment dans l'entreprise comme Jenkins, Maven et Jira.

Les problèmes principaux, que nous avons rencontrés, concernent la conception du projet au départ, cependant celle-ci est devenue beaucoup plus claire une fois la programmation du projet commencée.