

RAPPORT DE PROJET

**Chama BENLAFKIH, Clémence LEMEILLEUR,
Carina MALLEVILLE**

Promo 56, Année 2021/2022 – 4IR-SI-A1

*“Système de Clavardage distribué interactif multi-
utilisateur temps réel”*

Octobre 2021- Janvier 2022

Encadrant : S.Yangu

RAPPORT DE PROJET

Chama BENLAFKIH, Clémence LEMEILLEUR,
Carina MALLEVILLE

Promo 56, Année 2021/2022 – 4IR-SI-A1

*“Système de clavardage distribué interactif multi-utilisateur
temps réel”*

Octobre 2021- Janvier 2022

Encadrant: S.Yangui

Table des matières

I- CONCEPTION DU SYSTEME	2
A. <i>DETAIL DES CHOIX DE CONCEPTION</i>	2
B. <i>ORGANISATION DE LA CONCEPTION</i>	2
C. <i>DIAGRAMMES DE CONCEPTION.....</i>	6
1- <i>Diagramme des Use Case</i>	6
2- <i>Diagramme des Classes</i>	6
3- <i>Diagramme de Séquence.....</i>	7
4- <i>Diagramme Composite</i>	8
II- ARCHITECTURE ET CHOIX TECHNOLOGIQUES DU SYSTEME.....	9
A. <i>BASE DE DONNEES</i>	9
B. <i>GUI</i>	9
C. <i>AUTRES LIBRAIRIES UTILISEES</i>	10
III- PROCEDURE D'EVALUATION ET DE TESTS.....	10
A. <i>CHOIX DES DIFFERENTS TESTS.....</i>	10
B. <i>APPLICATION DE CES TESTS ET RESULTATS</i>	10
IV- PROCEDURE D'INSTALLATION ET DE DEPLOIEMENT	11
A. <i>CONFIGURATIONS PREALABLES</i>	11
B. <i>COMMANDES DE LANCEMENT.....</i>	11
C. <i>SCRIPT</i>	11
V- MANUEL D'UTILISATION SIMPLIFIE	12
A. <i>DEMARRAGE DE L'APPLICATION.....</i>	12
B. <i>CONNEXION.....</i>	12
1- <i>Inscription.....</i>	12
2- <i>Connexion à un compte existant</i>	12
C. <i>CHAT</i>	13
D. <i>DECONNEXION</i>	13
TABLE DES ILLUSTRATIONS	14
TABLE DES ANNEXES	15

I- Conception du système

A. Détail des choix de conception

Dans la conception de notre projet nous avons dû faire certains choix afin de respecter au maximum le cahier des charges tout en étant capable de rendre le travail demandé par le client en temps voulu.

De notre côté, notre service de messagerie permet à deux employés d'une même entreprise de communiquer en temps réel. Pour cela il suffit d'un pseudo et un mot de passe, que l'on donne lors de la première connexion et qui sera demandé à chaque connexion. Ensuite, si le pseudo est valide, l'utilisateur est créé si c'est la première connexion ou alors se connecte à son compte existant.

Une fois connecté, l'utilisateur peut profiter pleinement du service de messagerie avec l'ouverture d'une fenêtre sur laquelle plusieurs actions lui sont proposées :

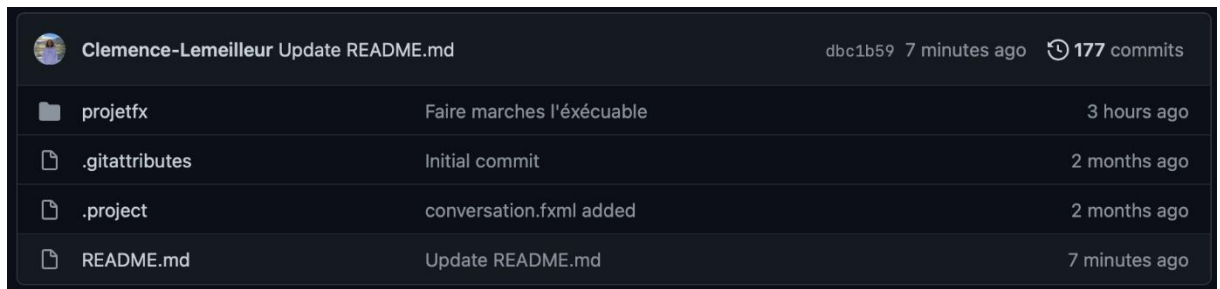
- Il lui est possible de **modifier son pseudo**.
- Il a accès à la **liste des personnes en ligne** qu'il peut rafraichir si besoin, même si celle-ci se met automatiquement à jour lors de la connexion et déconnexion d'un utilisateur.
- Il peut, en cliquant sur le pseudo d'un des utilisateurs en ligne, dont la liste s'affichera sur la gauche de son écran, **démarrer une conversation** en envoyant un message.
- Il peut, une fois ses actions terminées, **se déconnecter** grâce à un bouton sur la fenêtre principale. Son nom disparaîtra alors de la liste des personnes connectées.
- Il est également possible **d'effectuer une recherche dans l'historique** des messages d'une conversation, afin de pouvoir retrouver un message à partir d'un mot clé ou d'une lettre.

Si deux utilisateurs ont déjà eu une conversation par le passé sur l'application de clavardage, elle sera visible lors de l'ouverture de la fenêtre de discussion avec la personne concernée. On y retrouvera l'ensemble des messages ainsi que la date et l'heure à laquelle ils ont été envoyés.

