



**OURAHOU - EYNAUD**

**RAPPORT DE PROJET**  
**CHAT SYSTEM**

**1<sup>er</sup> semestre 2022/2023**

**INSA**

INSTITUT NATIONAL  
DES SCIENCES  
APPLIQUÉES  
**TOULOUSE**





# PROJET

---

Dans ce projet, nous devons réaliser un système de chat avec différentes « contraintes » liées à un cahier des charges. Notre système doit permettre à différentes personnes de pouvoir démarrer une ou plusieurs sessions de clavardage avec d'autres utilisateurs, pour pouvoir communiquer entre eux.

## SOMMAIRE

---

### I – COO et différents diagrammes.

- a. Diagramme de Cas d'Utilisation
- b. Diagramme de Séquence
- c. Diagramme de Classe

### II – Architecture de notre système.

- a. Le module Back
- b. Le module Front
- c. Le module Commun

### III – Les procédures de test et d'évaluation.

### IV – Installation et Utilisation du système.

- a. Installation
  - b. Utilisation
- 

# I – COO et différents diagrammes

## a. Diagramme de Cas d'Utilisation



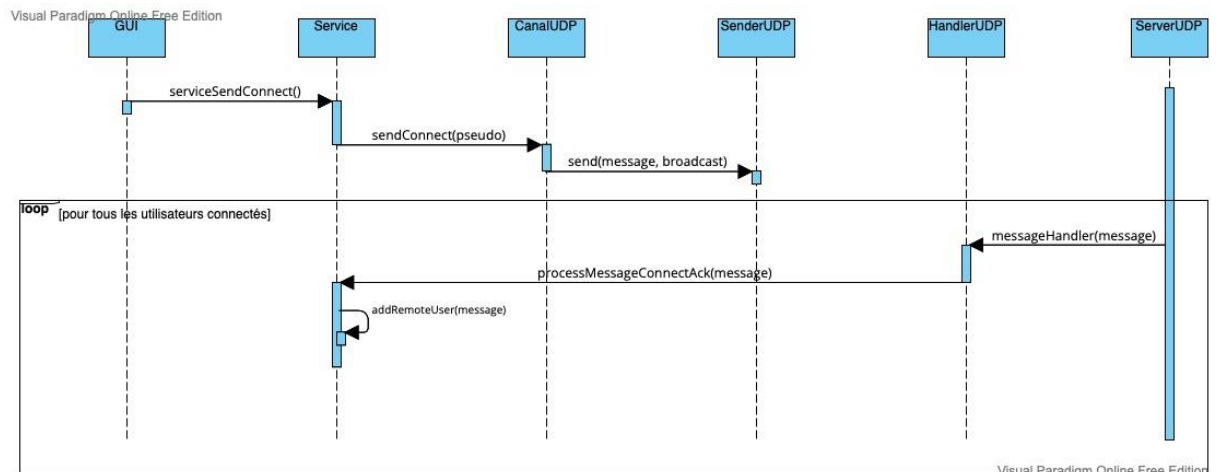
Voici notre diagramme de Cas d'Utilisations qui explique comment va fonctionner notre système de chat. Le diagramme de Cas d'Utilisation prend en compte des éléments « simples » de la conception pour simplement expliquer le

fonctionnement. Ainsi, on représente les éléments cités dans le cahier des charges et qui sont donc attendus dans le résultat final : choisir un pseudonyme, pouvoir le changer, envoyer un message, démarrer une session, etc... .

## b. Diagramme de Séquence

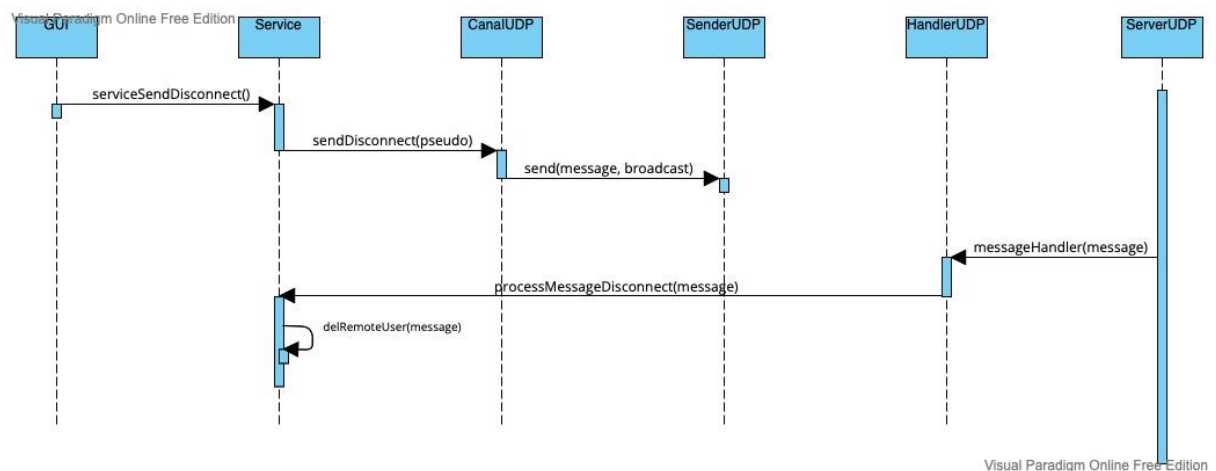
Pour pouvoir rendre plus lisible la lecture, nous avons effectués plusieurs diagrammes de séquences.

