Geley Esteban
Bordais Gabriel
4IR-C1

INSA TOULOUSE

Rapport de Projet

Système de Clavardage en réseau local

Année universitaire 2022-2023

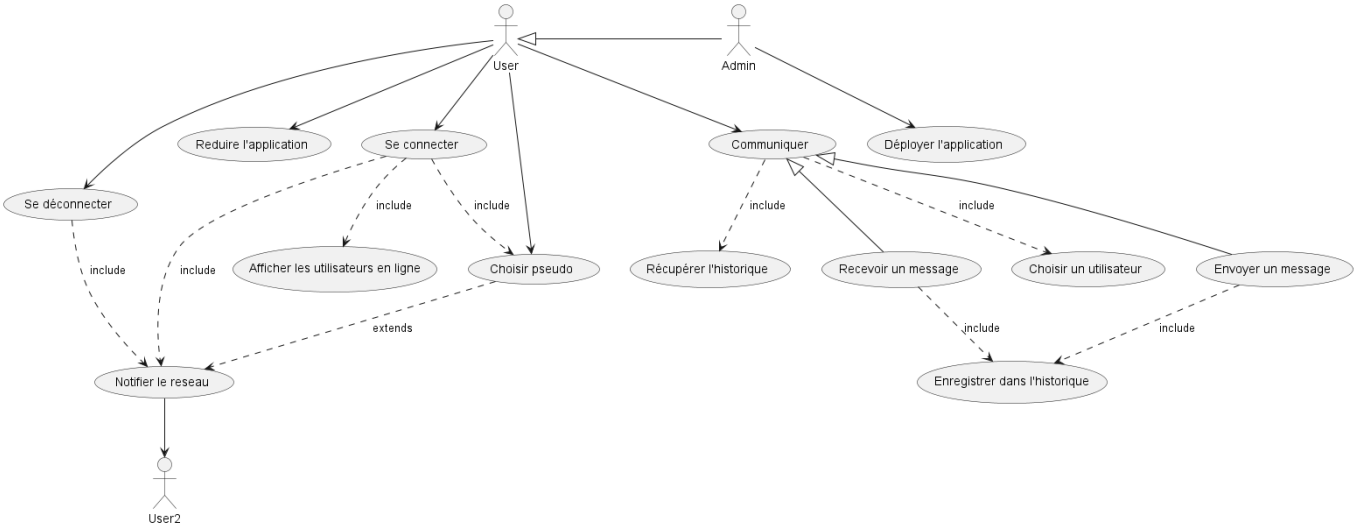
Introduction

Dans le cadre de l'UF **Conception et Programmation Avancée**, nous avons eu l'occasion de travailler sur la conception et le développement d'un **système de clavardage** (chat) en réseau local.
Un cahier des charges établissant un ensemble de fonctionnalités nous a été imposé et nous avons alors pensé, organisé et implémenté une des solutions possible pour répondre à ce dernier.
Ce rapport rassemble une description de notre projet à travers notre **conception UML**, nos **choix d'implémentation** mais aussi un **manuel d'utilisation simplifié** permettant la prise en main en bon et due forme de notre application : **Clac Chat**

Notre code est disponible sur le dépôt git suivant : [Github_Clac_Chat]