Si un utilisateur change de pseudo, la liste des utilisateurs actifs est automatiquement mise à jour comme précisé précédemment. En revanche, les utilisateurs du système ne sont pas prévenus lors d'un changement de pseudo. Cela ne nous a pas paru être un problème dans le sens où cette application de chat est proposée à une entreprise, dans le cadre professionnel. Nous partons donc du principe qu'il n'y aura pas de blagues de la part des employés se donnant un pseudo qui ne permettrait pas de les identifier de manière claire.

B. Organisation de la conception

Au niveau de notre dépôt **Git**, nous l'avons organisé en plusieurs dossiers dans une même branche. Il est possible de voir l'organisation de ce dernier ci-dessous :




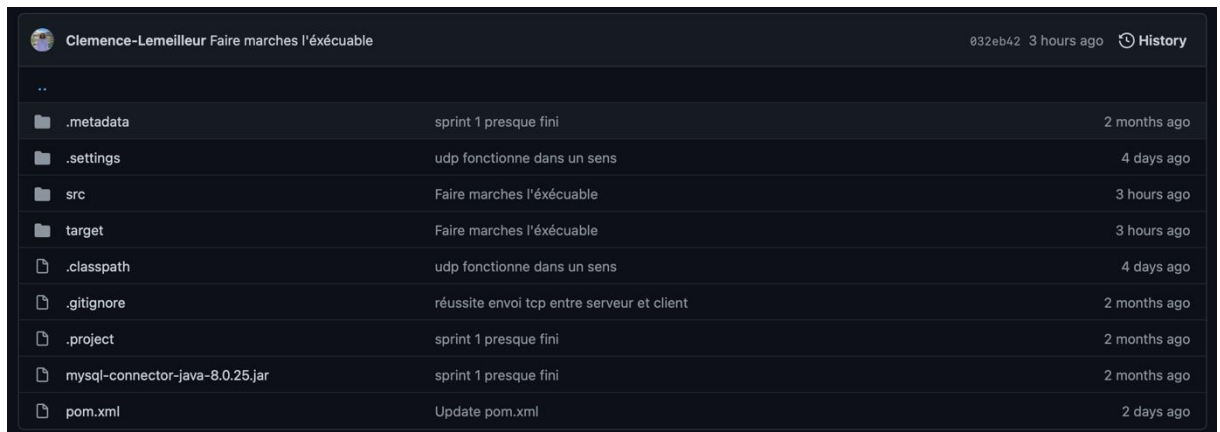
	Clemence-Lemeilleur Update README.md	dbc1b59 7 minutes ago	🕒 177 commits
📁	projetfx	Faire marches l'exécutable	3 hours ago
📄	.gitattributes	Initial commit	2 months ago
📄	.project	conversation.fxml added	2 months ago
📄	README.md	Update README.md	7 minutes ago

Figure 1 : Organisation du Github




	Clemence-Lemeilleur Faire marches l'exécutable	032eb42 3 hours ago	🕒 History
..			
📁	.metadata	sprint 1 presque fini	2 months ago
📁	.settings	udp fonctionne dans un sens	4 days ago
📁	src	Faire marches l'exécutable	3 hours ago
📁	target	Faire marches l'exécutable	3 hours ago
📄	.classpath	udp fonctionne dans un sens	4 days ago
📄	.gitignore	réussite envoi tcp entre serveur et client	2 months ago
📄	.project	sprint 1 presque fini	2 months ago
📄	mysql-connector-java-8.0.25.jar	sprint 1 presque fini	2 months ago
📄	pom.xml	Update pom.xml	2 days ago

Figure 2 : Organisation du Projet

Comme visible sur la première capture d'écran, notre dépôt contient un README afin de faciliter la reprise du travail par d'autres développeurs par la suite, si nécessaire, mais également de donner toutes les informations nécessaires à la configuration, l'installation et l'utilisation du service de messagerie.

Sur la deuxième figure on peut reconnaître le dossier source qui est suivi du main, java, mais contenant également le dossier de tests qui seront détaillés plus bas.

Nous avons volontairement choisi de ne pas utiliser de branches puisque nous ne maîtrisons pas totalement l'utilisation de l'outil d'intégration continu Git au début du projet. Il nous paraissait dommage de passer du temps sur cet aspect-là au lieu de le consacrer entièrement au développement de l'application de messagerie. Notre dépôt comporte donc qu'une seule branche principale, la branche « main ».

Afin d'organiser le partage des tâches et d'avoir un suivi de notre avancement nous avons utilisé un **Jira**. Nous avons découpé le travail total en quatre sprints.

Pour chacun d'entre eux, nous définissons les users storys, qui étaient ensuite divisés en plusieurs tâches. A chacune des storys, était associé un nombre de points afin de nous permettre de gérer combien de temps nous consacrerions à chacune d'entre elles.

Régulièrement nous avons fait le point entre nous pour classer les tickets en fonction de ce qui était terminé, en cours ou non commencé.

Le premier sprint était consacré aux fonctionnalités basiques de création de login, interface de connexion. Ce qui constitue la première approche de l'utilisateur.

Nous avons décidé de faire ce choix puisque notre objectif était de fournir à notre client quelque chose de fonctionnel, de visuel, le plus rapidement possible. Ainsi lors d'un

premier bilan nous pouvons montrer au client une première interface, avec une connexion fonctionnelle.