Diagramme de séquence de connexion :



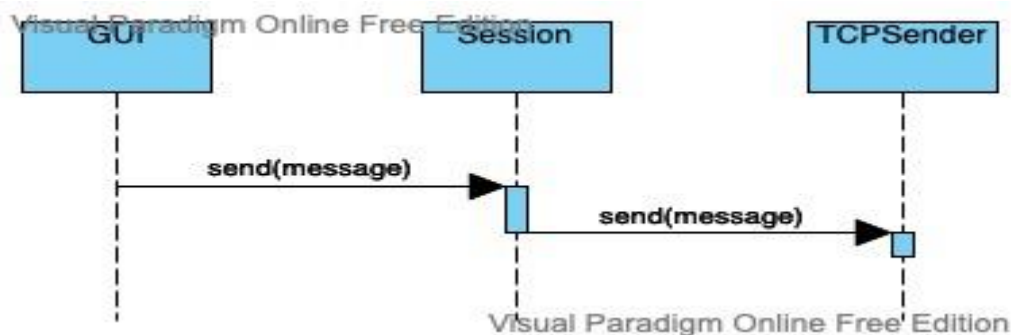
Ce diagramme retranscrit la manière dont nous avons implémenter la connexion sur notre système de chat, avec la demande de connexion, à partir d'un pseudo, initiée depuis l'interface graphique pour ensuite l'envoyer aux différents utilisateurs depuis le canal UDP. Ensuite, on accepte la connexion puis on ajoute l'utilisateur à la liste des utilisateurs.

Diagramme de séquence de déconnexion :

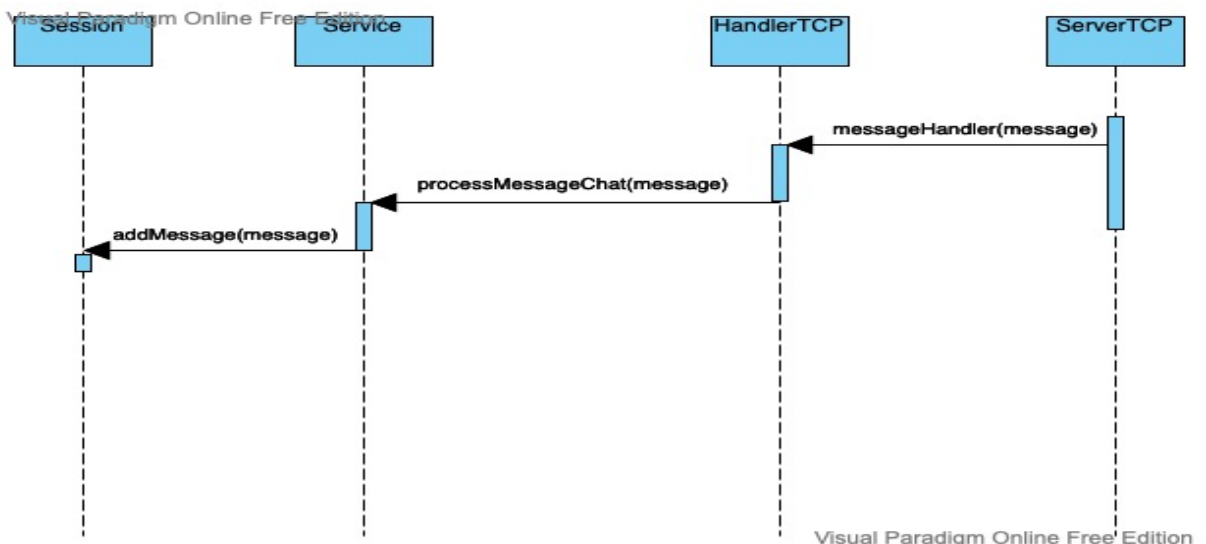


On au même principe que précédemment, mais pour la déconnexion, avec une demande de déconnexion, puis on enlève de la liste des utilisateurs.

Diagramme de séquence d'envoi et de réception de message :

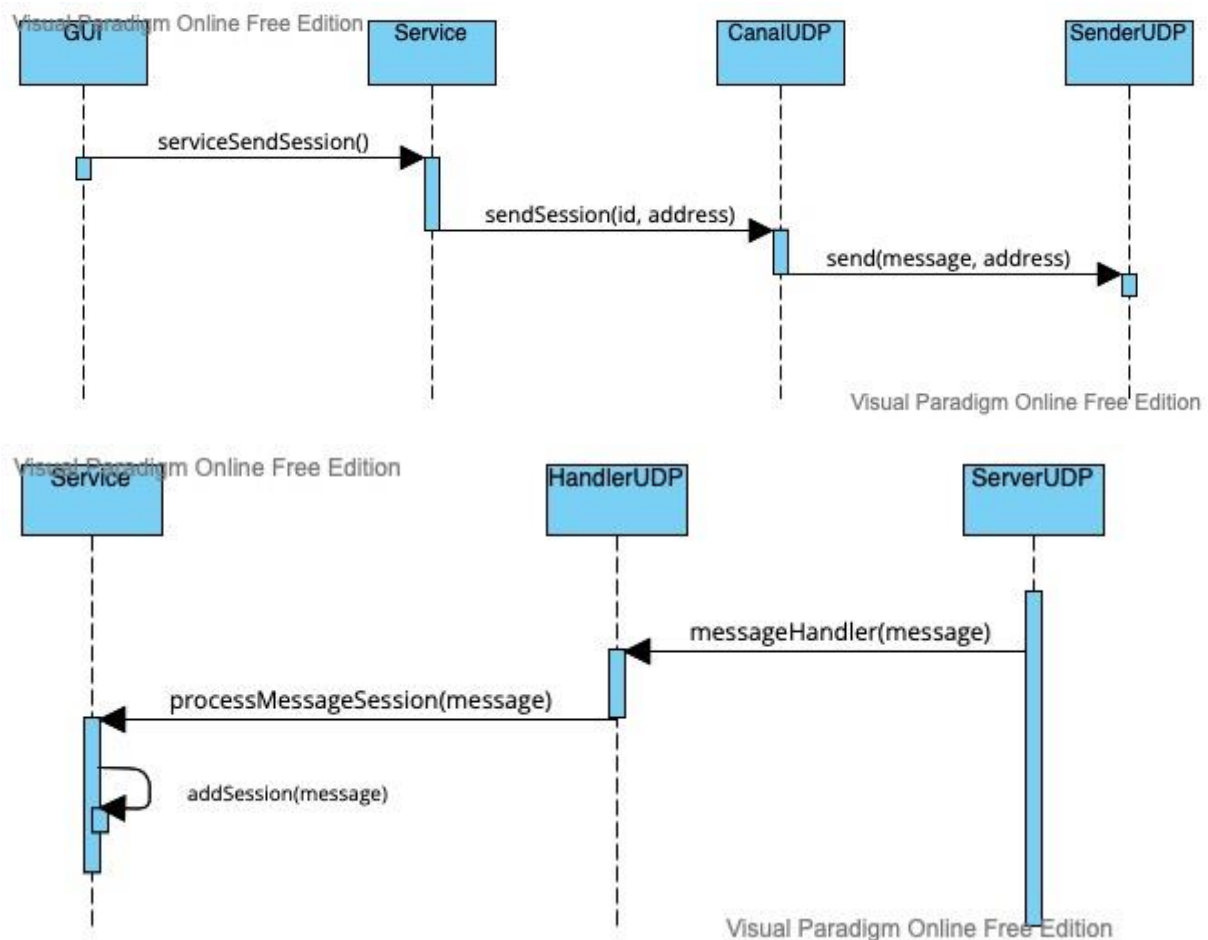


Depuis l'interface graphique on envoie le message sur la session concernée, qui elle va s'occuper d'activer la fonction « send(message) » depuis TCPSender qui s'occupera de la transmission du message.