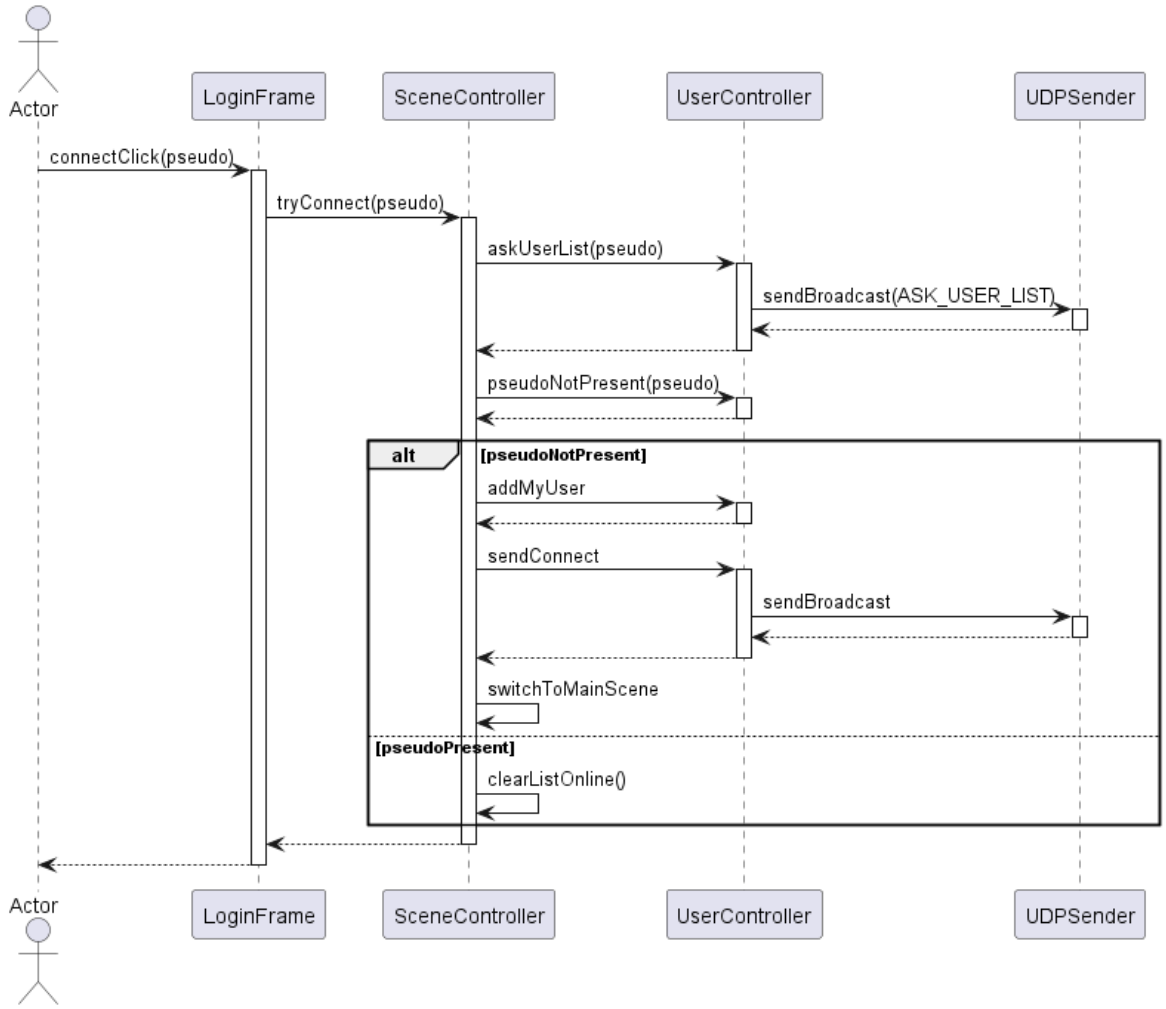
Conception et diagrammes

Diagramme des cas d'utilisation

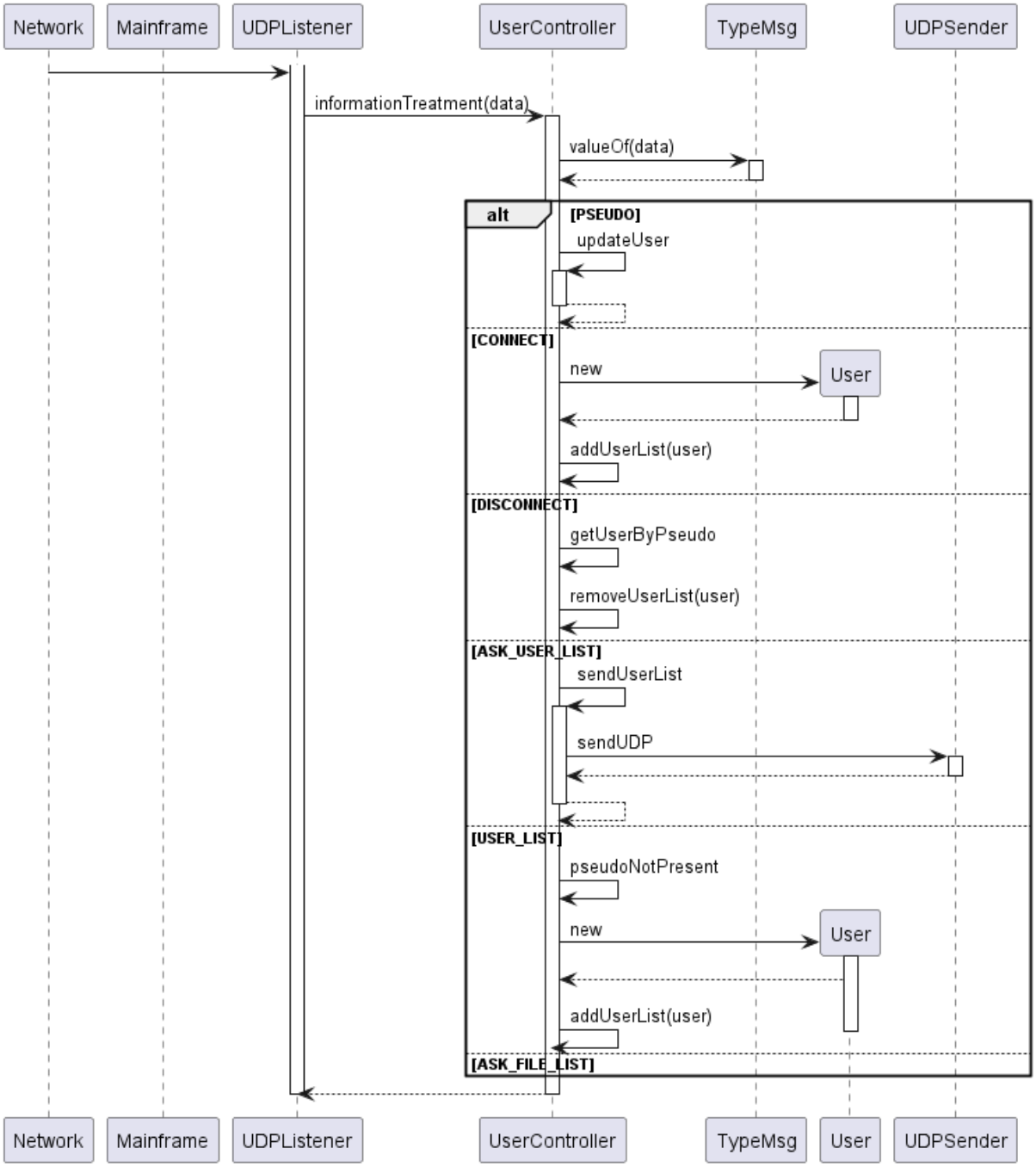


Diagrammes de séquence

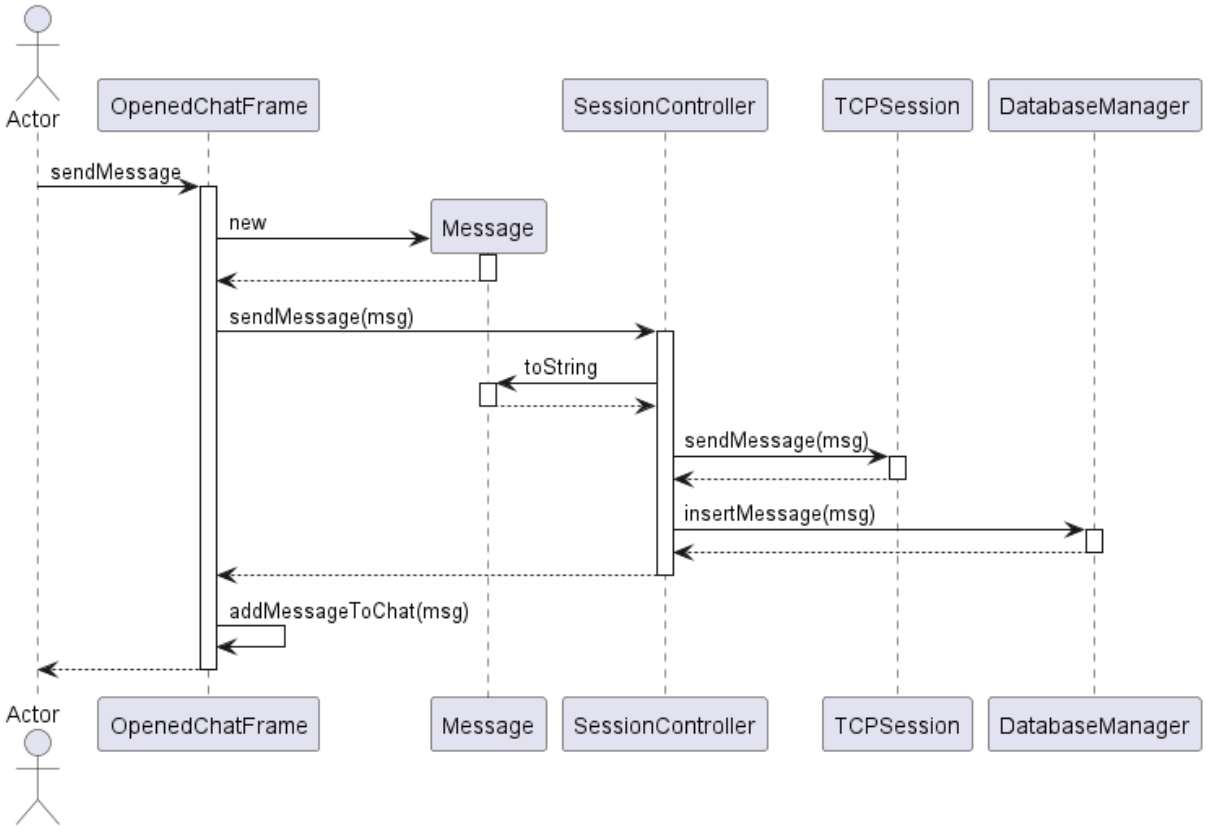
Connexion à l'application et vérification de la validité du pseudo :