Tickets terminés [Afficher dans le navigateur de tickets](#)

Clé :	Résumé :	Type de ticket :	Epic :	État :	Responsable :	Story points
POO-6	Create login	Story		TERMINÉ(E)		5
POO-16	Verification valid login	Tâche		TERMINÉ(E)		-
POO-22	Enregistrer login dans BDD	Tâche		TERMINÉ(E)		-
POO-7	Change login	Story		TERMINÉ(E)		3
POO-31	Créer interface connexion	Tâche		TERMINÉ(E)		-
POO-32	Créer interface accueil avec champ changement login	Tâche		TERMINÉ(E)		-
POO-33	Vérifier que l'utilisateur existe	Tâche		TERMINÉ(E)		-

Figure 3 : Sprint 1

Le deuxième était axé sur la suite de l'expérience qu'aura l'utilisateur à savoir, voir la liste des utilisateurs connectés et avoir une interface de conversation.

Tickets terminés [Afficher dans le navigateur de tickets](#)

Clé :	Résumé :	Type de ticket :	Epic :	État :	Responsable :	Story points
POO-9	See online users	Story		TERMINÉ(E)		3
POO-15	Create an active users list	Tâche		TERMINÉ(E)		-
POO-26	Afficher liste des utilisateurs actifs	Tâche		TERMINÉ(E)		-
POO-34	créer l'interface Conversation qui s'ouvre lorsque l'on cli...	Tâche		TERMINÉ(E)		-

Figure 4 : Sprint 2

Le troisième nous a permis de nous consacrer sur la suite logique à savoir permettre à l'utilisateur de communiquer avec un autre. Cela passe par le fait d'avoir une connexion TCP et UDP fonctionnelle, pouvoir envoyer des messages après l'ouverture de la connexion et finir par une fermeture de connexion. Pour cela, il y a eu tout une dimension d'affichage des messages à gérer également dans la fenêtre de conversation.

Tickets terminés [Afficher dans le navigateur de tickets](#)

Clé :	Résumé :	Type de ticket :	Epic :	État :	Responsable :	Story points
POO-10	Start chat	Story		TERMINÉ(E)		8
POO-35	demande de connexion TCP client	Tâche		TERMINÉ(E)		-
POO-36	création de thread conversation par le TCP server	Tâche		TERMINÉ(E)		-
POO-37	Création de la classe message	Tâche		TERMINÉ(E)		-
POO-38	Envoi du message d'un user à l'autre	Tâche		TERMINÉ(E)		-
POO-39	End connexion	Tâche		TERMINÉ(E)		-
POO-40	gestion d'udp	Tâche		TERMINÉ(E)		-
POO-43	Afficher le message reçu sur l'interface	Tâche		TERMINÉ(E)		-
POO-45	Recherche dans l'historique	Story		TERMINÉ(E)		-

Figure 5 : Sprint 3

Et le dernier avait pour but de gérer les accès concurrents grâce aux threads. En effet, il nous était au début impossible d'envoyer 2 messages d'affilés par un même utilisateur. Nous devions forcément attendre une réponse avant de continuer la discussion. Cela était dû à un problème de threads et notre 4^{ème} et dernier sprint fût consacré à le résoudre.

Ce dernier sprint comporte également la création des tests qui seront détaillés plus bas.

Tickets terminés						Afficher dans le navigateur de tickets
Clé :	Résumé :	Type de ticket :	Epic :	État :	Responsable :	Story points
POO-44	UDP	Story		TERMINÉ(E)		-
POO-41	envoi d'un message quand on change le pseudo et chan...	Tâche		TERMINÉ(E)		-

Figure 6 : Sprint 4

Pour finir, avec **Jenkins**, nous avons réalisé trois projets Maven. Nous avons tout d'abord, créé un projet Maven destiné à compiler le code que l'on a appelé CompilePOO. Puis, nous avons créé un autre projet Maven pour construire le fichier .war, appelé BuildPOO. Nous avons finalement terminé par le projet de test, TestPOO, qui a pour but d'exécuter les tests unitaires présents dans notre code.

Nous avons relié nos projets Maven à notre dépôt git pour plus de fluidité. Nous avons également configuré Jenkins de sorte que tous les projets s'exécutent les uns après les autres. Voici l'organisation visuelle des trois projets :

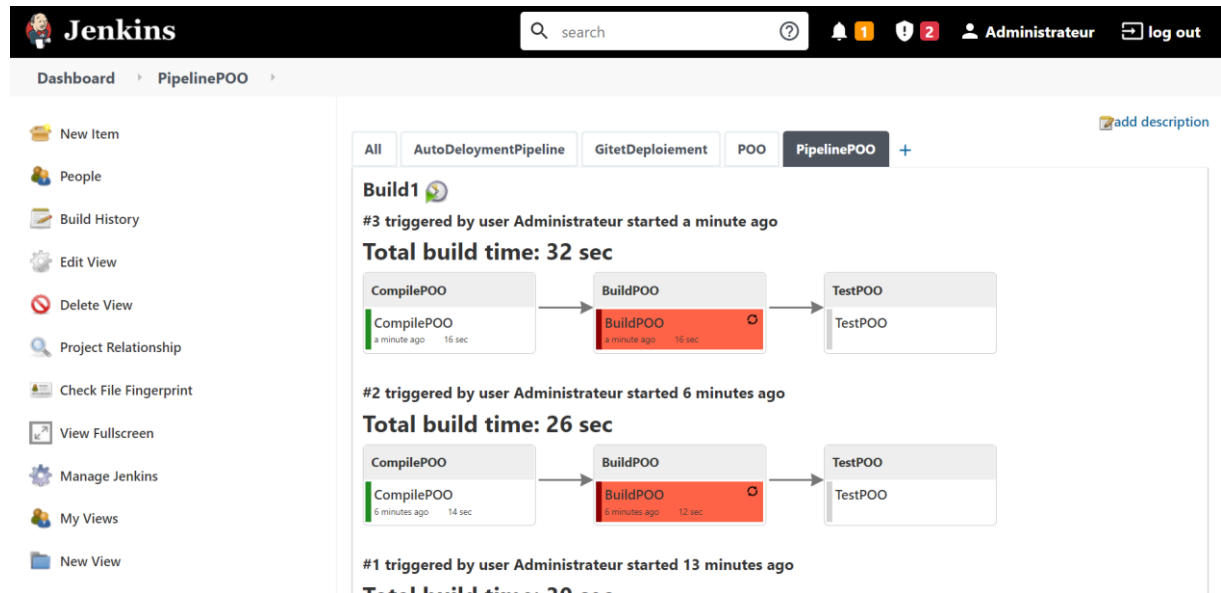


Figure 7 : Jenkins

C. Diagrammes de conception

1- Diagramme des Use Case

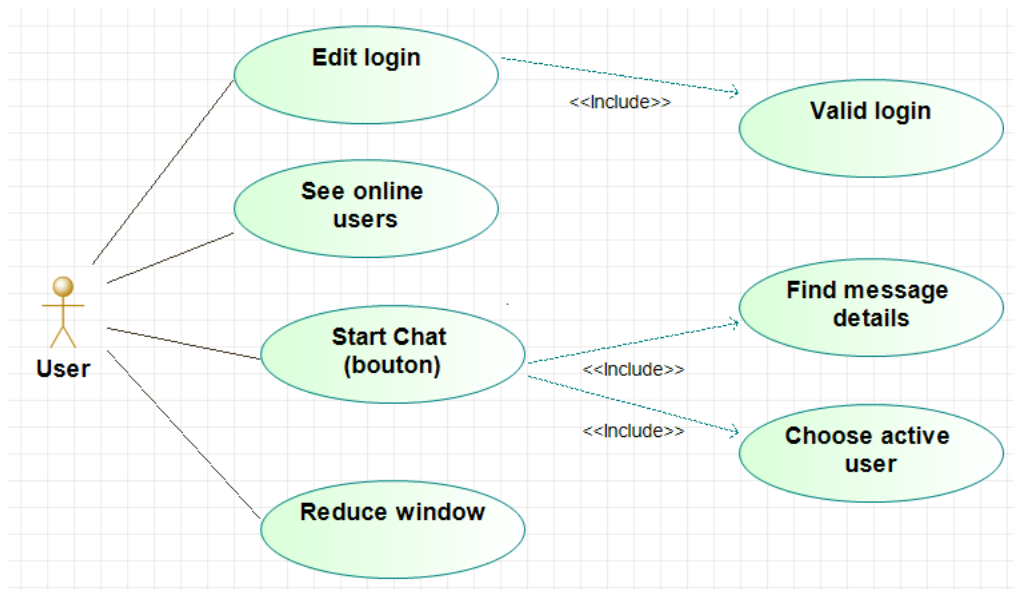


Figure 8 : Diagramme des Use Case

2- Diagramme des Classes

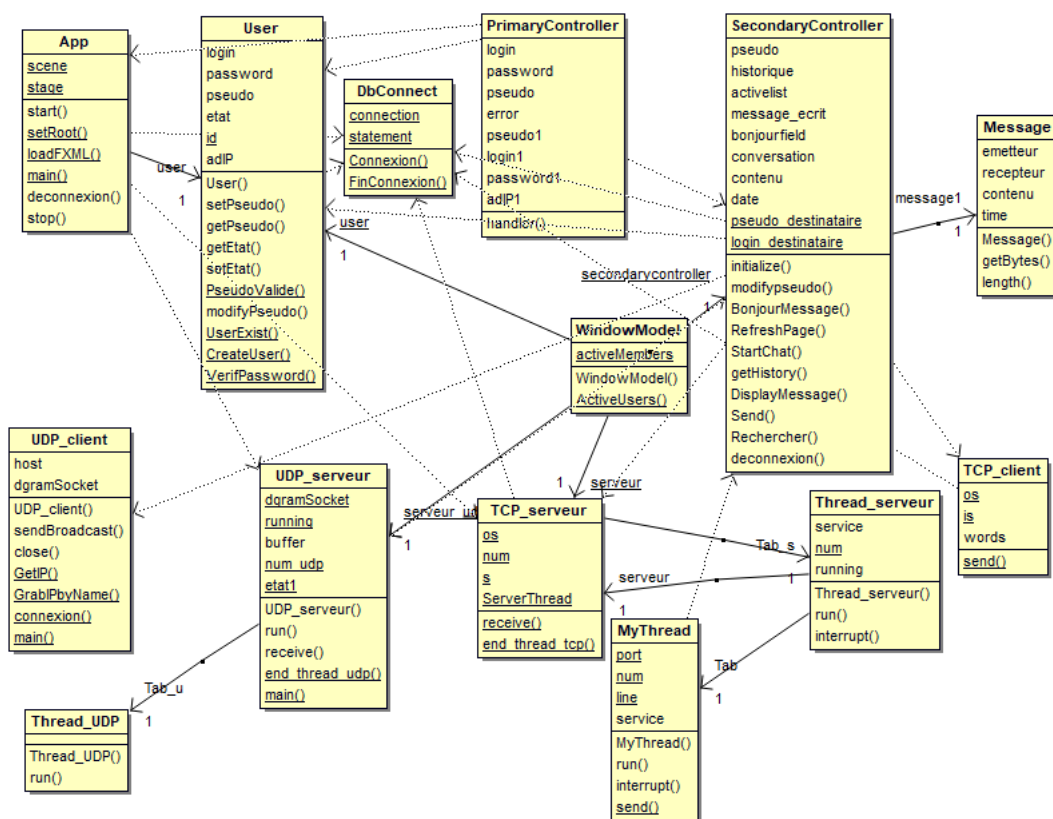


Figure 9 : Diagramme de Classes

3- Diagramme de Séquence

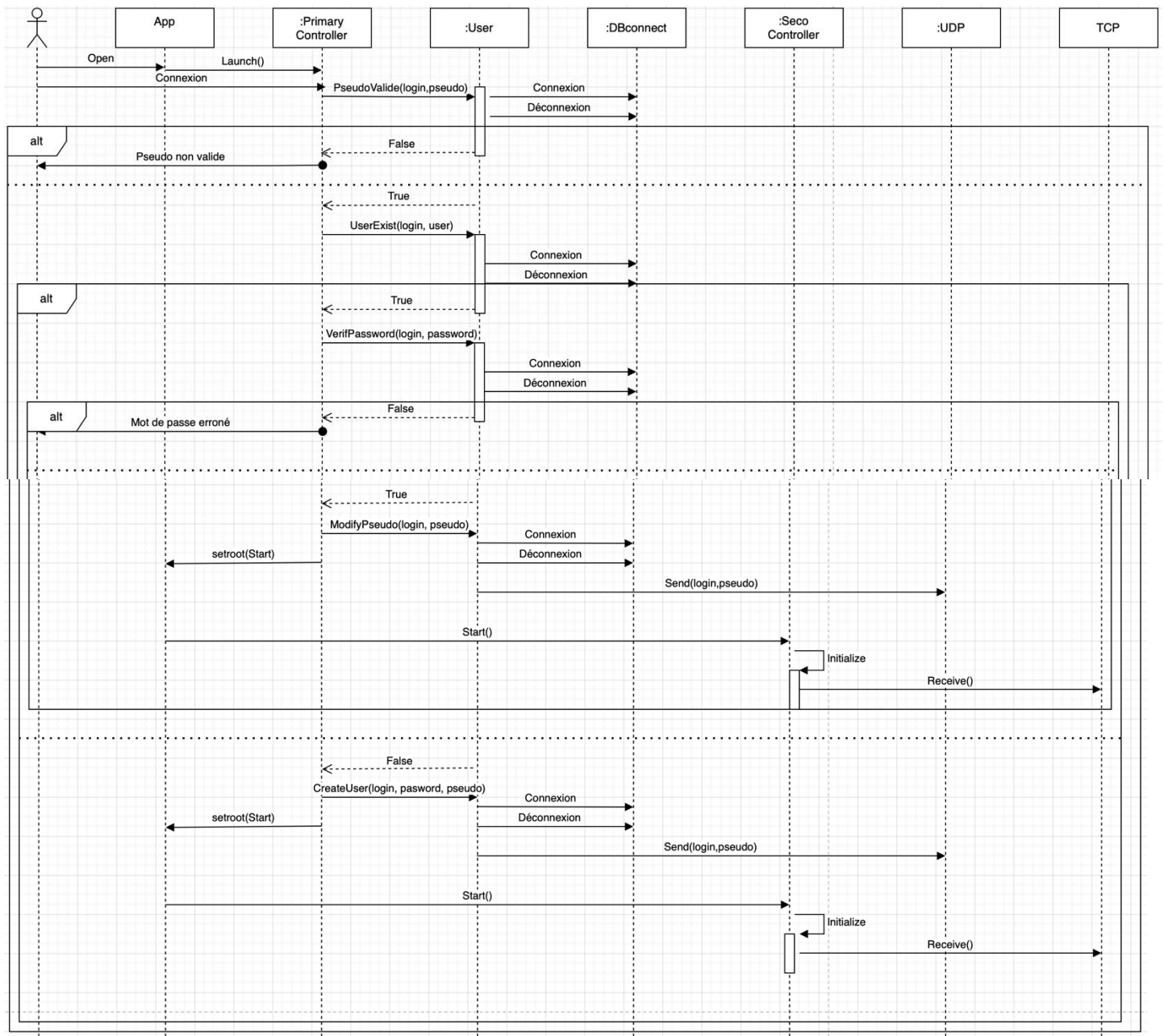


Figure 10 : Diagramme de Séquence général

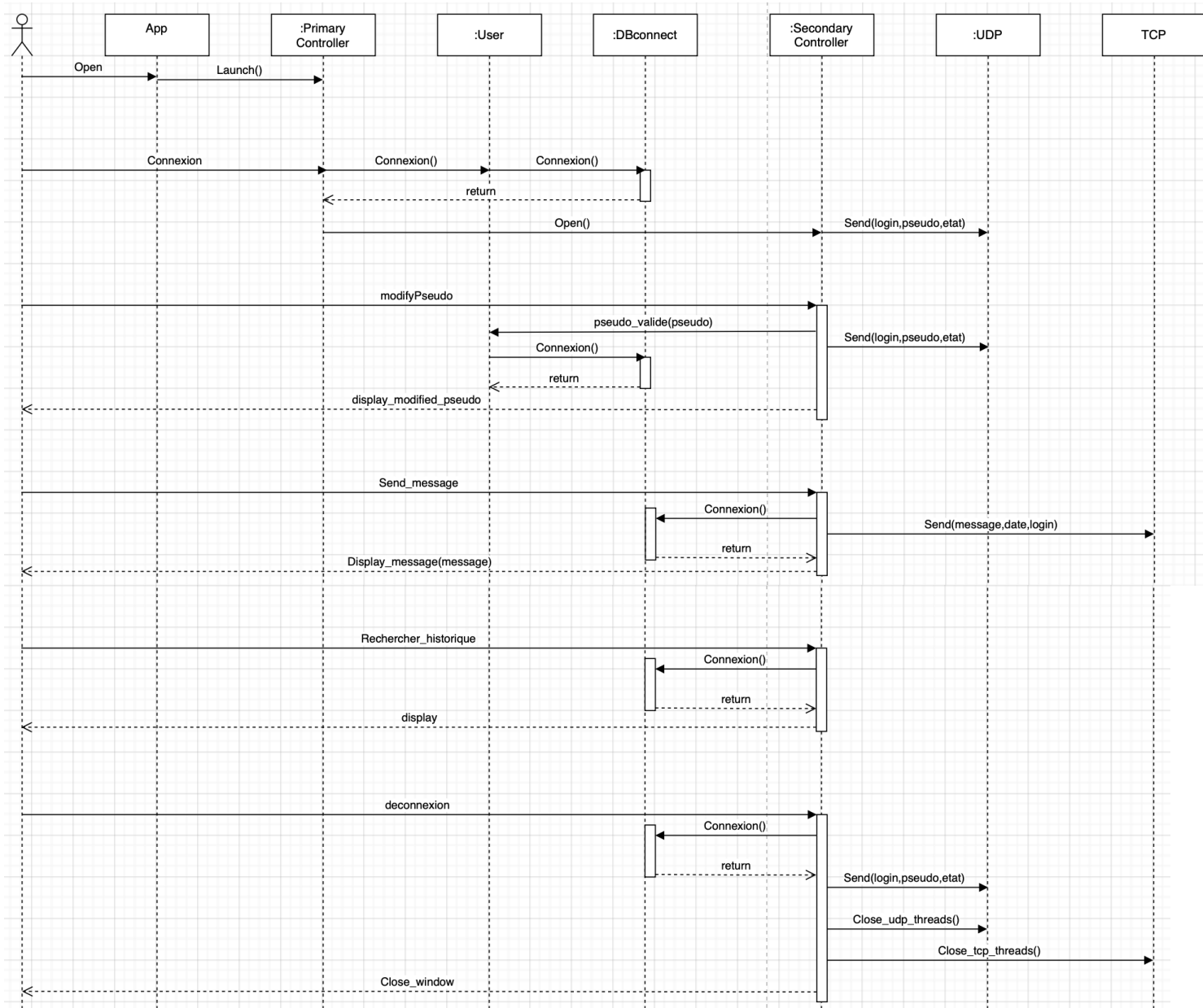


Figure 11 : Diagramme de Séquence de Connexion

4- Diagramme Composite

Nous n'avons pas jugé essentiel de représenter le diagramme composite étant donné qu'il n'apporte aucune information à notre conception, dû au fait que nous n'avons ni classe mère, ni classe fille.

II- Architecture et choix technologiques du système

A. Base de données

Afin de permettre une traçabilité des conversations, une sauvegarde des messages et un enregistrement des utilisateurs nous avons choisi d'utiliser une base de données qui serait sur le serveur de l'entreprise utilisant le service de messagerie.

Lors de la création d'un compte utilisateur, les informations de login, mot de passe et pseudo seront stockées dans cette base de données et vérifiées lors de chaque connexion. Elles pourront bien évidemment être mises à jour lors d'un changement de pseudo par exemple.