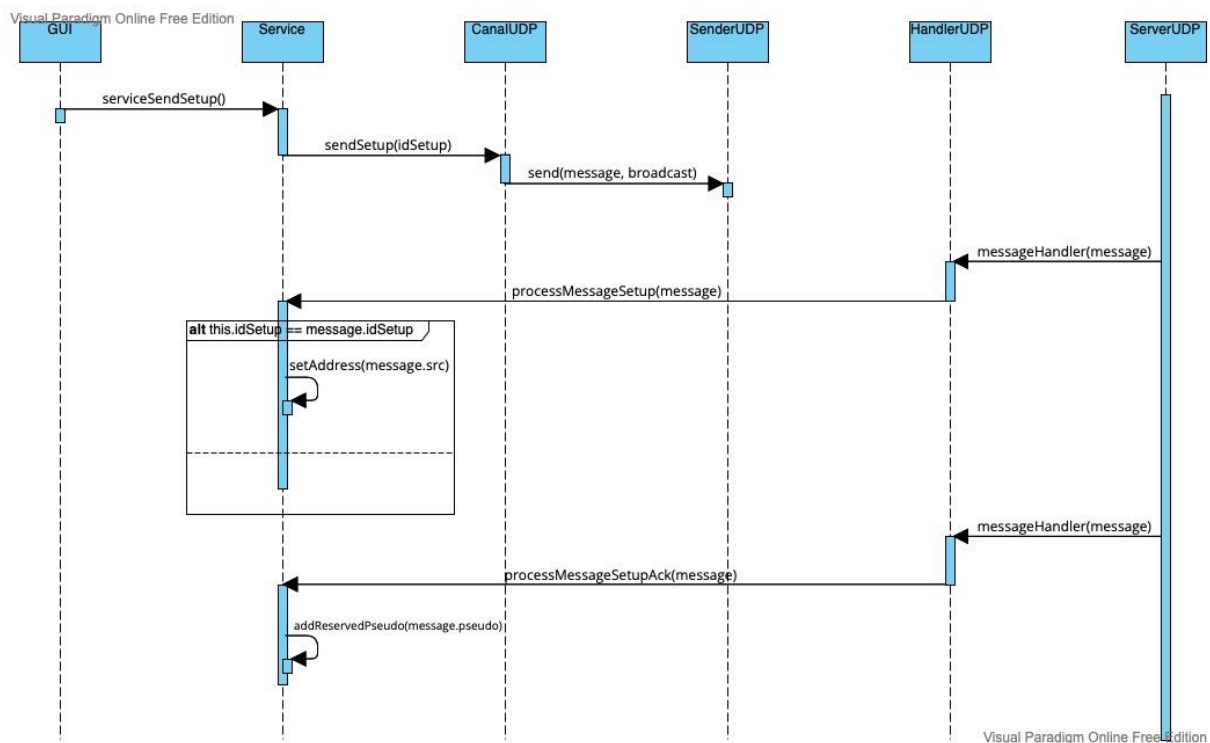
Après avoir reçu le message envoyé par le TCPSender, le ServerTCP traite le message (en fonction de son type, voir le diagramme de classe) puis l'affiche sur la session de l'utilisateur concerné par la session.

## Diagramme de séquence de création de session :



Depuis l'interface graphique, on initie la création d'une session. Pour cela, on établit la connexion via UDP avec l'autre utilisateur pour le prévenir de cette création. Une fois que les éléments essentiels sont envoyés par le SenderUDP, le ServerUDP traite le message reçu (qui notifie de la création d'une session) et ajoute une nouvelle session à l'utilisateur concerné.

## Diagramme de séquence d'initiation avant connexion :

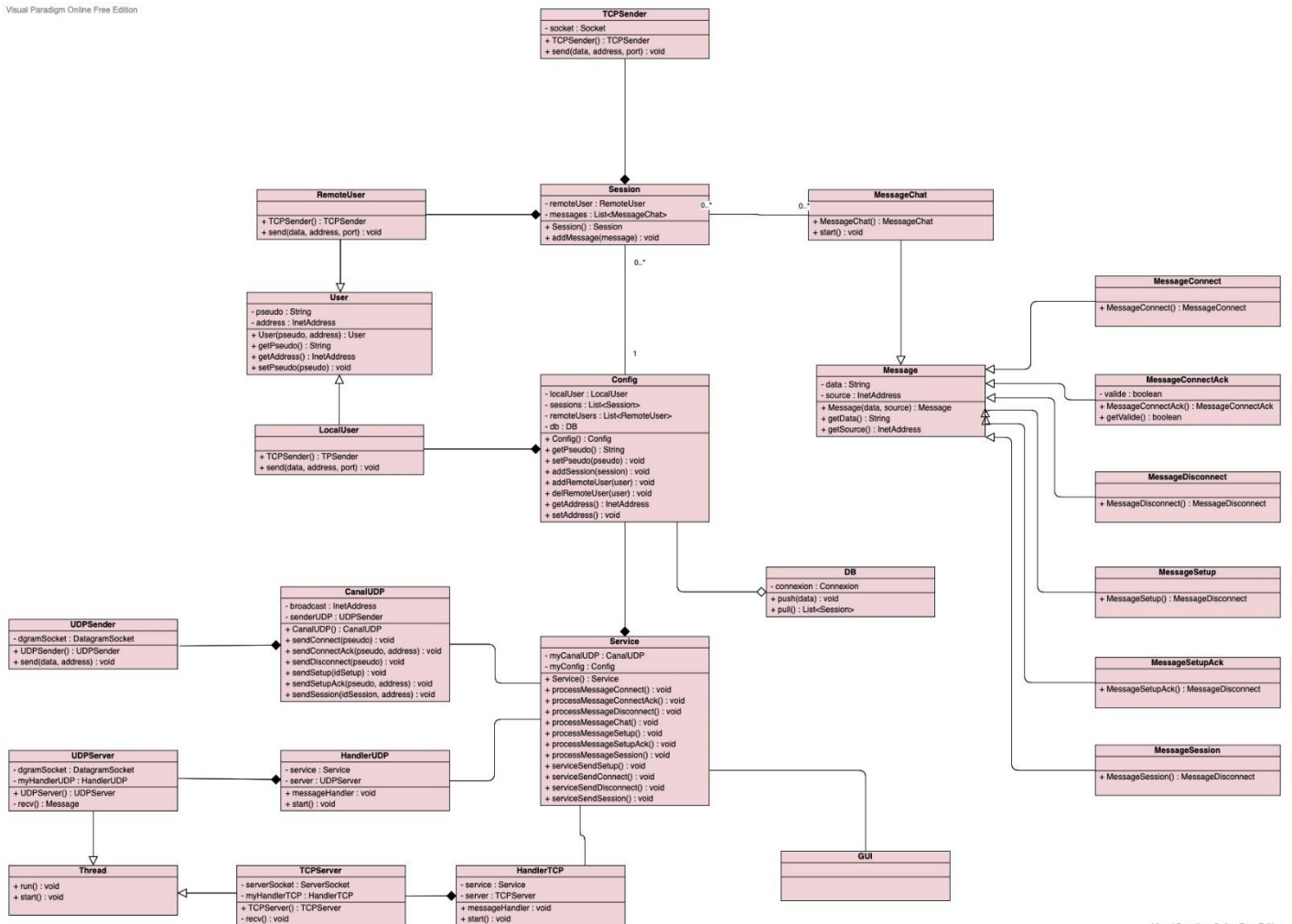


Avant de se connecter, on envoie un message signifiant notre arrivée à tous les utilisateurs déjà présent sur le système (c'est ce qui est réalisé en lançant la première méthode : « `serviceSendSetup()` »). En lançant cela en broadcast, nous avons donc deux cas à traité :

- La réception de notre propre message envoyé en broadcast, ainsi cela nous permettra de récupérer notre propre adresse
- La réception d'un message envoyé par un autre utilisateur en broadcast, et dans ce cas là nous lui renvoyons notre pseudo.

## c. Diagramme de Classe

Visual Paradigm Online Free Edition



Visual Paradigm Online Free Edition

Voici notre diagramme de classe qui relate toutes les classes et dépendances de notre logiciel.

Il y a 7 classes qui héritent de la classe Message. Chacune de ces classes représente un type de communication différent, permettant de rendre le code plus clair et se repérer plus facilement dans celui-ci.

On a également utilisé un design pattern « Singleton » pour la Data Base de façon à pouvoir créer une unique instance de la Dao. Pour ce faire, on utilise une méthode garantit Thread safe par Java.

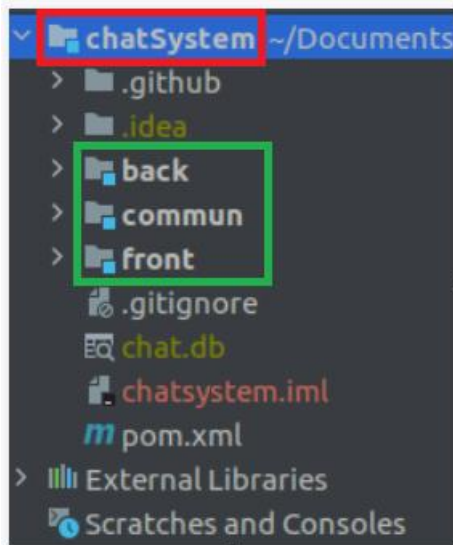


Pour tout le système, nous avons également utilisé le design pattern IoC (Inversion of Control) grâce à l'injection de dépendances. Cette architecture nous permet de développer un code plus maintenable. De plus, l'IoC nous permet une implémentation par interface, dans le but de rendre le code plus flexible, plus évolutif et de faciliter les tests unitaires.

## II – Architecture de notre système

---

Notre architecture est composée d'un module qui en contient 3 autres :

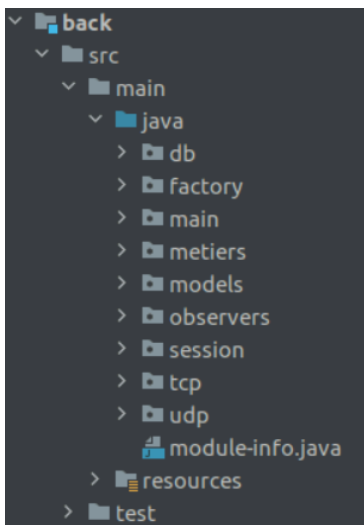


Le module « mère » : chatSystem, englobe les 3 autres modules :

Front, Commun et Back.

A travers ces trois modules, nous répartissons les différents aspects du chatSystem de façon à pouvoir structurer le code et avoir un projet organiser, de manière à pouvoir également se repérer facilement.

## a. Le module Back

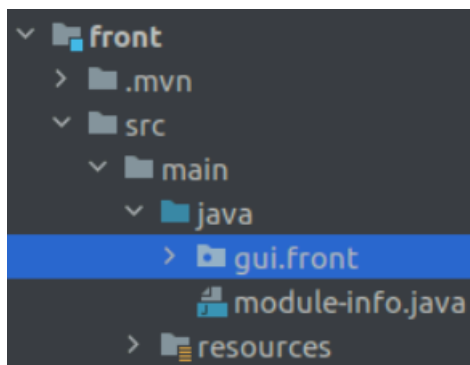


Dans ce module, on retrouve tous les éléments essentiels au fonctionnement du système de chat, soit toutes les classes visibles dans le diagramme de classe (englober dans les packages visibles sur la photo) de façon à pouvoir implémenter les différents types de services nécessaires à la réalisation du système de chat. Entre autres, on pourra gérer les choses suivantes :

- Envoyer et recevoir des messages de tous types.
- Enregistrer les messages dans la base de données.
- Gestion et création d'une session.
- Créer et afficher des utilisateurs avec leurs éléments qui les caractérisent
- Gérer les listes associées au système de clavardage.

## b. Le module Front

Dans ce module, on implémente l'interface graphique liée à notre application.



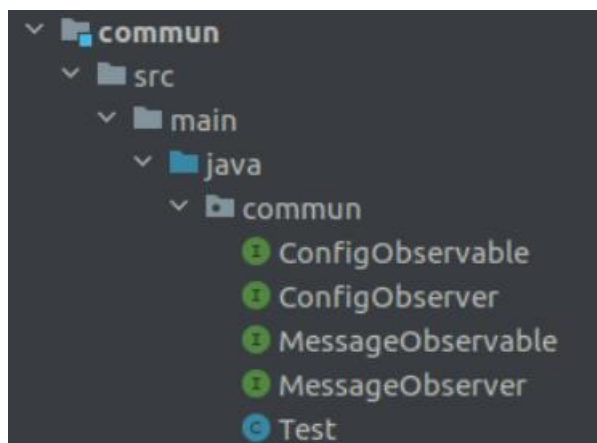
Dans le package « gui.front », on retrouve trois classes :

- ChatController
- LoginController
- PopupController

Ces trois classes vont permettre de gérer l'interface graphique en s'appuyant sur les différents éléments

implémenter dans le module Back (association de l'interface graphique avec l'échange de message, la connexion, déconnexion, ...).

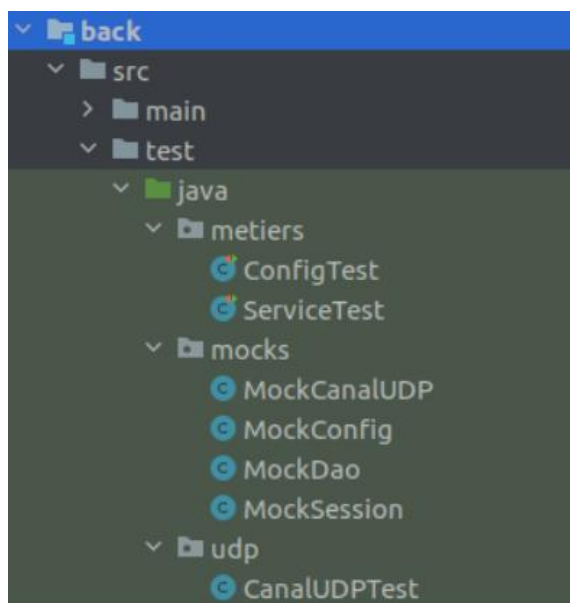
### c. Le module Commun



Dans ce module, on retrouve uniquement les interfaces d'observation entre le front et le back. Il s'agit donc simplement de la liaison avec ces deux modules.


## III – Les procédures de test et d'évaluation

---



On a réalisé plusieurs Tests Unitaires dans notre code. Ces Tests Unitaires concernent toutes les méthodes des classes Service et Config, les deux classes les plus importantes de notre code, pour lesquels on a utilisé l'injection de dépendances.

L'utilisation d'interfaces dans notre code nous permet d'améliorer l'implémentation de ces Tests Unitaires, en isolant chaque classe.



Pour ce faire on utilise des « Mock », soit des « fausses » classes très simples qui vont venir remplacer les dépendances de la classe testé. Grâce à cette pratique, nous alons pouvoir tester précisément la classe ciblée.

## IV – Installation et Utilisation du système

---

### a. Installation du système

Pour installer le système, il suffit de générer le .jar grâce à maven -assemnly-plugin. Une fois que nous obtenons le fichier .jar, il suffit de suivre les indications suivantes :

#### Récupération du JAR (Méthode 1)

##### Clone le projet en local

```
git clone https://github.com/Mathis05000/chatSystem.git
```

##### Générer le fichier JAR

```
mvn clean package
```


##### Exécuter le JAR

```
cd front/target  
java -jar ./front-1.0-jar-with-dependencies.jar
```

#### Récupération du JAR (Méthode 2)

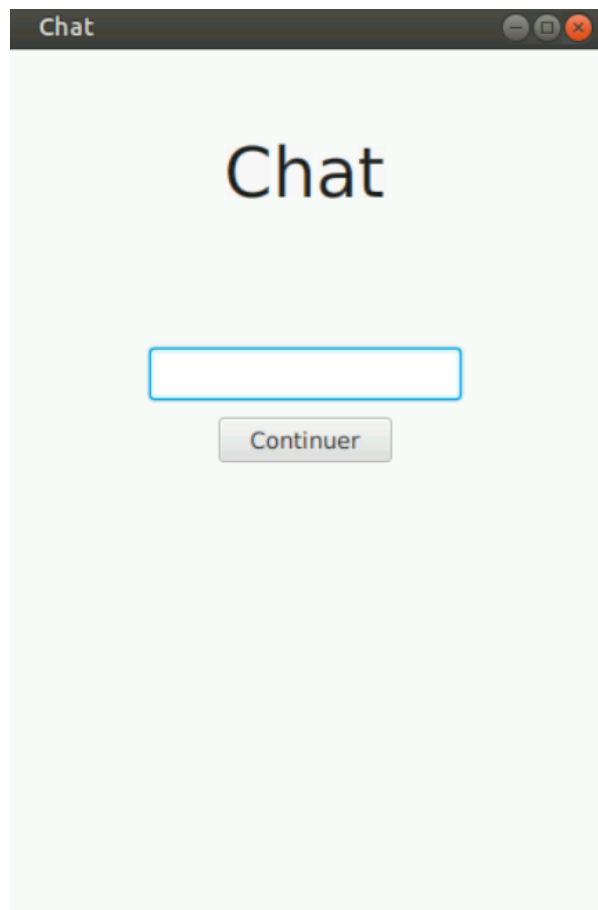
- Télécharger l'artifact généré du dernier commit dans le git action en cliquant dessus
- Extraire le JAR dans le dossier de votre choix
- Exécuter le jar :

```
java -jar ./front-1.0-jar-with-dependencies.jar
```



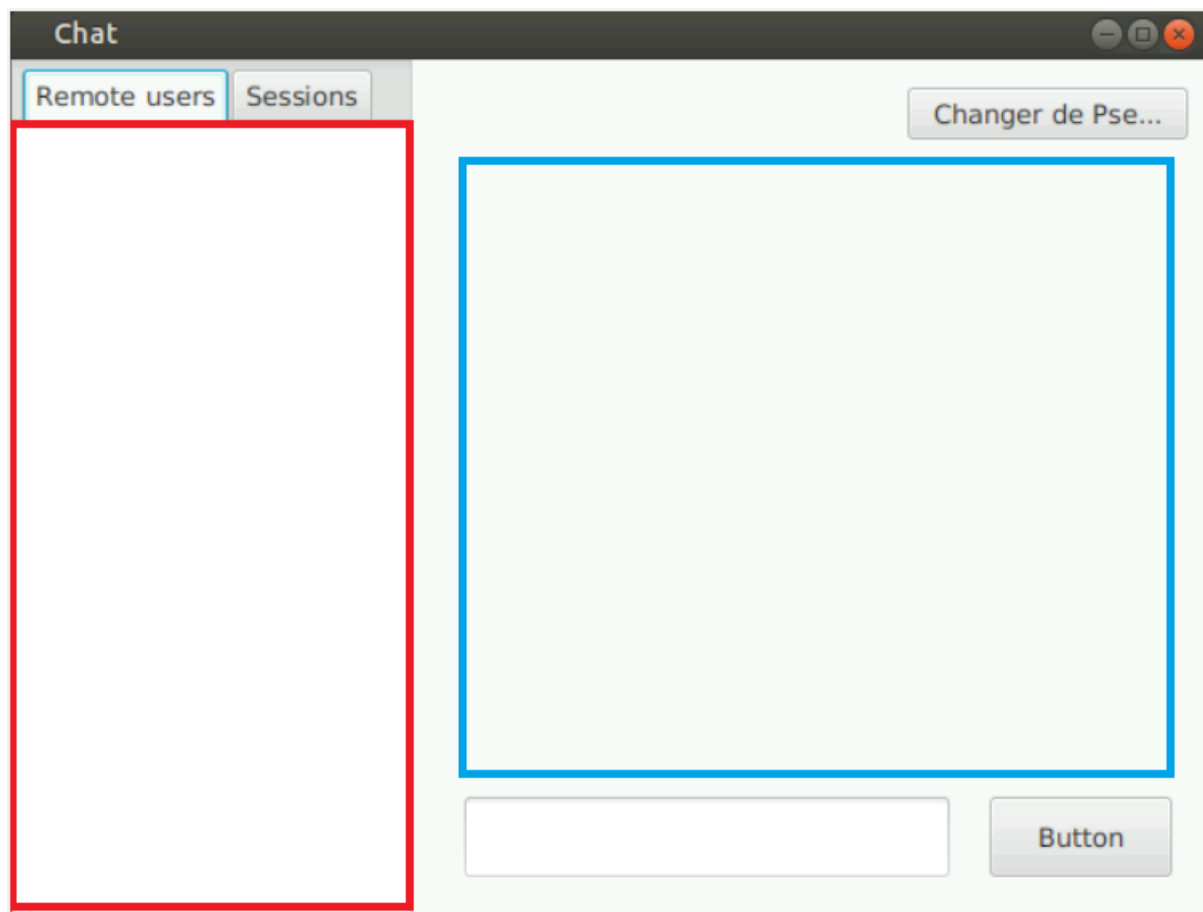
## **b. Utilisation du système**

Après avoir effectué l'installation de l'application, on peut la lancer pour pouvoir l'utiliser. Au lancement, une fenêtre apparaîtra et aura la forme suivante :



Au travers de cette page, nous avons une unique possibilité : entrer un pseudo pour accéder au système de clavardage. Si le pseudonyme choisis est déjà utilisé par un autre utilisateur, alors la fenêtre se mettra à jour et affichera : « Pseudo déjà utilisé ».

Une fois que nous choisis un pseudonyme adéquat, nous pouvons entrer directement dans la fenêtre principale qui va nous permettre de démarrer des sessions de clavardage.




Voici l'interface générée après avoir entré le pseudonyme.  
Dans celle-ci, on a accès à deux listes différentes :

- Celle des « Remote Users », qui nous donnera la liste des utilisateurs connectés.
- Celle des « Sessions », qui nous donnera la liste des utilisateurs pour lesquels une session de clavardage est engagée.

Ces deux listes apparaîtront sur le rectangle « rouge » en appuyant sur les onglets « Remote Users » et « Sessions » pour obtenir l'une ou l'autre.

Dans le rectangle « bleu », on pourra observer la discussion entre deux utilisateurs. Cette discussion sera générée en cliquant sur un des « Remote Users » présent dans la liste.



Par la suite, les messages pourront s'écrire dans la zone de texte associée et en cliquant sur « Button », nous pourrons envoyer le message qui apparaîtra donc sur la zone bleue.

En ce qui concerne la déconnexion, elle se fait simplement en quittant la fenêtre ouverte. Lors de la fermeture de la fenêtre, les autres utilisateurs sont informés automatiquement de la déconnexion de l'utilisateur concerné.