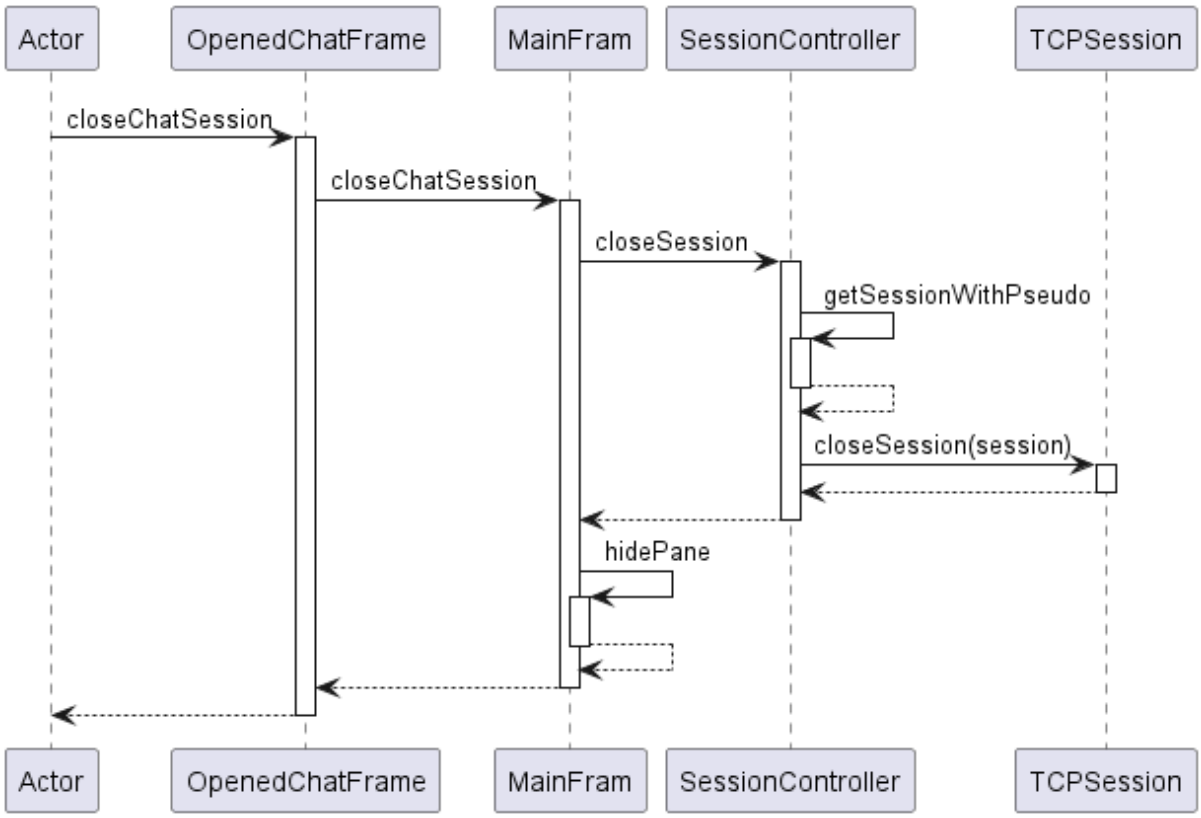
Réception et traitement de message UDP :



Envoi de message TCP :



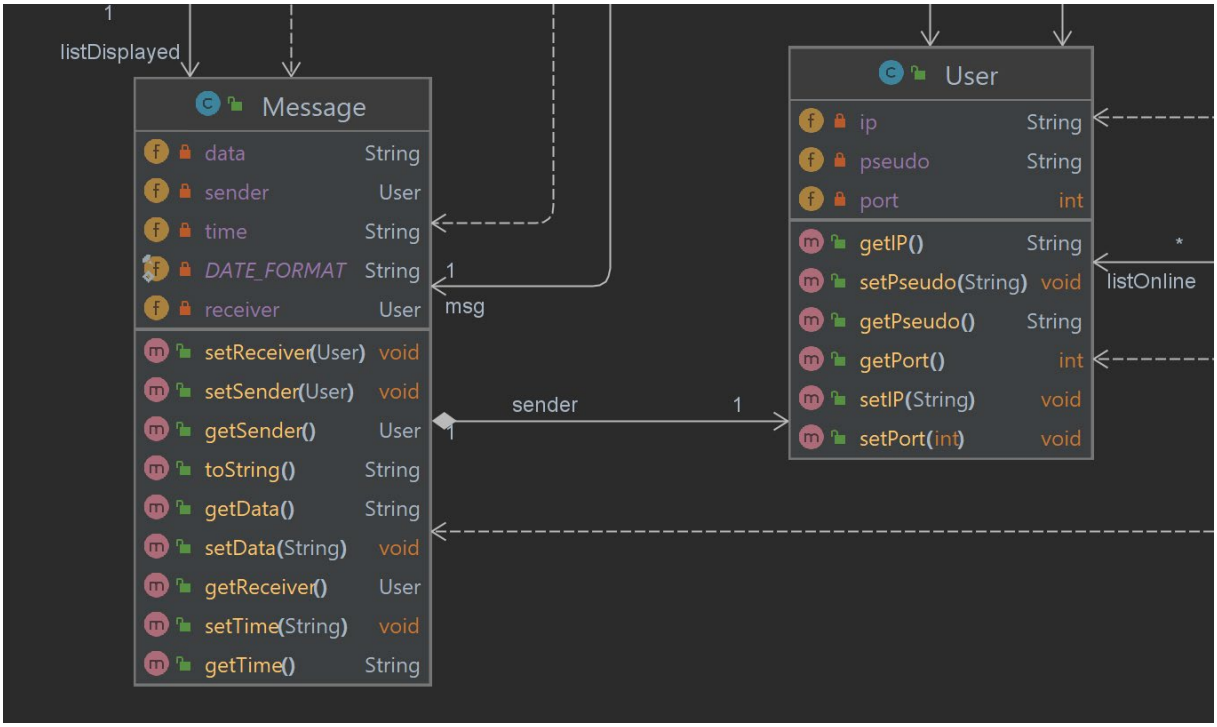
Fermeture de session et d'un thread d'écoute TCP :



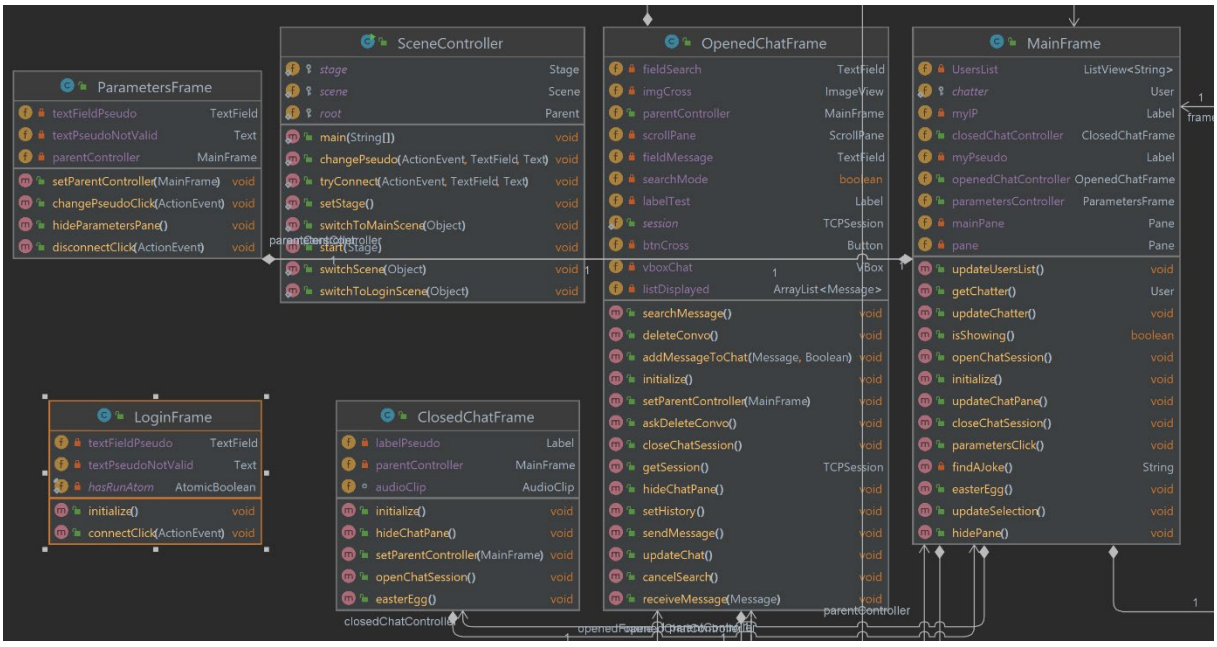
Diagrammes de classes

Le diagramme de classe complet comportant la base de donnée est disponible en annexe de ce document.

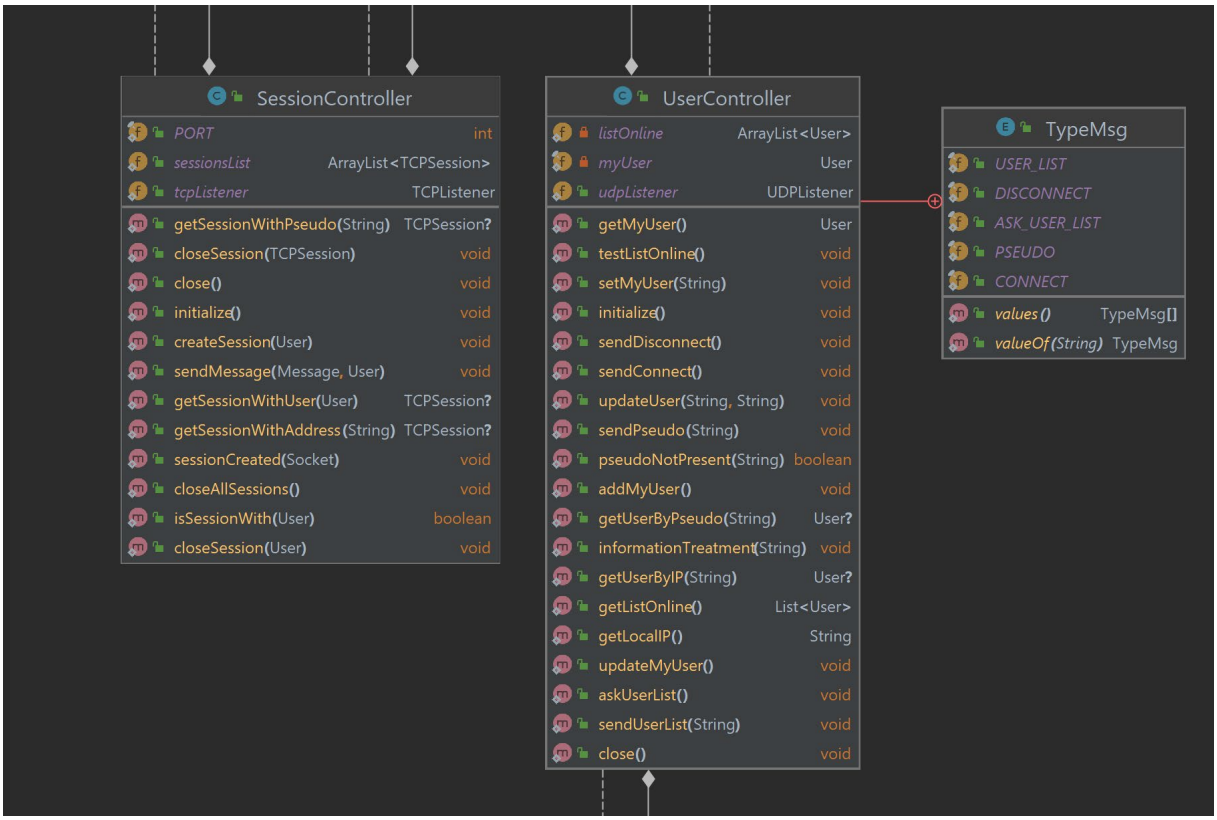
Package model :



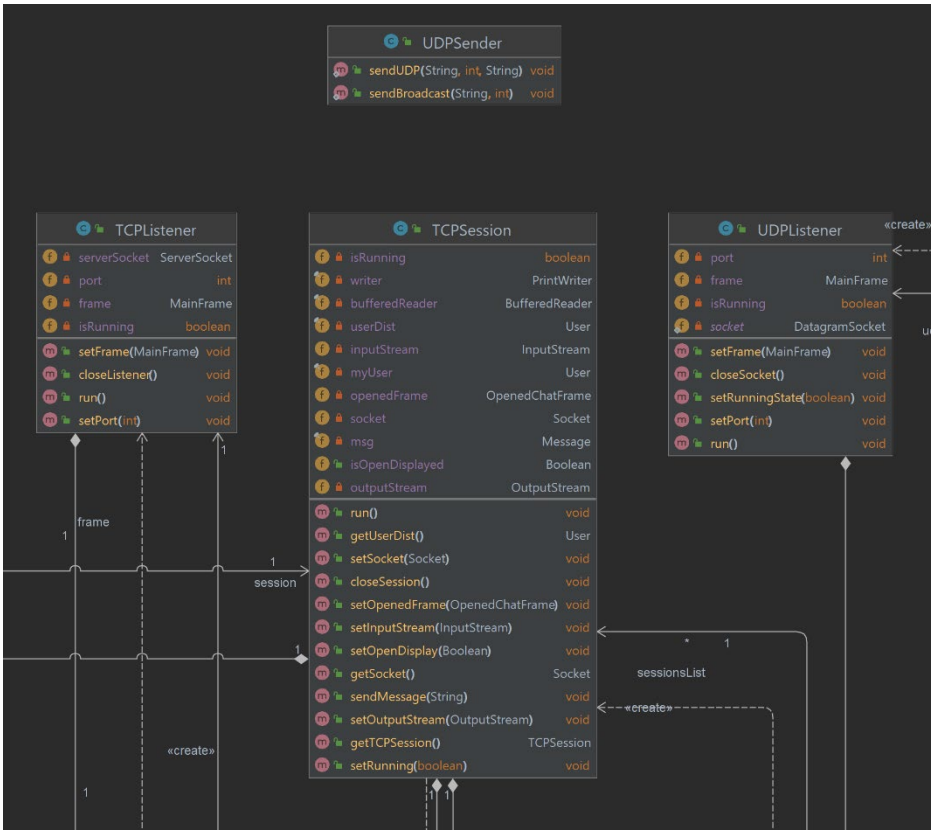
Package view :



Package controller :



Package network :



Architecture du projet

Notre projet a été organisé selon le pattern **Model-View-Controller** afin de ne pas se perdre au sein de notre implémentation. Ce dernier décompose le projet en trois principaux packages :

- Un modèle (**Model**) contenant les données à afficher, ici les messages et les users
- Une vue (**View**) contenant la présentation de l'interface graphique, ici en FXML
- Un contrôleur (**Controller**) contenant la logique concernant les actions de l'user

En plus de ces trois principaux packages, nous avons un package contenant la partie réseau (TCP / UDP) ainsi qu'un package contenant la gestion de la base de données avec des fonctions opérant des instructions SQL.

Les informations concernant l'utilisateur local ainsi que les différents utilisateurs connectés à l'application sont stockées dans le **User Controller**, quant à la liste des sessions de chat, elle est stockée dans le **Session Controller**.

De plus, notre système a été déployé sous **Maven** afin de pouvoir en simplifier le déploiement. En effet, cet outil permet de « convertir » notre projet en un fichier **.jar** sans aucun conflit et avec **peu de configuration** de la part de l'utilisateur. Maven s'occupe de dynamiquement télécharger du matériel à partir des dépôts logiciels connus. La configuration de ce dernier s'effectue dans un fichier **POM.xml**.

Choix technologiques

Réseau

L'application devant communiquer sur le **réseau local**, nous avons dû mettre en place **plusieurs protocoles** réseaux pour cette dernière.

Protocole UDP :

La première phase lors de l'utilisation de l'application est bien évidemment celle de la **connexion**. L'objectif est alors de récupérer la liste de tous les utilisateurs connectés et de les notifier de notre connexion si notre pseudo est valide (non utilisé).

Pour effectuer ces quelques manipulations, un **broadcast UDP** est utilisé. En effet, lors de la tentative de connexion, l'utilisateur effectue une **demande** de *liste Online* auprès de tous les utilisateurs et récupère celle du premier qui lui renvoie. Il la stocke alors et compare son pseudo auprès de ceux présents dans cette liste. Si l'unicité du pseudo est vérifiée, il envoie un message de **connexion** en broadcast et tous **les utilisateurs** la recevant **mettent à jour** leur *liste Online* avec ce nouvel utilisateur.

Pour les **changements de pseudo** ou les **déconnexions**, il s'agit du même principe : l'utilisateur envoie un **message en broadcast** avec en en-tête le **type d'opération** voulu

ainsi que **son pseudo** et **son adresse IP**. À la réception, les utilisateurs actualisent leur *liste Online* en conséquence.

UDP ne possédant **aucune contrainte de qualité de service**, il est parfaitement adapté pour ce type d'opération. En effet, la **connexion** est alors **rapide** et l'actualisation se fait quasiment en instantanée. Si nous avions sélectionné le protocole TCP, les contraintes de connexions rapides n'auraient pas été respectées.

Ce protocole est géré par les classes : *UserController*, *UDPListener* et *UDPSender*.

Protocole TCP :

Une fois la **connexion** au sein de l'application **effectuée** et la *liste Online* initialisée, il faut permettre à l'utilisateur de **communiquer** avec différents utilisateurs distants.

Pour cela, nous avons utilisé le protocole **TCP**. En effet, à chaque fois qu'un utilisateur décide d**'ouvrir une session de chat** avec un utilisateur distant, **une session TCP est ouverte** à l'aide de **différents threads** permettant **plusieurs sessions de chat en simultané**. Chaque session ouverte est stockée au sein de *sessionsList* et est retirée dès lors qu'elle est refermée par un utilisateur.

Nous avons décidé au sein de notre application que l'ouverture et la fermeture d'une session TCP nécessitait la demande d'un seul des deux utilisateurs ; ces actions ne nécessitent pas la validation de l'autre utilisateur et sont effectuées instantanément.

TCP possédant une **qualité de service en fiabilité**, il est parfaitement adapté pour l'envoi et la réception de message. En effet, nous cherchons ici un **échange de messages sans perte**.

Ce protocole est géré par les classes : *SessionController*, *TCPListener* et *TCPSender*.

Base de données

Afin de **stocker les différents messages** au sein de l'application, nous avons décidé de mettre en place une **base de données décentralisées** pour des questions de simplification. En effet, nous avons utilisé **SQLite**, un module simple mais très efficace.

Ainsi, **chaque utilisateur** possède **sa propre base de données** sur son ordinateur où il y stocke ses différentes conversations.

Chaque table correspond à une conversation qui a **pour nom l'adresse IP de l'utilisateur distant**, étant donné que nous avons défini comme identifiant unique l'adresse IP.

Chaque table est composé de 5 colonnes :

- l'**index** au sein de la table,
- l'**adresse IP** de l'envoyeur,
- l'**adresse IP** du receveur,
- le **contenu du message**,
- la **date** du message.

IP_@IPuserDist				
index	@IP sender	@IP receiver	text	time

Figure 1: Structure d'une table de la base de donnée

Alors, tous les messages sont décomposés et stockés de cette manière.

Interface graphique :

Concernant l'**interface graphique**, nous avons tout d'abord **designé notre interface** à l'aide de **Figma**, puis nous nous sommes servis du plugin **JavaFX** ainsi que du logiciel **SceneBuilder** afin de l'importer au sein de notre projet.

La mise en place de cette interface fut assez simple car **SceneBuilder** nous a permis de **placer tous les éléments** à l'aide des **informations données par Figma** : la taille, la position ainsi que le code CSS correspondant afin d'avoir l'application la plus esthétique possible.

Ainsi, **chaque document FXML** généré grâce à SceneBuilder **possède un contrôleur** en **Java**. Chaque **élément** peut alors être **exploité** à l'aide de **JavaFX**. Concernant le **design de ces éléments**, il est explicité directement **au sein du fichier FXML** auquel il appartient ou alors **au sein du fichier** *style.css* dans le dossier *ressources*.

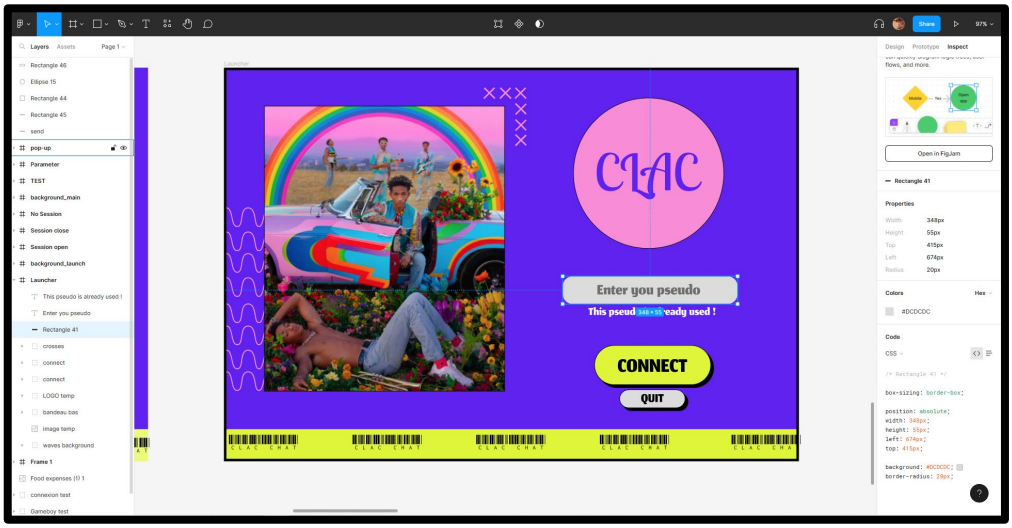


Figure 2: Exemple de design réalisé sur Figma (Interface de connexion)

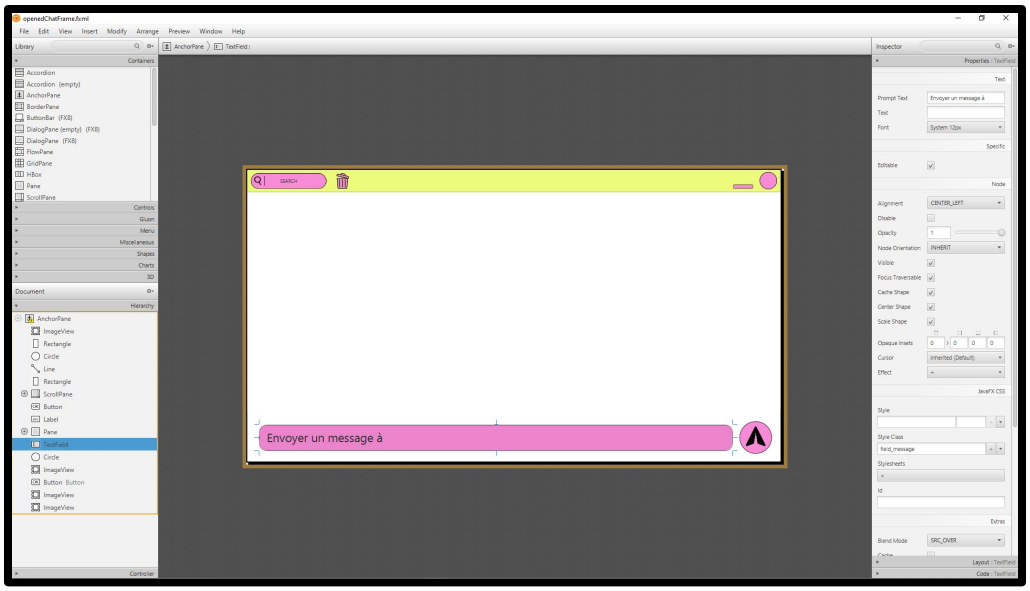


Figure 3: Exemple de mise en place des éléments sur SceneBuilder (Onglet de chat | Session ouverte)

Procédures d'évaluation et de test

Le déploiement de l'application ne s'est quant à lui pas effectué avec Jenkins comme préconisé mais avec GitHub Actions. Cela permet d'effectuer un paquetage de l'application lorsque l'on pousse des modifications sur le dépôt Git.

Nous n'avons pas utilisé de procédure de test automatique durant le développement du projet car elles auraient été trop compliquées à mettre en place dans le cas de tests réseaux. Nous avons préféré effectuer des tests à la main sous forme de scénarios. Par exemple pour les tests de **TCP**, **UDP** ainsi que le la **base de données**, nous les avons effectués **indépendamment les uns des autres** au sein du fichier *testMain.java* situé au sein du package *test* en décrivant des scénarios tels que la connexion d'un utilisateur et en constatant les résultats avec des écritures dans le terminal.

L'**interface** quand à elle a été testée en **situation réelle** au **fur et à mesure** de son développement. Le principe du **FXML** et de **JavaFX** étant **inconnu pour nous**, cela nous a pris un peu de temps afin de maîtriser ce fonctionnement et nous avons passé du temps à réussir à **imbriquer les onglets de chat** ainsi que **l'onglet des paramètres** au sein de **l'interface principale**.

Une fois l'**interface terminée** et **tous les éléments interconnectés**, nous avons testé l'application en réseau local avec **deux utilisateurs**. Nous avons eu beaucoup de soucis concernant l'**actualisation en temps réel** de l'interface. En effet, la liste des utilisateurs connectés s'actualise à chaque connexion/déconnexion/changement de pseudo et les sessions de chat doivent s'actualiser à chaque message reçu. De nombreuses heures de recherches dans beaucoup de direction plus tard, **nous avons réussi**.

Une fois ces tests à deux utilisateurs effectués, nous pensions avoir fini mais nous avons décidé d'effectuer des **tests à trois utilisateurs**. Nous nous sommes alors rendu compte

que l'interface **arrivait à effectuer l'actualisation en temps réel des messages que pour la première session de chat ouverte**. Nous avons alors donc dû modifier quelques paramètres afin que cela fonctionne pour toutes les sessions de chat.

Quelques **autres bugs** ont bien sûr été détectés **au sein de nos différentes manipulations** de test de l'interface finale mais ces derniers étaient la plupart du temps assez **rapidement corrigeables**.

Procédure d'installation et de déploiement

Prérequis :

- Java 11 ou version ultérieure
- Maven
- JavaFX
- Une connexion réseau pour les utilisateurs

Installation :

- Téléchargez ou clonez le dépôt git : [Github_Clac_Chat]
- Utilisez Maven pour compiler et construire le projet avec la commande :

```
mvn clean install
```

- Exécutez l'application avec la commande :

```
java -jar target/clac.jar
```

Manuel d’utilisation simplifié

Dans cette partie, nous allons vous présenter le fonctionnement général de Clac Chat. À noter que, en plus des différentes options présentées ci-après, **2 Easter eggs** sont cachés au sein de l’application, à vous de les trouver !

Interface de connexion (Login)

Au lancement de l’application, une fenêtre s’ouvre : l’**interface de connexion**. Grâce à cette interface, vous pouvez vous authentifier au sein de l’application en entrant le pseudo désiré et en appuyant sur le bouton *CONNECT* .

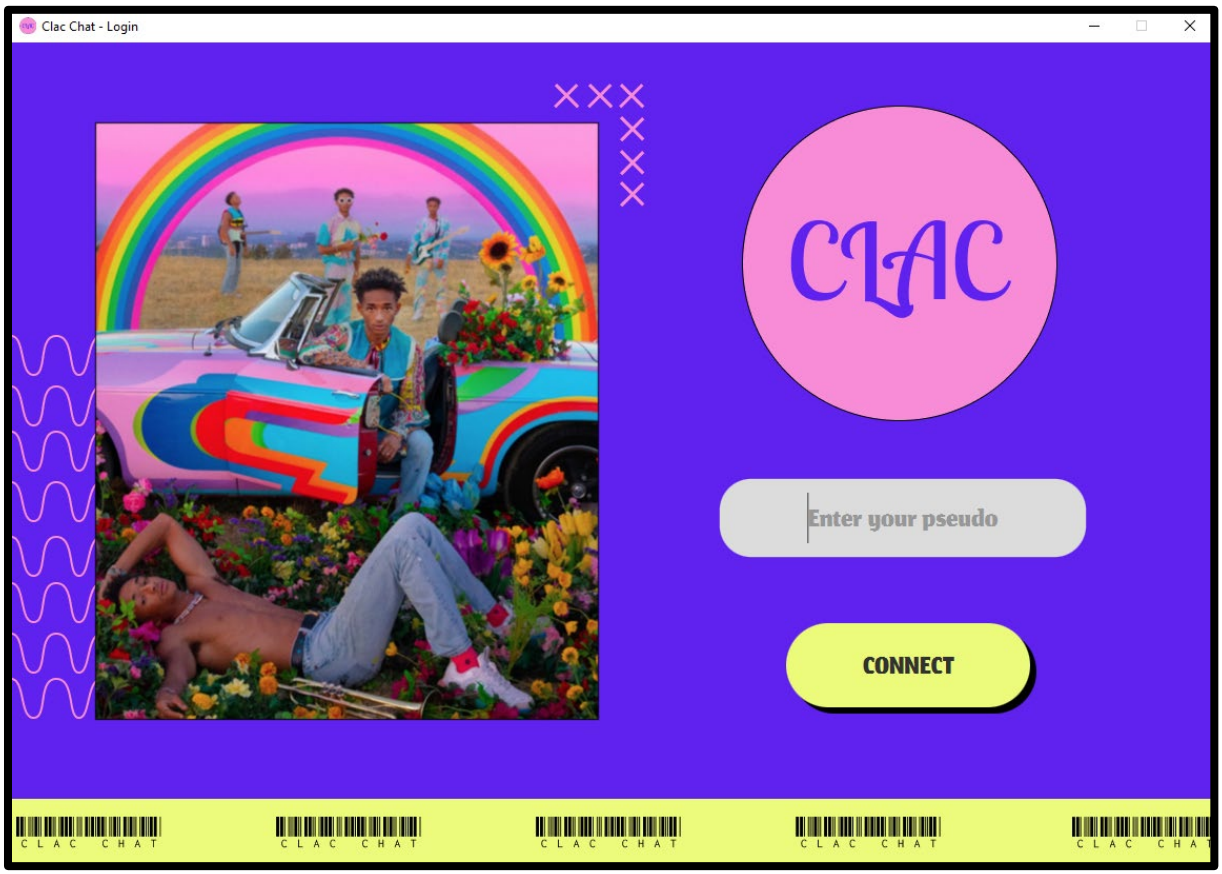


Figure 4: Interface de connexion

Une fois le bouton *CONNECT* pressé, l’application vous informera si votre **pseudo** n’est **pas valide** : si ce dernier comporte des **espaces**, des **caractères spéciaux**, **plus de 15 caractères** ou bien encore s'il est tout simplement **vide**. De plus, l’application interdira l’utilisation d’un **pseudo déjà utilisé** par un membre en ligne, il vous en informera alors également. Dans tout ces cas de figure, un *message d’erreur* sera affiché et vous pourrez à nouveau rentrer un pseudo.

Une fois que vous avez choisi un pseudo valide, l’application vous dirigera vers l’**interface principale** de l’application.

Interface principale (Main Page)

Figure 5: Interface principale

Voici l'**Interface principale**, contenant de nombreux éléments :

- En haut à gauche : vos informations personnelles (pseudo & IP)
- En dessous de vos informations personnelles : la *liste des utilisateurs connectés*
- En bas à gauche : le bouton *PARAMETERS*
- En haut à droite : notre mascotte

(Le reste n'est que pure décoration, vous ne pouvez pas interagir avec)

Si vous cliquez sur le *pseudo d'un utilisateur dans la liste*, cela ouvre un **onglet de chat** en distinguant si la **session est ouverte ou non**.

Si vous cliquez sur le bouton *PARAMETERS*, cela ouvre l'**onglet paramètres**.

Onglet de chat | session fermée

Lorsque vous cliquez sur le *pseudo d'un utilisateur*, si ni vous ni lui n'a ouvert la session de chat, cet onglet s'ouvre.

Vous pouvez voir sur cet onglet quelques éléments :

- Au centre : notre mascotte
- En bas : un message informatif ainsi que le bouton *OPEN CHAT SESSION*
- En haut à droite : le bouton de *fermeture d'onglet**

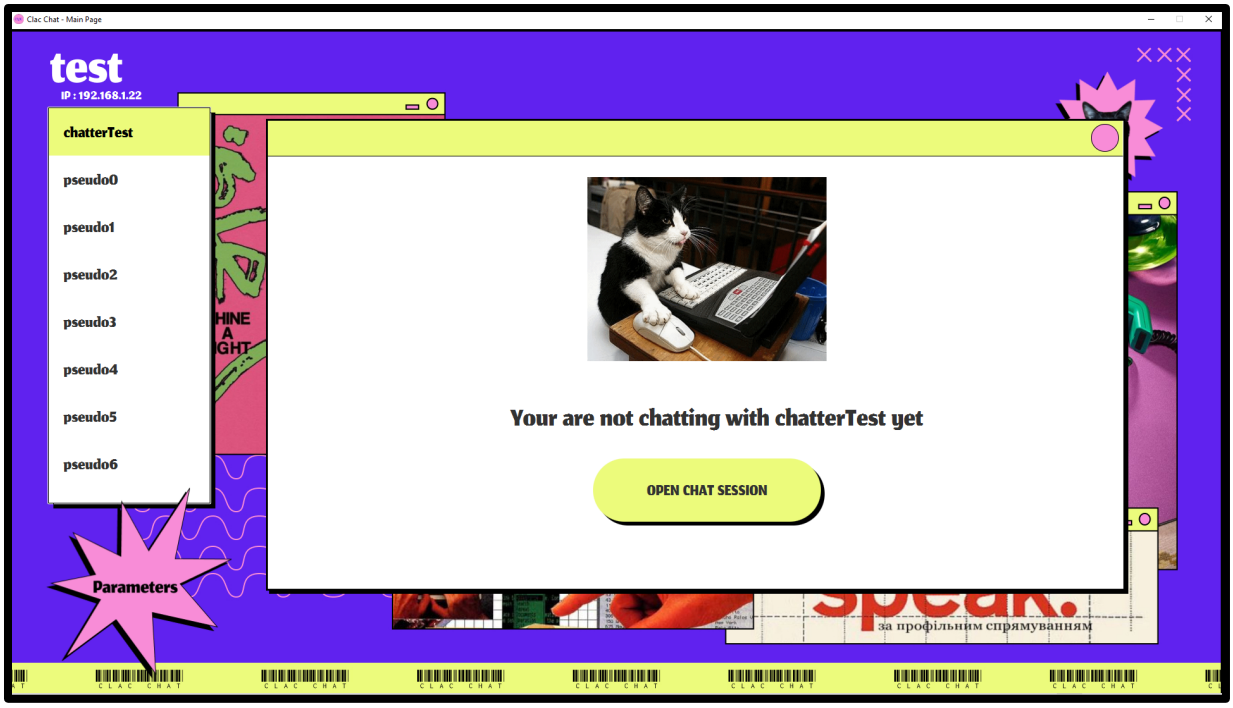


Figure 6: Onglet session de chat fermée

Si vous appuyez sur le bouton de *fermeture d'onglet* (o) ou si vous sélectionnez un *autre utilisateur*, cet onglet se **fermera**, et la **session restera fermée**.

En revanche, si vous appuyez sur le bouton *OPEN CHAT SESSION*, l'onglet s'actualisera pour laisser place à l'onglet **onglet de chat | session ouverte**. De même si l'utilisateur distant appuie de son côté sur ce *bouton*, l'actualisation s'effectuera en temps réel.

Onglet de chat | session ouverte

Si la **session de chat** avec l'utilisateur sélectionné a été **ouverte** par lui ou par vous, cet onglet s'ouvre.

On peut y apercevoir :

- Au centre : l'historique des messages, avec le nom de l'envoyeur ainsi que la date et l'heure d'envoi.
- En bas : La *zone de texte* pour écrire un message, ainsi que son *bouton d'envoi*.
- En haut à gauche : la *zone de recherche de messages*
- A droite de la zone de recherche : la *corbeille* de suppression d'historique
- En haut à droite : les boutons de *minimisation et de fermeture d'onglet*

Ainsi, pour **envoyer un message**, il vous suffit de l'**écrire dans la zone de texte** adaptée puis d'appuyer sur le **bouton d'envoi** ou bien d'appuyer sur la touche *ENTREE* de votre clavier. A chaque **envoi ou réception de message**, l'affichage s'actualisera en temps réel.

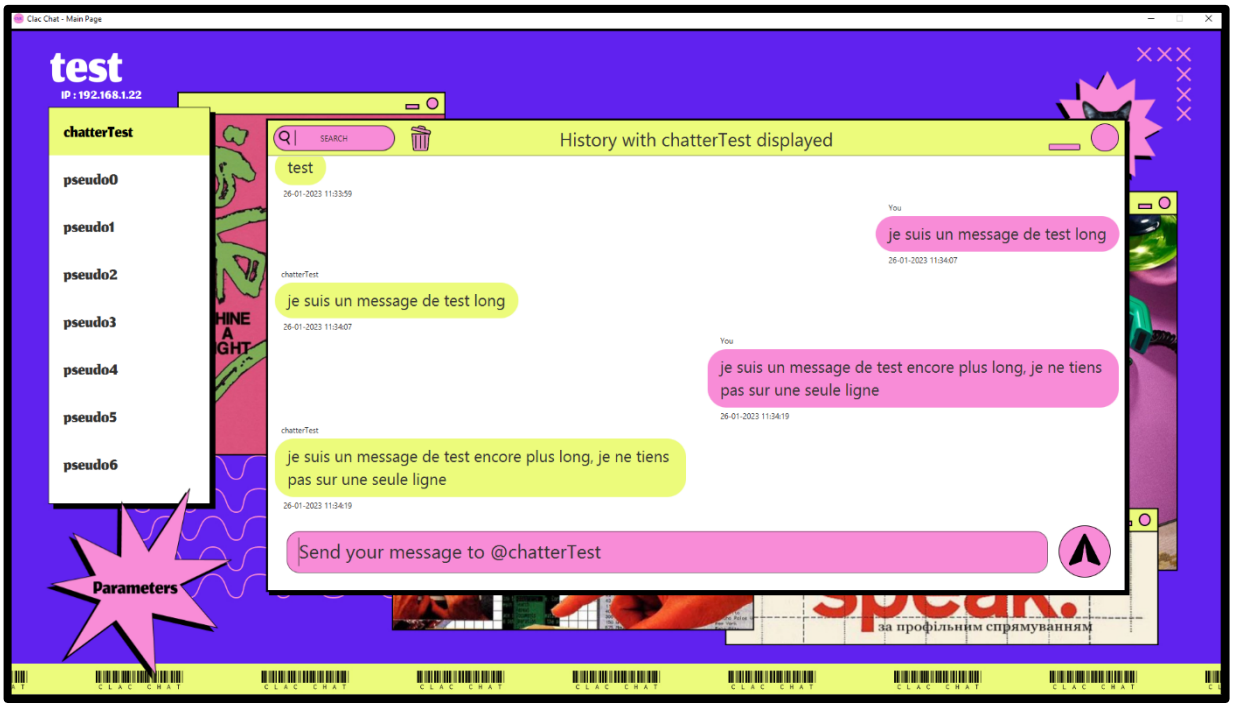


Figure 7: Onglet session de chat ouvert

Pour **rechercher** un mot ou une phrase clef au sein des messages, vous pouvez renseigner le champ *SEARCH* et appuyer sur la *loupe*. L'onglet de chat n'**affichera** alors que **les messages contenant la chaîne de caractère renseigné**. Pour **annuler la recherche**, il vous suffira d'appuyer sur la *croix* qui apparaît lorsque qu'une recherche est effectuée.

Figure 2: Champ "Search" avec le mot "message" renseigné

De plus, si vous le voulez, vous avez la possibilité de **supprimer l'intégralité de l'historique** de conversation en appuyant sur la *poubelle rose*. Cette dernière vous ouvrira un **pop-up de confirmation** étant donné que cette **action** est **sans retour**, aucun historique supprimé ne pourra être restauré.

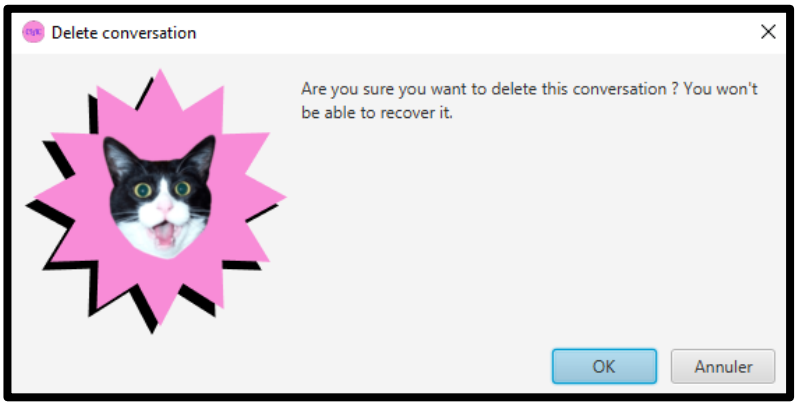


Figure 3: Pop-up de confirmation de suppression d'historique

Pour finir, vous avez la possibilité de **minimiser la discussion** ou bien de la **fermer complétement** en appuyant sur les *icônes cor*respondantes. (respectivement – et o) Si vous décidez de **fermer la discussion**, cela **fermera la session de chat** pour vous comme pour l'utilisateur distant. Vous retournerez alors sur l'**interface principale** vierge et l'**utilisateur distant** se retrouvera sur l'**onglet de chat | session fermée**.

Si vous décidez de **parler à un autre utilisateur**, il vous suffira de **cliquer** sur **un autre pseudonyme** dans la *liste des utilisateurs connectés*. Cela **ne fermera pas votre session de chat** mais la **minimisera** seulement, vous pourrez alors **rouvrir cet onglet** en cliquant sur le **pseudo de l'utilisateur distant correspondant**.

Onglet paramètres

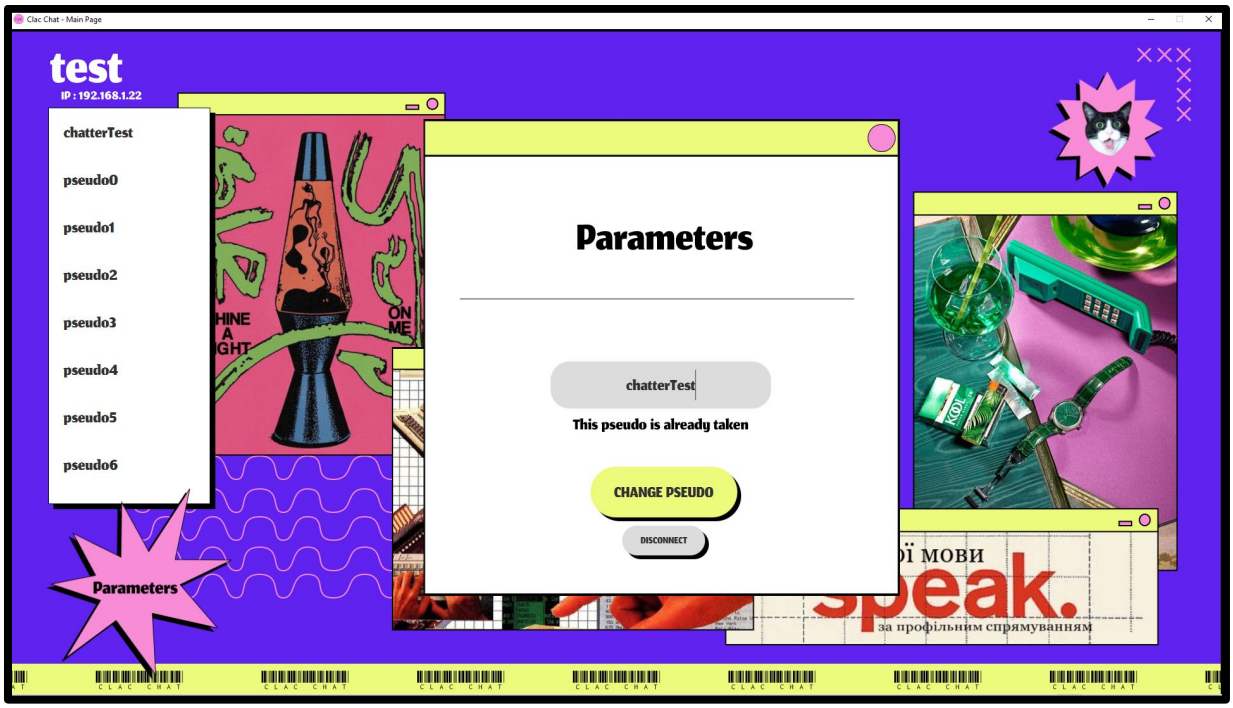


Figure 10: Onglet paramètres, avec un changement de pseudo impossible

Au sein de l'**onglet paramètres**, vous avez la possibilité de **changer de pseudo** ou bien de vous **déconnecter**. Les **règles** concernant les **pseudos** sont **les mêmes** qu'au sein de l'**interface de connexion**. L'ouverture de cet onglet enlèvera la sélection d'un utilisateur distant et **minimisera l'onglet de chat** si il y en a un d'ouvert.

L'action de **changement de pseudo** ou bien de **déconnexion** actualisera en temps réel l'interface de tous les utilisateurs en ligne.

- Si vous **changez de pseudo** : celui-ci sera actualisé au sein de leur liste d'utilisateurs connectés et au sein de leur onglet session chat (fermée ou ouverte) s'ils vous ont sélectionné dans la liste.
- Si vous vous **déconnectez** : vous serez retiré de leur liste d'utilisateurs et leur onglet session chat (fermée ou ouverte) sera fermée s'ils vous ont sélectionné dans la liste.

Conclusion

Dans l'ensemble, notre projet de développement de cette application a été un exercice stimulant et instructif qui nous a permis de découvrir beaucoup de technologies et de pratiques alors inconnues comme par exemple l'utilisation des threads, l'affichage graphique ou bien l'intégration avec Maven. Nous avons suivi un cahier des charges établissant un ensemble de fonctionnalités et avons réussi à concevoir et développer une solution fonctionnelle dont nous sommes satisfaits.

Les principales difficultés que nous avons rencontrées ont été lors de l'utilisation de dépendances telles que JavaFX ainsi que tout le paramétrage allant avec, et non pas des difficultés purement liées au développement. Ces difficultés ont été notre principal défi lors de ce projet, mais nous avons finalement réussi à surmonter ces obstacles grâce à des recherches approfondies.

Annexes

Annexe 1 : diagramme de classe complet