Cette base de données contient également un tableau d'objets messages. L'objet contient les attributs de destinataire, émetteur, et date de l'envoi. En effet, pour afficher un historique de conversation entre deux personnes nous devons parcourir toute cette liste en sélectionnant les objets dont les émetteur et destinataire correspondent aux 2 personnes de la conversation. De ce côté-là ce n'est pas optimal en termes de recherche dans une liste. Elle est parcourue plusieurs fois et, en fonction de sa longueur, cela peut ralentir le système lorsqu'une quantité remarquable de données y sera stockée.

Une solution aurait pu être de créer un tableau à double entrée comprenant d'une part l'émetteur et d'une part le destinataire. Nous aurions donc dû simplement accéder à 2 cases du tableau pour avoir l'ensemble de la discussion en remplaçant tous les messages par ordre de date. Cette solution nous a paru plus complexe à mettre en place bien que nous ayons essayé. Nous avons donc préféré nous concentrer sur l'expérience utilisateur et donc rendre une application fonctionnelle même si non optimale.

B. GUI

Pour l'aspect visuel de notre application nous avons utilisé les librairies Java FX que nous avons mises en place grâce à l'application Scene Builder. Elle permet de construire une scène directement en visuel et d'exporter le code correspondant afin de l'intégrer dans notre projet par la suite.

Nous avons utilisé le pattern MVC (Model View Controller) afin de permettre la meilleure expérience utilisateur possible :

- Le Model est WindowModel.

Une classe modèle est présente et comporte les données partagées par les différentes interfaces.

- Le Controller est représenté par les controllers.

Nous avons deux classes controllers qui nous permettent d'utiliser et d'exploiter ce qui se trouve dans ces fichiers fxml.

- L'Interface est assurée par les fichiers fxml.

Les fichiers fxml correspondant au code mis en place grâce à cette application sont disponibles dans le répertoire des ressources. Ces fichiers représentent donc les interfaces, le visuel, ce qui sera visible de l'utilisateur et ce avec quoi il va interagir. Nous récupérons donc ces actions pour pouvoir les exploiter et nous affichons les informations qu'il doit avoir en sortie pour pouvoir répondre à son besoin.

C. Autres librairies utilisées

Nous n'avons pas utilisé d'autres librairies qu'il nous a paru intéressant de mentionner dans ce rapport. Nous avons bien sûr utilisé les classes IOException, SQLException ou encore Thread et Socket.

III- Procédure d'évaluation et de tests

A. Choix des différents tests

Au niveau des tests nous avons choisi de mettre en place des tests unitaires afin de s'assurer du fonctionnement de la base de données, du service TCP et du service UDP. Nous avons, pour cela, créé trois fichiers junit test case.

Pour le test de la base de données, nous avons repris les fonctions principales de la classe User pour s'assurer de leur fonctionnement. En effet, les fonctions qui utilisent ou écrivent dans la base de données sont situées dans la classe User.

Pour le test du service TCP, nous avons seulement lancé la fonction de réception des messages de notre classe TCP_serveur et la fonction d'envoi de messages de notre classe TCP_client.

Pour le test du service UDP, nous avons fonctionné de la même façon que pour TCP, en lançant la fonction de réception des messages de la classe UDP_serveur et la fonction d'envoi de messages de la classe UDP_client.

B. Application de ces tests et résultats

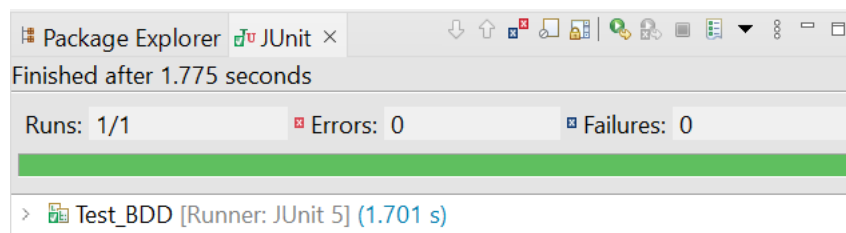


Figure 12 : Test de la BDD

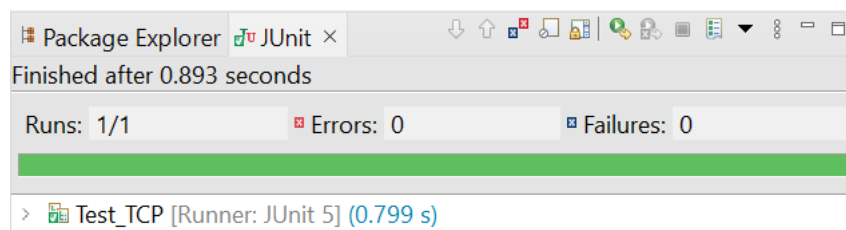


Figure 13 : Test TCP

IV- Procédure d'installation et de déploiement

A. Configurations préalables

Comme dans toute application et mise en place de système il y a des configurations à mettre en place préalablement pour garantir le bon fonctionnement du système.

De notre côté, cela consiste en deux étapes principales :

- Il faudra que, lors de l'installation du système l'adresse IP de broadcast du réseau de l'entreprise soit rentrée en dur.
- De plus, l'interface (eth0 par exemple) devra être rentrée en dur également, en fonction de l'interface utilisée pour relier les machines de l'entreprise entre elles.
- Pour finir, les port broadcast et TCP seront à rentrer à la main lors de la configuration.

Afin de faciliter la tâche à l'utilisateur, nous avons créé une classe VarGlobal qui comprend tous ces aspects de configuration et qui pourra être complétée par la personne qui mettra en place la messagerie au sein de l'entreprise.

B. Commandes de lancement

Pour télécharger notre application, l'utilisateur peut, en arrivant sur la page de notre dépôt Git, sur la droite dans l'onglet "release" télécharger la dernière version disponible en choisissant le bon .jar en fonction du système d'exploitation de l'ordinateur (Linux, Windows ou Mac).

Pour lancer l'application, il faut ouvrir un terminal, se placer où se trouve l'application et lancer la commande suivante :

```
java -jar App_Windows.jar  
java -jar App_Linux.jar  
java -jar App_Mac.jar
```

A choisir en fonction du système d'exploitation utilisé.

Notons qu'il est aussi possible de **lancer l'application en double cliquant** sur le fichier .jar.

L'application démarrera ensuite et l'utilisateur aura accès à toutes les fonctionnalités de messagerie.

C. Script

Nous n'avons pas trouvé nécessaire d'écrire un script bash permettant de lancer plus rapidement l'application, d'optimiser cette étape-là dans l'expérience utilisateur puisqu'en double-cliquant sur le fichier .jar, à télécharger, cette dernière se lance. Cela nous a paru assez intuitif et difficilement optimisable.

V- Manuel d'utilisation simplifié

A. Démarrage de l'application

Comme dit plus haut, pour télécharger notre application, l'utilisateur peut, en arrivant sur la page de notre dépôt Git, sur la droite dans l'onglet "release" télécharger la dernière version disponible en choisissant le bon .jar en fonction du système d'exploitation de l'ordinateur (Linux, Windows ou Mac).

Lorsque l'utilisateur souhaite lancer l'application, il se place dans le dossier où il a téléchargé l'exécutable, ouvre un terminal et exécute la commande suivante :

```
java -jar App_Windows.jar  
java -jar App_Linux.jar  
java -jar App_Mac.jar
```

Il pourra ensuite avoir accès à la fenêtre de connexion dont le fonctionnement est détaillé par la suite.

B. Connexion

1- Inscription

L'utilisateur lors de sa première connexion au service de messagerie renseigne un login et un mot de passe dont il devra se rappeler et renseigner à chaque connexion. De plus, il peut choisir un pseudo : c'est sous ce nom qu'il apparaîtra auprès des autres utilisateurs. Son login n'est pas communiqué. Il a la possibilité de le changer à chaque connexion, à condition que le pseudo choisi soit valide.

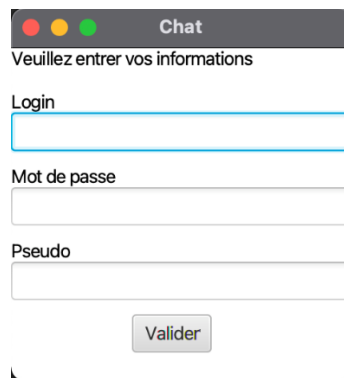


Figure 14 : Fenêtre de connexion

2- Connexion à un compte existant

L'utilisateur renseigne le pseudo et le mot de passe qu'il a donné lors de la dernière connexion et peut choisir un nouveau pseudo, qui peut très bien être le même que celui de la fois précédente.

A chaque utilisation de la messagerie, l'utilisateur a l'occasion de changer son pseudo à l'endroit prévu à cet effet (visible sur la capture d'écran de la fenêtre principale).

C. Chat

Une fois l'utilisateur connecté, une fenêtre principale s'ouvre :

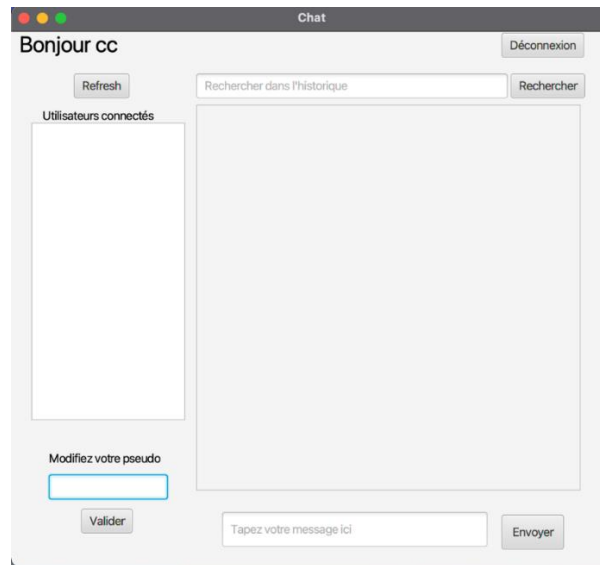


Figure 15 : Fenêtre principale

Il est alors possible pour l'utilisateur d'envoyer des messages et d'en recevoir en cliquant sur une personne de la liste connectée sur la gauche. Cette discussion apparaîtra dans la fenêtre de discussion.

Si un utilisateur se déconnecte, il disparaîtra en temps réel de la liste des utilisateurs actifs. S'il avait une discussion en cours avec un autre utilisateur, lorsque celui-ci tentera d'envoyer un message ce dernier ne sera pas envoyé. L'interlocuteur comprendra alors qu'il s'est déconnecté.

Il est également possible pour un utilisateur, une fois après avoir choisi son interlocuteur, de rechercher un mot dans l'historique de conversation pour retomber sur un message en particulier. Cette fonctionnalité est disponible au-dessus de la fenêtre de discussion.

D. Déconnexion

Lorsque l'utilisateur a fini d'utiliser le service de messagerie il peut cliquer sur le bouton de déconnexion de l'application se situant en haut à droite de la fenêtre principale et la fermer complètement.

Attention, en l'état, si on se déconnecte en cliquant sur le bouton, nous avons une exception qui se lève si on tente de se reconnecter. Il faut donc fermer l'application et la rouvrir.

Lorsqu'il le souhaitera il pourra toujours la relancer pour continuer une conversation, l'historique des anciens messages réapparaîtra lorsque l'utilisateur cliquera de nouveau sur son interlocuteur.

Table des illustrations

Figure 1. – Screenshot personnel, [23/01/2022]. *Organisation du GitHub.*

Figure 2. – Screenshot personnel, [23/01/2022]. *Organisation du Projet.*

Figure 3. – Screenshot personnel, [23/01/2022]. *Sprint 1.*

Figure 4. – Screenshot personnel, [23/01/2022]. *Sprint 2.*

Figure 5. – Screenshot personnel, [23/01/2022]. *Sprint 3.*

Figure 6. – Screenshot personnel, [23/01/2022]. *Sprint 4.*

Figure 7. – Screenshot personnel, [23/01/2022]. *Jenkins.*

Figure 8. – Screenshot personnel, [23/01/2022]. *Diagramme des Use Case.*

Figure 9. – Screenshot personnel, [23/01/2022]. *Diagramme de Classe.*

Figure 10. – Screenshot personnel, [23/01/2022]. *Diagramme de Séquence général.*

Figure 11. – Screenshot personnel, [23/01/2022]. *Diagramme de Séquence de connexionl.*

Figure 12. – Screenshot personnel, [23/01/2022]. *Test de la BDD.*

Figure 13. – Screenshot personnel, [23/01/2022]. *Test de TCP.*

Figure 14. – Screenshot personnel, [23/01/2022]. *Fenêtre de connexion.*

Figure 15. – Screenshot personnel, [23/01/2022]. *Fenêtre principale.*

Table des annexes

Annexe 1 : Lien GitHub	A
Annexe 2 : Lien Jira.....	A

Annexe 1 : Lien GitHub

<https://github.com/chamaben/new-Projet-COO-POO>

Annexe 2 : Lien Jira

<https://projectpoo.atlassian.net/jira/software/projects/POO/boards/1>

INSA Toulouse

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



MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE