INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport COO/POO 2022
Service de Messagerie Insatact

Quentin DOUARRE, Pierre FAVARY
28/01/2022

Index

Présentation du produit:

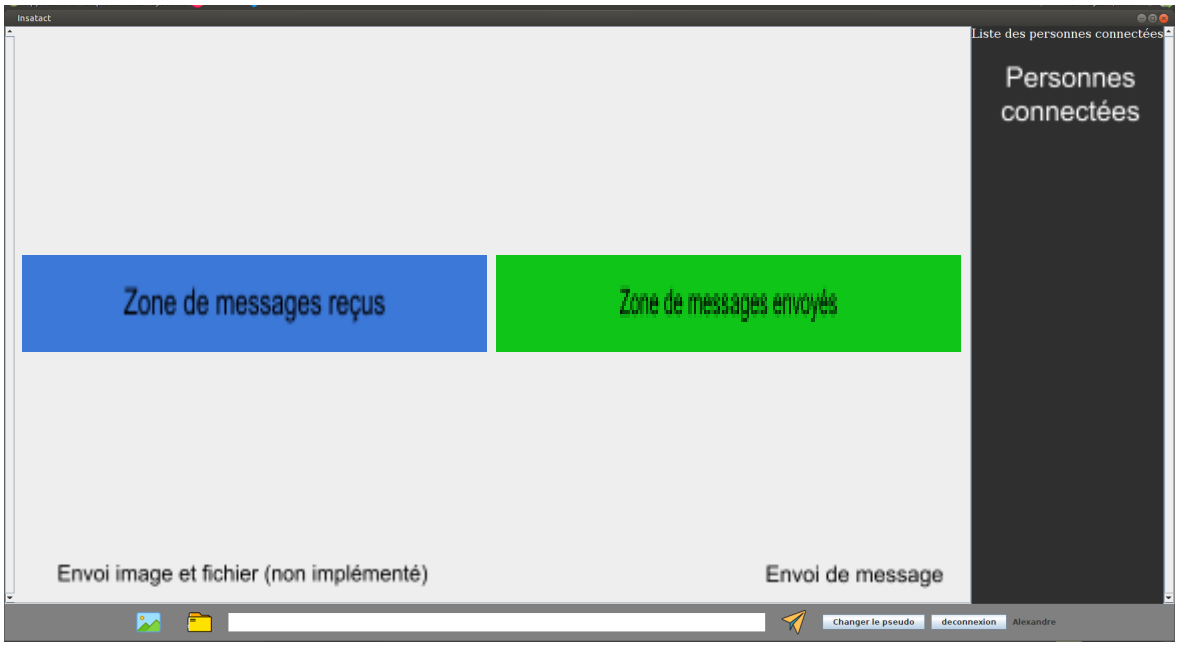
Ce projet, intitulé Insatact, a pour but la communication sur un même réseau interne de manière décentralisée; une base de données commune est néanmoins requise pour l'authentification ainsi que la récupération des messages échangés lors d'une ancienne session.

La messagerie permet l'envoi et la réception de messages jusqu'à 4095 caractères fiabilisés par l'emploi du protocole TCP, ainsi que leur affichage, le choix d'un identifiant unique et d'un mot de passe personnel. L'utilisateur peut échanger avec n'importe quel autre utilisateur connecté sur le même réseau et récupérer un historique des anciens messages à sa connexion.

L'utilisateur choisit également un pseudonyme personnel unique et modifiable qui est l'identité qui sera présentée aux autres.

La présence de chaque utilisateur est signalée régulièrement via un broadcast UDP (message envoyé à tous les utilisateurs du réseau sans garantie de fiabilité) contenant identifiant, pseudonyme et adresse IP.

Aucune connaissance en informatique n'est requise pour l'utilisation de ce programme.



Interface graphique principale de notre service de messagerie

Description de l’architecture de la messagerie:

Remarque: Nos diagrammes UML contiennent en plus de la partie implémentée en Java la gestion des média et des fichiers réfléchie en conception.

Tous nos diagrammes UML sont disponibles sur notre git. Nous avons laissé notre premier jet. Tous les diagrammes finaux commencent par new.

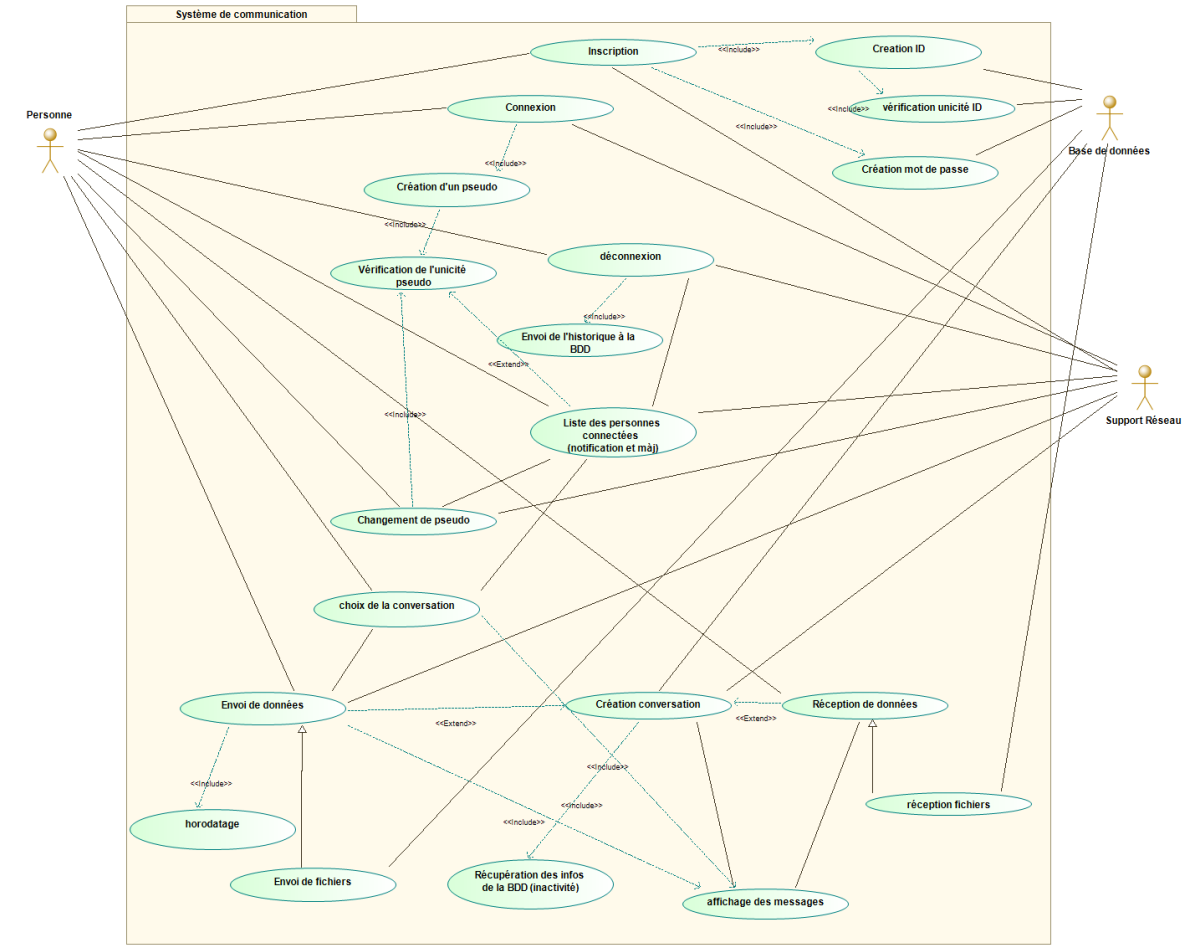


Diagramme UseCase de Insatact

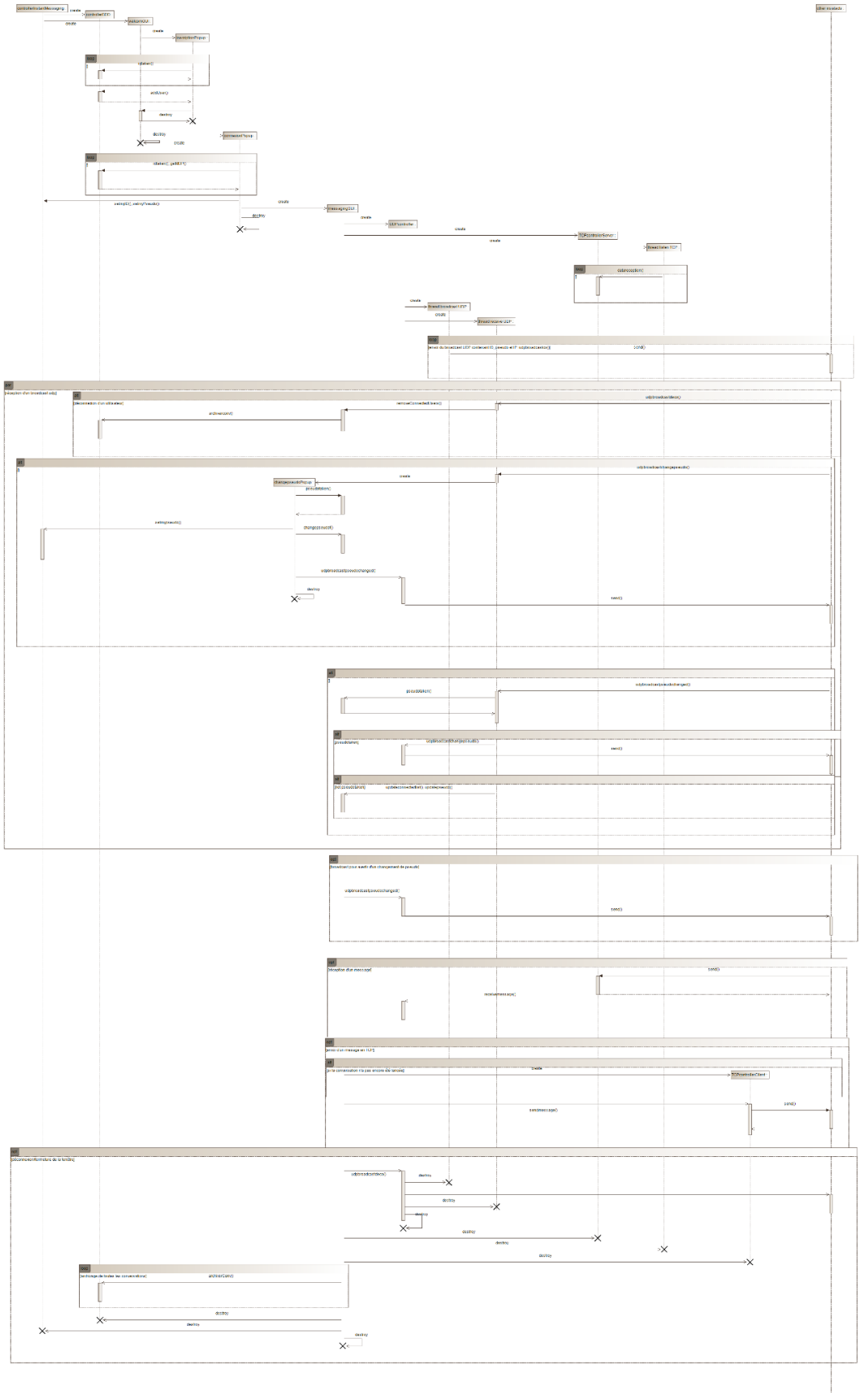


Diagramme de séquence boîte blanche de Insatact

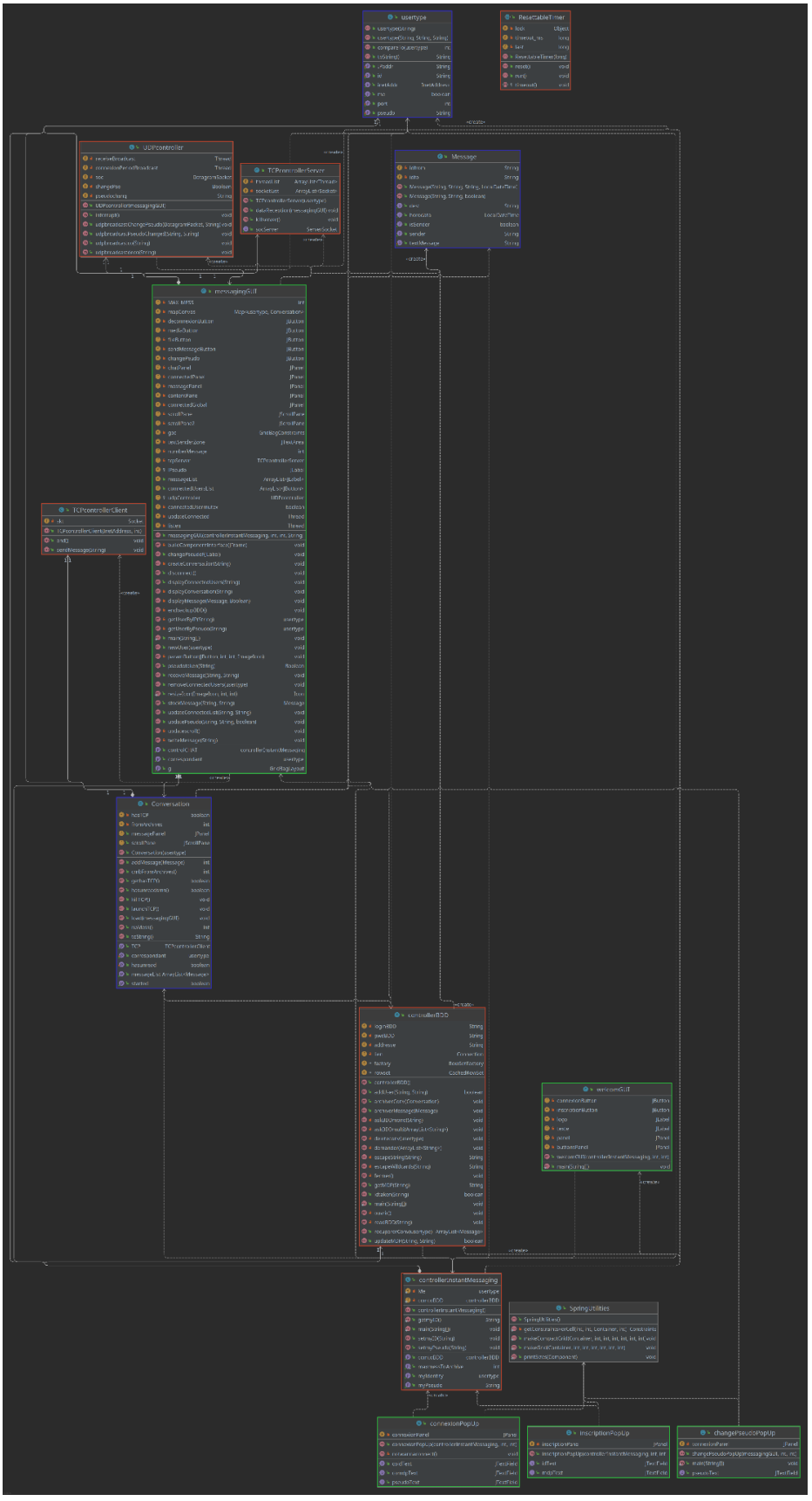


Diagramme de classe de Insatact

Il y a trois couleurs sur le diagramme de classe. Le rouge symbolise le découpage de notre messagerie en contrôleurs, le vert pour l'interface graphique, le bleu pour les conversations, messages, users.

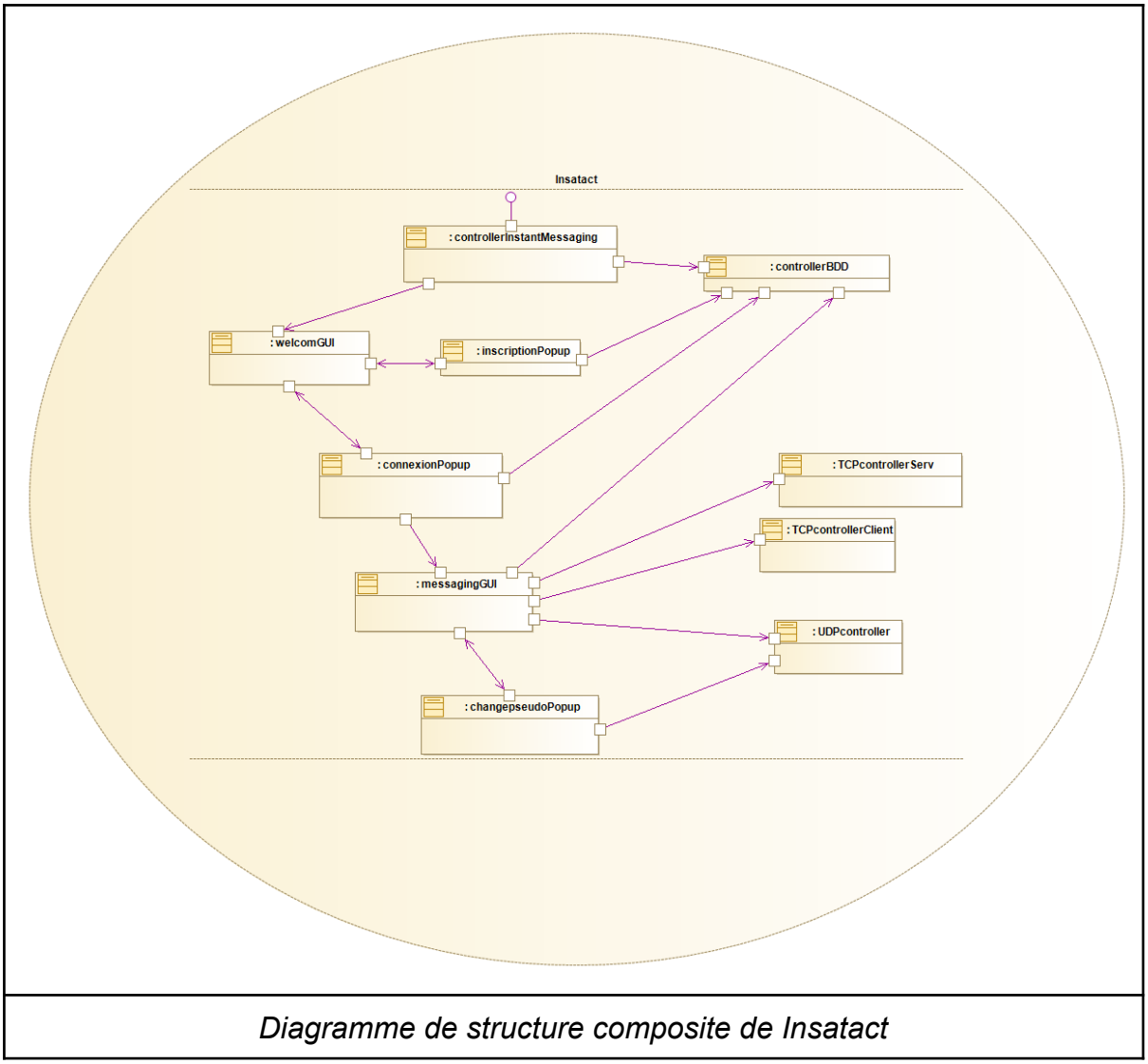


Diagramme de structure composite de Insatact

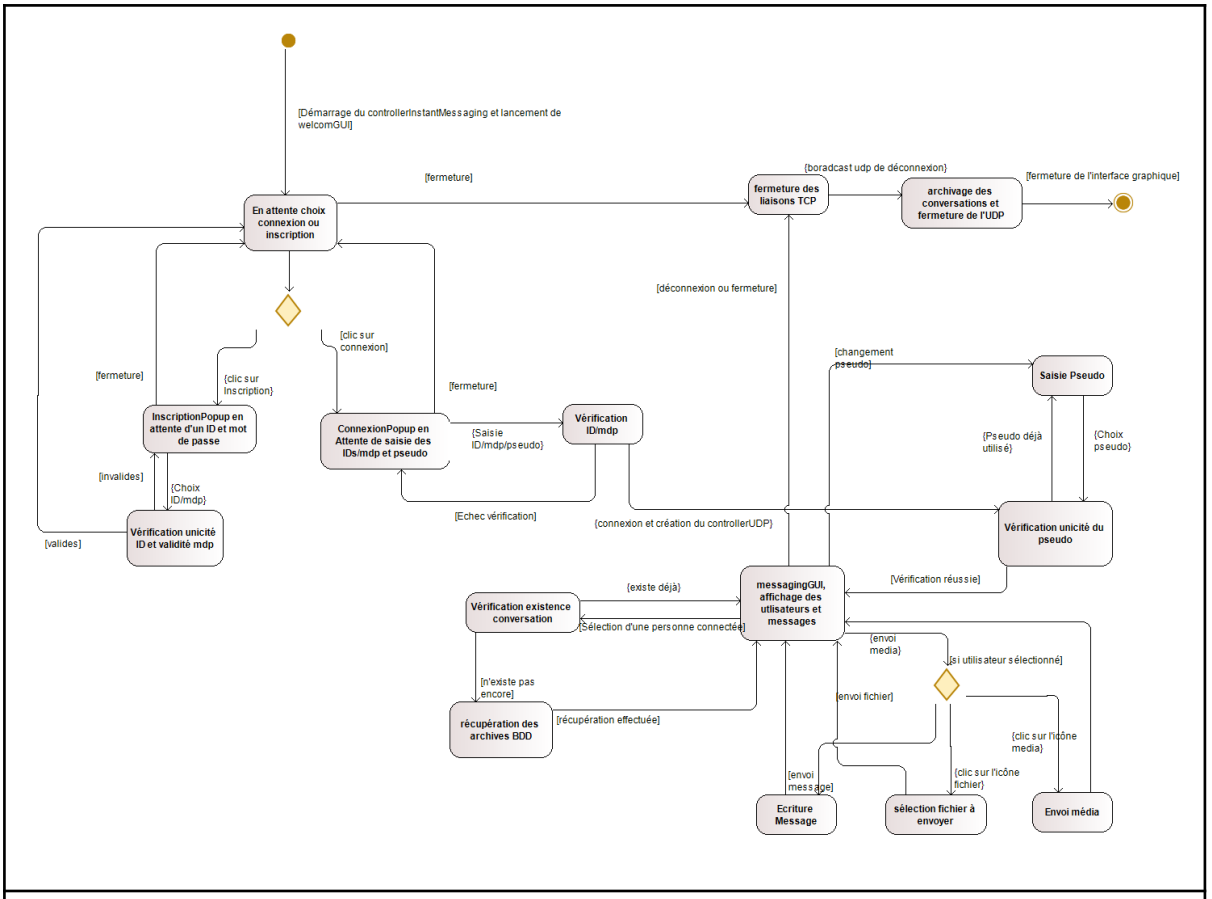


Diagramme d'états de Insatact

Nous avons séparé notre messagerie en cinq contrôleurs. Le controllerInstantMessaging est l'entité chargée du lancement de la messagerie, de son interface graphique et de la gestion de l'identité de l'utilisateur . Le controllerBDD nous permet d'interagir facilement avec la base de données centralisée. Il contient l'ensemble des fonctions nécessaires au bon fonctionnement de la messagerie, à son interaction avec la base de données ainsi qu'à la maintenance de cette dernière.

Nous avons décidé de scinder le controllerNetwork (contrôleur réseau) en trois parties: une pour le protocole UDP, deux pour TCP: client et serveur.

Pour controllerUDP, nous avons créé deux threads. Le premier est chargé de la réception du trafic UDP et doit enclencher une série d'actions en fonction du trafic reçu. Le deuxième envoie périodiquement un paquet pour signaler aux utilisateurs périodiquement que la personne est connectée. Dans la classe controllerUDP, il y a également des fonctions d'envoi d'UDP broadcast pour permettre l'envoi d'une information "système" à toutes les autres interfaces UDP connectées.

Côté TCP, nous avons une classe cliente et une classe serveur. Chacune, respectivement, possède les fonctions et attributs nécessaires pour l'établissement d'une connexion, la fermeture, l'envoie côté client et réception côté serveur. De plus, une création de threads dans la classe serveur a été ajoutée afin de permettre au serveur de recevoir des messages de plusieurs utilisateurs en même temps.

La partie interface graphique a été découpée en 5 parties. Tout d'abord, nous avons la page d'accueil qui va inviter l'utilisateur à s'inscrire et à se connecter. Pour chacune des actions, une pop-up est prévue. L'inscription permet l'enregistrement des IDs de l'utilisateur dans la BDD. La connexion permet l'entrée dans la messagerie en vérifiant identifiant et mot de passe dans la BDD. L'interface de la messagerie contient les fonctions permettant l'affichage de la liste des personnes connectées, l'affichage des messages écrits, envoyés, reçus.

Un objet central au fonctionnement de la messagerie est néanmoins difficilement visible sur les diagrammes UML: mapConvos.
Voici comment fonctionne grossièrement la messagerie: les messages sont de type "Message" contenant le texte du message bien sûr, mais aussi les identifiants expéditeur et destinataire. Ces messages sont stockés en liste dans un objet "Conversation", qui contient également un client TCP et qui est associé à un unique correspondant. Nous devions stocker une liste d'utilisateurs connectés (avec leur propre type "usertype") et une liste d'objets Conversation pour éviter des manipulations trop lourdes; nous avons donc décidé d'utiliser une hashmap pour les associer. Une hashmap permet d'associer des objets d'un type A à des objets d'un type B, en considérant que les objets A -dans notre cas les utilisateurs- sont les clés pour accéder aux objets B -dans notre cas les conversations.
Cela a grandement facilité nos méthodes, comme la recherche d'un utilisateur spécifique dans la liste des connectés ou l'obtention d'une conversation via l'identifiant du correspondant.

Choix des technologies utilisées:

Nous avons choisi d'implémenter ce service en Java car ce dernier est un langage orienté objets donc très pratique pour fragmenter notre projet en entités afin d'implémenter au fur et à mesure et à la fin assembler des objets entre eux. La machine virtuelle associée au langage facilite considérablement la portabilité du code. De plus, le Java étant très utilisé dans l'industrie, une équipe qui devrait rajouter du contenu sur le service pourrait le faire d'autant plus facilement.

Pour gérer la partie graphique de notre service, nous avons utilisé SWING, une technologie java intéressante pour découvrir et se familiariser avec les interfaces graphiques. Néanmoins elle n'est plus très utilisée du côté de l'industrie.

L'IDE choisi est Visual Studio Code car ce dernier est le plus populaire dans le monde du développement et nous semblait plus ergonomique. Il nous paraît donc intéressant d'apprendre à l'utiliser d'autant plus que dans le cadre académique, nous avons travaillé avec Eclipse.

Afin de pouvoir développer à plusieurs, nous avons opté pour la solution git appuyée par la plateforme github. Ce dernier nous a permis de partager nos changements rapidement via un dépôt. Nous avons travaillé avec une unique branche car nous nous synchronisons de sorte que chacun modifie des fichiers que l'autre ne touche pas (cf répartition du travail).

Dans le cadre de ce projet, nous avons eu besoin d'une base de données centralisée. Pour cela, nous avons utilisé un serveur de notre école en utilisant la technologie MySQL. Cette technologie a l'avantage d'être assez légère par rapport aux concurrentes.

Afin d'automatiser le projet et ainsi faciliter le déploiement du projet, nous avons utilisé Apache Maven. Ce dernier nous permet de générer un .jar très facilement exécutable.

Enfin, de sorte d'avoir un suivi régulier de l'avancement du projet, nous avons utilisé la méthode AGILE à l'aide de l'outil JIRA. Ce dernier est très apprécié et utilisé dans l'industrie. Il était donc judicieux de nous familiariser avec son fonctionnement.

Répartition du travail:

Quentin : Interfaces graphiques, UDP, TCP, JIRA, Maven
Pierre : BDD, Gestion classes Conversation, Message, Usertype, WelcomGUI

Ci-dessus est présentée notre répartition du travail initial. Vers la fin du projet, nous avons tous les deux touché à l'ensemble des parties afin de débugger, de peaufiner l'application et de parfaire son assemblage.

Nous avons découpé notre travail en 6 sprints. Le premier sprint de Jira consistait à avoir les interfaces graphiques opérationnelles pour notre service de messagerie. Le deuxième sprint était consacré à la connexion des différentes interfaces graphiques entre elles, au début de la gestion de la BDD, d'UDP. Ce sprint a eu du retard car nous n'arrivions pas à trouver et à mettre les bonnes librairies pour la BDD. Durant le troisième sprint, nous avons fini la mise en place de la BDD et le broadcast UDP. Le quatrième sprint a permis le développement de la partie TCP, client et serveur et la gestion des conversations entre les différents utilisateurs. Le cinquième sprint a été consacré à la gestion de l'archivage des conversations dans la BDD, de l'automatisation du projet avec Maven, de débogage. Nous avons également fait un scénario test pour le client. Le dernier sprint nous permet d'aider le

client à la prise en main du projet en proposant un ReadMe, des commentaires, des diagrammes UML.

Installation de l’application:

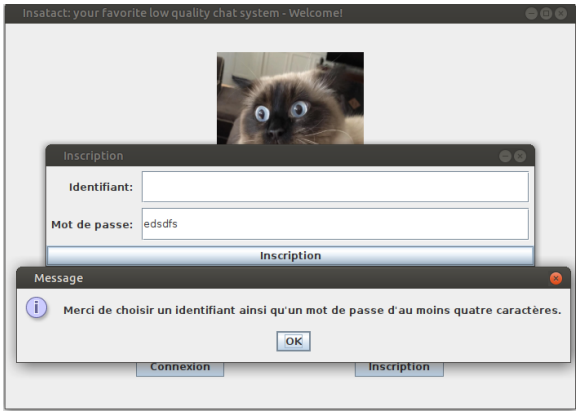
1. Allez sur: https://github.com/Quintus618/Projet-Java-2022
2. Faites un git clone (commande “git clone https://github.com/Quintus618/Projet-Java-2022.git” dans un dossier)
3. Installez Maven
4. Rendez-vous dans /Projet-Java-2022/Implementation_Messagerie/insatact
5. Tapez : mvn clean compile
6. Tapez : mvn package
7. Tapez: java -jar [fichier .jar]

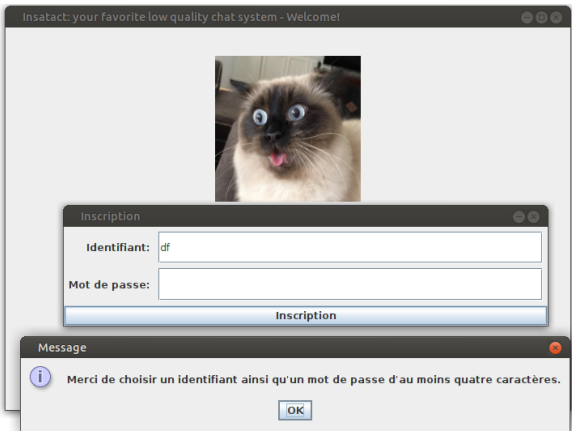
Scénario de tests:

ATTENTION: Lors du lancement de la messagerie, si vous ne voyez pas le chat en image, relancez.

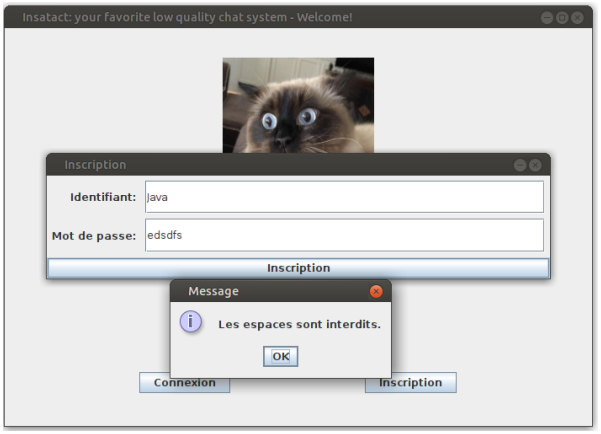
Inscription d'un nouvel utilisateur:

1. Oubliez de remplir un des champs

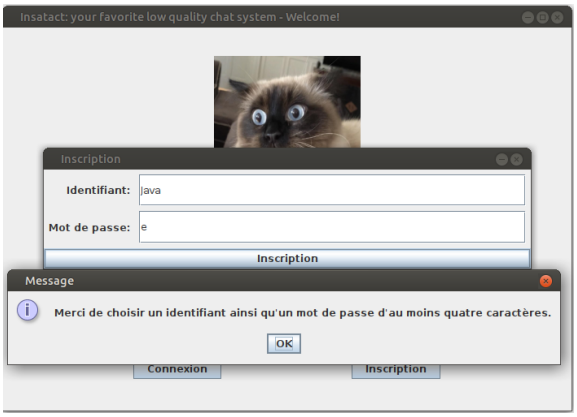




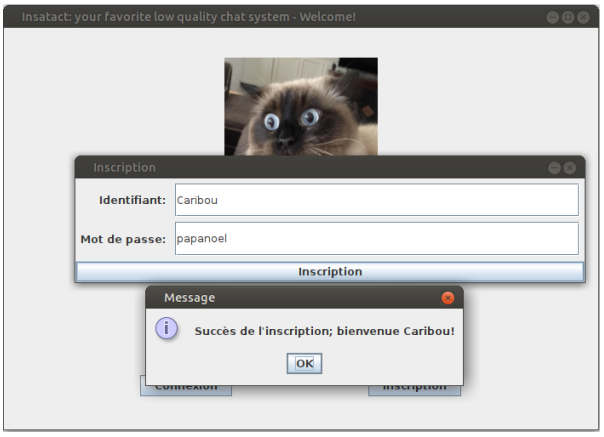
2. Saisissez votre ID avec un mot de passe



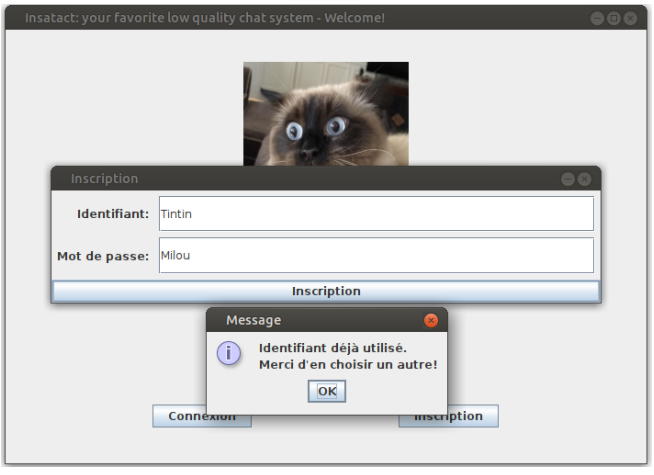
3. Saisissez un mot de passe avec moins de 4 caractères



4. Inscrivez-vous en évitant les erreurs ci-dessus



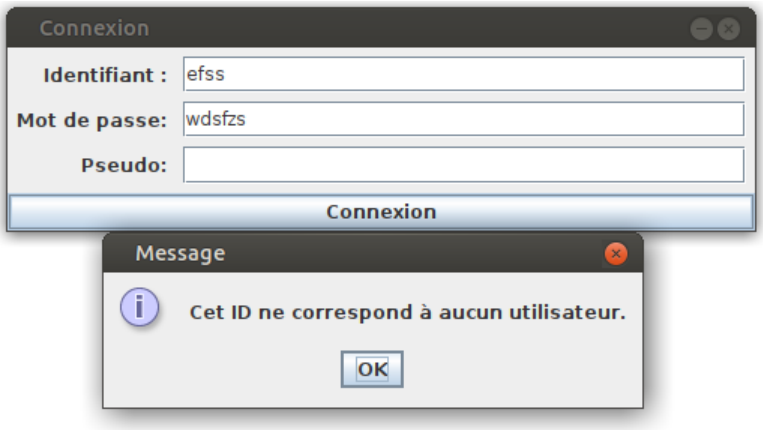
5. ATTENTION, vous ne pourrez pas vous inscrire avec des identifiants déjà utilisés



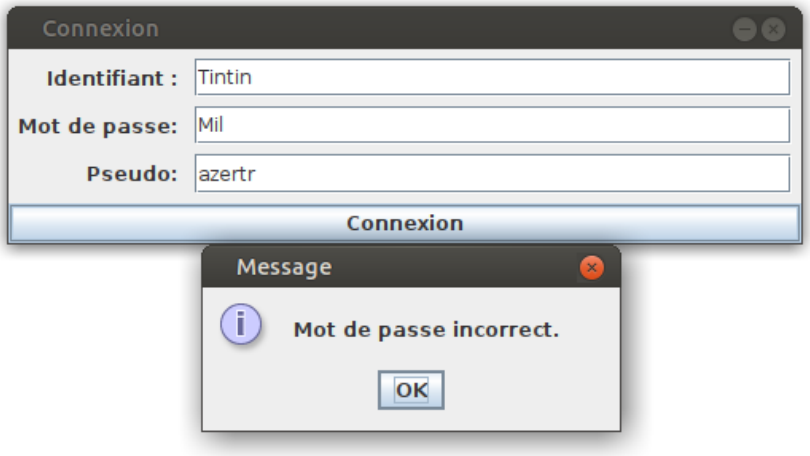
6. Faites une inscription pour trois machines.

Connexion à la messagerie:

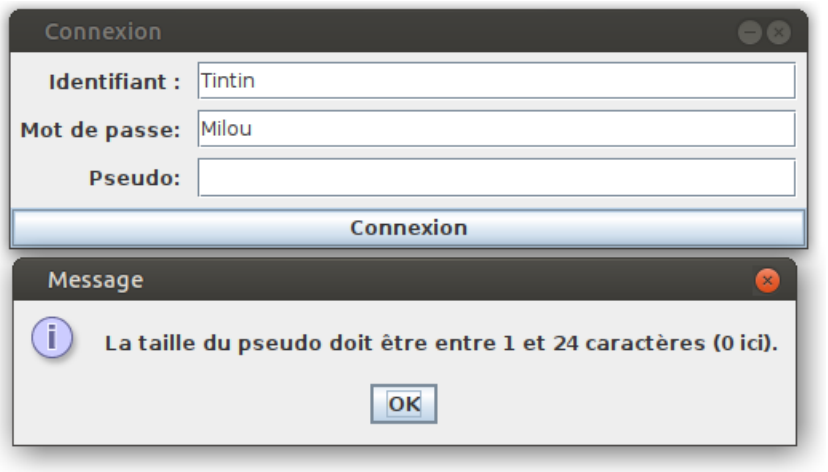
1. Saisir un mauvaise ID



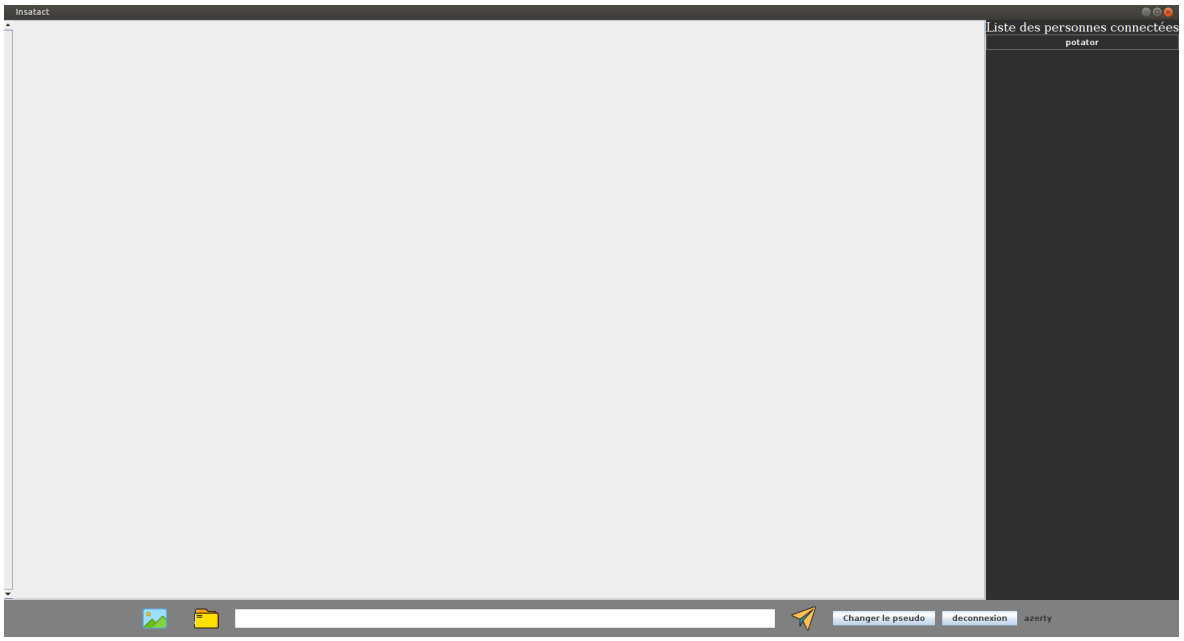
2. Saisir un mauvais mot de passe



3. Saisir les bons identifiants mais pas de pseudo



4. Faire tout comme il faut sur deux machines

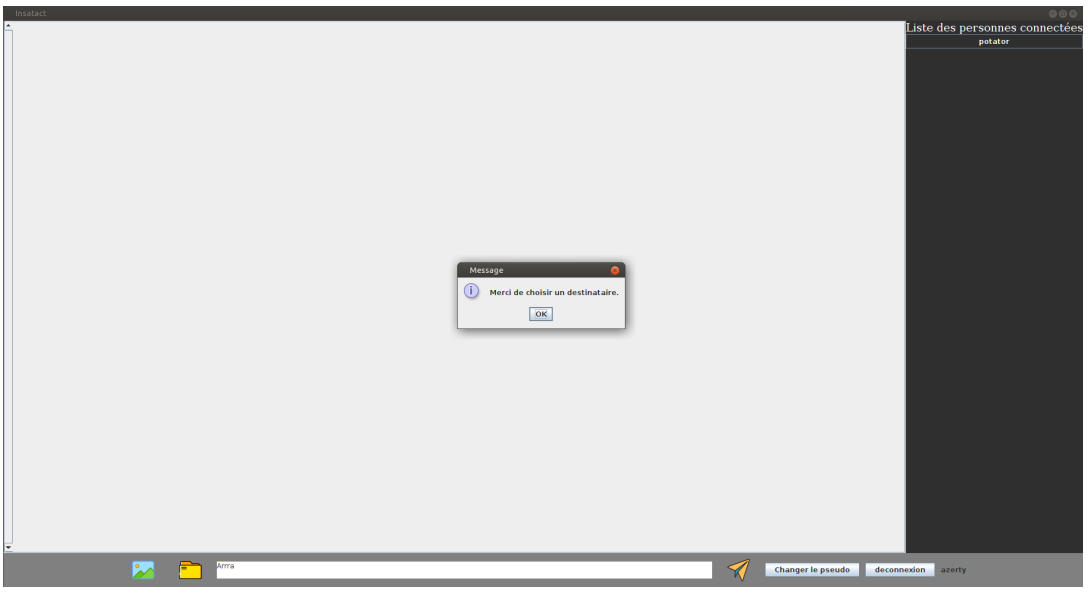


Vous voilà au sein de la messagerie. Sur la photo, nous voyons dans la liste des connectés un autre utilisateur, potator.

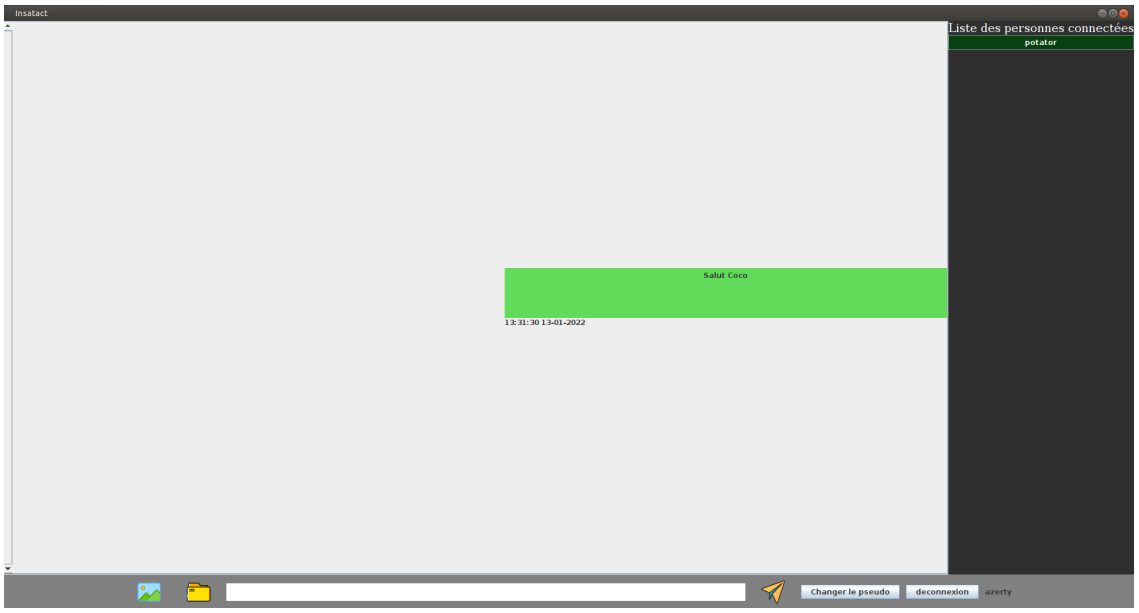
5. Faites une connexion avec deux des utilisateurs.

Envoi de messages:

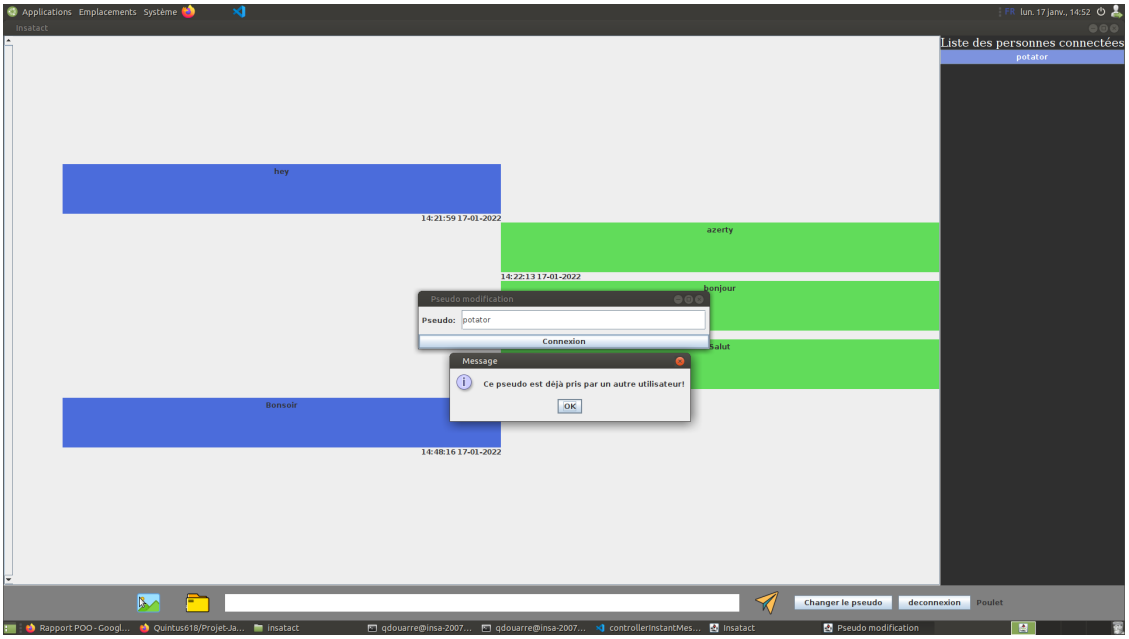
1. Essayez de cliquer sur l'avion directement



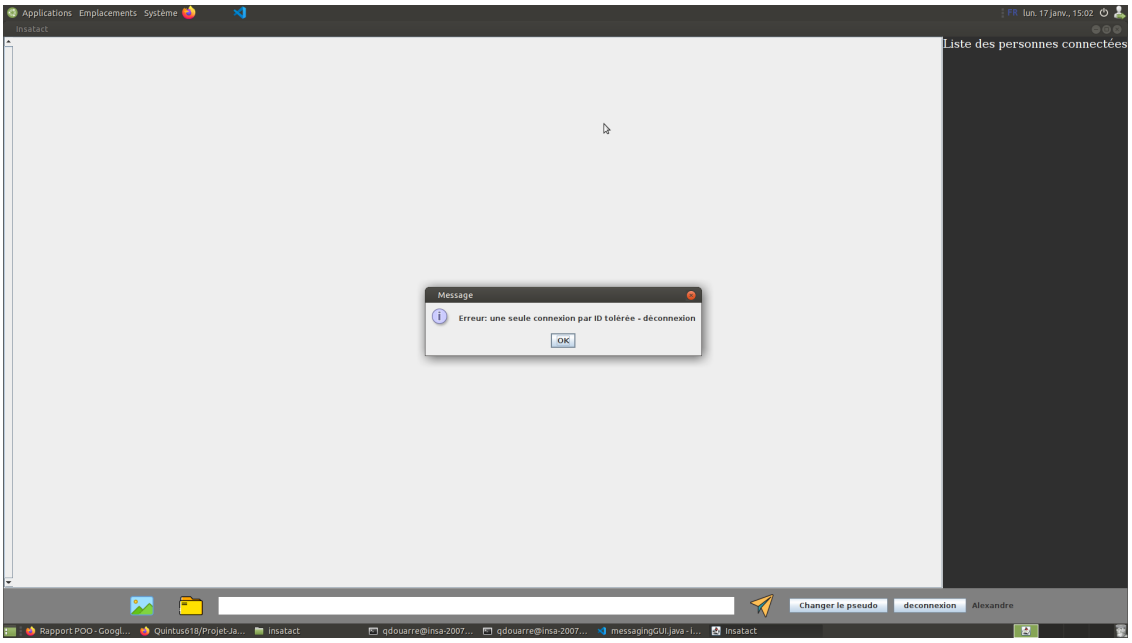
2. Maintenant sélectionnez votre destinataire dans la liste des connectés et appuyez sur l'avion



3. Maintenant appuyez sur Changer le pseudo et sélectionner potator: un message indiquant que le pseudo est déjà pris va apparaître



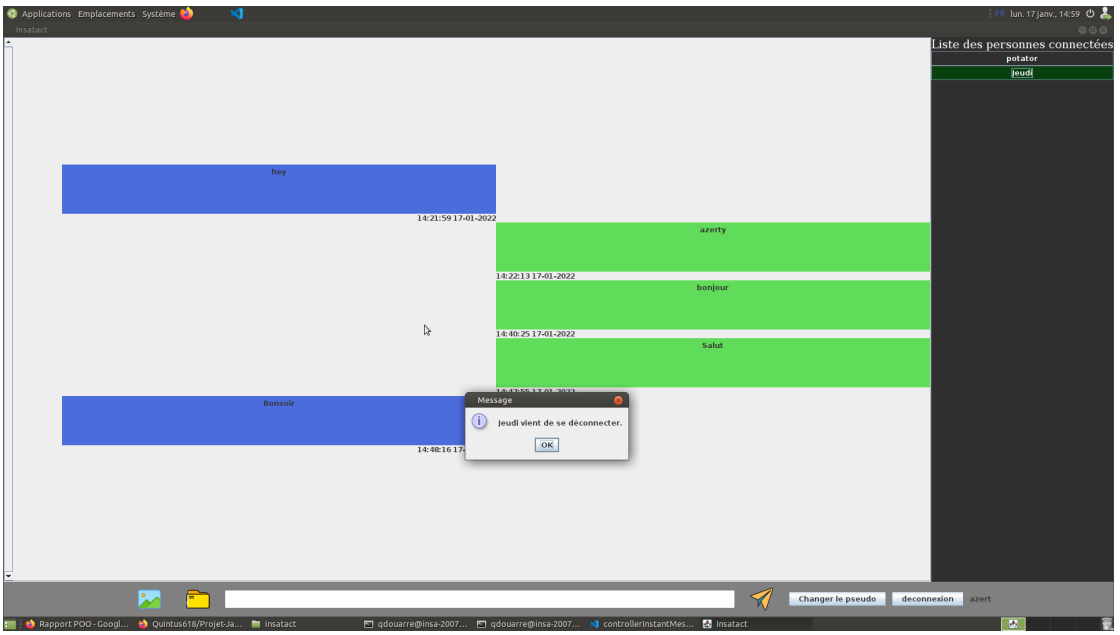
4. Changez votre pseudo avec un non utilisé
5. Maintenant connectez la troisième machine avec le même pseudo, une invitation à changer de mot de passe est envoyée aux deux machines
6. La troisième machine va se reconnecter avec les mêmes IDs, les deux machines concernées vont être expulsées



Remarque: Nous avons utilisé un jeu de couleurs pour la liste des connectés: le vert indique la conversation active, le bleu des messages non lus et enfin le lavande des conversations établie.

Déconnexion :

Appuyez sur le bouton déconnexion



Remarques: La méthode appelée à la fermeture de la fenêtre est identique à celle appelée en appuyant sur le bouton de déconnexion.

Pistes de développement futur:

Notre service de messagerie, pour l'instant, fonctionne sur un réseau local. Elle est optimisée pour un réseau à faible perte comme les réseaux filaires de type ethernet. Nous prévoyons la mise en place de timers, notamment lors de la déconnexion pour les réseaux locaux sans fil du type 802.11. Certaines méthodes sont déjà implémentées avec un booléen "fromTimer" en paramètre donnant un comportement spécifique.

Nous prévoyons de faire des fonctions afin d'améliorer la sécurité de l'application comme par exemple avoir la possibilité de changer de mot de passe (déjà écrite dans controllerBDD), chiffrer le trafic…Si possible gérer les autorisations d'écriture et de lecture à la base de données.

Nous prévoyons également d'ajouter de nouvelles fonctionnalités comme l'envoi d'images de différents formats ainsi que le partage de fichier en utilisant la BDD (cf Conception Orientée Objets, la table est déjà prête dans la BDD).

Nous envisageons d'ouvrir notre service à des utilisateurs externes. Pour cela, nous avons réfléchi à deux fonctions pour traiter des informations de la liste des connectés. La première récupère des informations de la liste locale: elle va par exemple checker cette liste pour savoir si un pseudo n'est pas déjà pris. Cette approche ne suffit pas pour un réseau avec un fort taux de pertes. C'est pour cela qu'une seconde approche est à envisager reposant sur un multicast UDP, pour le changement de mot de passe. Ainsi des utilisateurs extérieurs pourront recevoir les informations des personnes connectées.

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

Université Fédérale Toulouse Midi-Pyrénées
Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE