MANUSCRIPT CHAT SYSTEM

Pour:
DA COSTA MAMEDE CORRÊA Vitor - *dacostam@insa-toulouse.fr*
LOPEZ ROMERO Luis Alberto - *lopezrom@insa-toulouse.fr,*
ORGANISTA CALDERÓN José Daniel - *organist@insa-toulouse.fr,*

Enseignants:
Arthur Bit-Monnot - *abit@insa-toulouse.fr*
Sami Yangui - *yangui@insa-toulouse.fr*
Nawal Guermouche - *guermouc@insa-toulouse.fr*

Vendredi 28 Janvier

Institut national des Sciences appliquées de Toulouse
GEI
INFORMATIQUE ET RÉSEAUX
2022

INSA TOULOUSE

SOMMIER

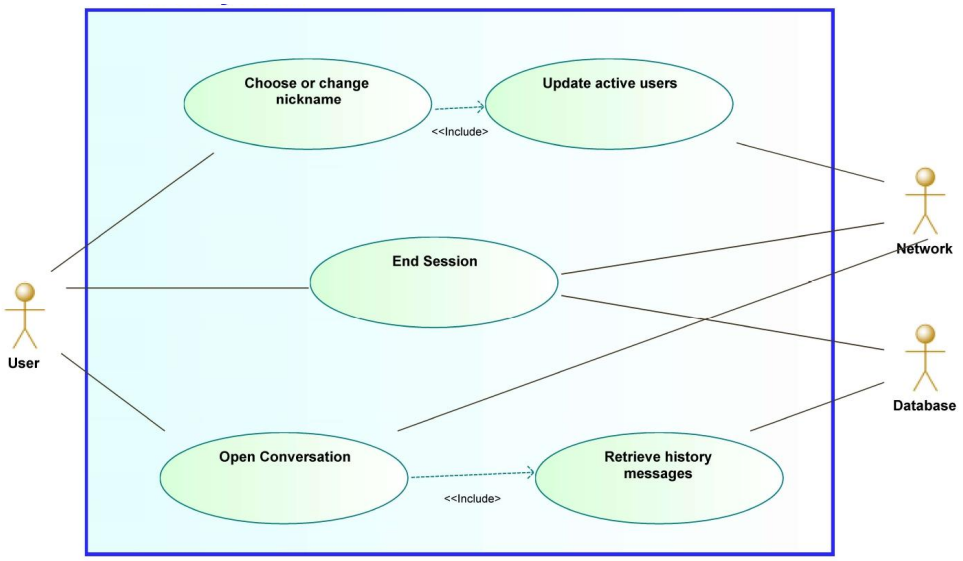
1. Introduction Générale

Dans le cadre du premier semestre du 4ème année de la spécialisation IR à l'INSA Toulouse, il nous est proposé de faire la conception, modélisation et programmation du projet *SYSTÈME DE CLAVARDAGE DISTRIBUE INTERACTIF MULTI-UTILISATEUR TEMPS RÉEL*, comme prérequis partiel d'approbation au sein des cours de Conception Orientée Objet, Programmation Orientée Objet, Processus De Développement Logiciel Automatisé, entre autres.

Ce projet nous a permis de mettre en œuvre plusieurs concepts et compétences acquis aux cours; Ce rapport présente ce projet dans sa globalité avec précisions sur certains points que nous jugeons plus importants pour l'évaluation de notre acquisition de ces connaissances.

2. Conception du projet

En utilisant les informations fournies par le client (dans ce contexte adapté, les requirements fonctionnelles et non-fonctionnelles contenu dans le cahier de charges des exigences utilisateur). Nous avons fait la conception du projet, en commençant par la création des diagrammes UML, le diagramme de cas de use qui contient l' abstraction des principales fonctionnalités du système est montré ici-dessous, les autres peut-être vérifié dans le repo git du projet dans le dossier "DiagramsImages".



3. Architecture du système et choix technologiques

Dans cette section nous présentons quelques points sur des principales techniques, classes et choix technologiques faites pendant l'exécution du projet.

Maven
Pour la gestion de dépendances et gestion du projet en général nous avons utilisé la technologie maven, ce qui avec le fichier de configuration pom.xml nous a énormément simplifié et aidé avec le processus de compilation et déploiement.

Broadcast UDP
Nous avons utilisé le mécanisme de broadcast UDP pour la découverte de utilisateurs et pour l'envoi d'actualisation d'états, comme le changement de pseudo, ou la connexion/déconnexion d'un utilisateur.

Messaging TCP
Pour la communication entre seulement deux utilisateurs, notamment pour l'envoi des messages du chats, le protocole utilisé est le TCP, en raison de sa fiabilité.

Base de Données Centralisé
Pour sauvegarder et récupérer l'historique de messages nous avons choisi d'utiliser une base de données centralisé, plus précisément un serveur MySQL hébergé aux serveurs de l'INSA, de cette façon tous les clients sont connecté au serveur SQL et sauvegarder tous les messages qu'ils ont reçu, pour retrouver l'historique de messages il faut juste consulter la base.

Interface avec la base de données
Pour faire l' interface avec la base de données, nous avons créé une classe qui s'appelle "HistoryService" afin de gérer tous ces interactions, de ce façon nous pouvons contrôler l'accès à la base et aussi s'assurer de la fiabilité et intégrité des interactions/queries faites, en évitant des possibles sql injections et autres actions malicieux.

Model MVC
La organizational des fichiers du projet a été fait d'accord avec le standard MVC, d'une façon que nous pouvons facilement identifier et séparer la partie graphique/logique, de la partie backend et de contrôle, ainsi que les structures de données.

Réutilisation de code
Nous pouvons aussi relever la réutilisation de code pour le moyen de la fonctionnalité de l'héritage dans la langage de programmation java, pour ce projet

nous avons centralisé et créé une classe de base pour tous les “views” et après on pourrait facilement hériter de ce classe pour créer nouveaux views.

Réception des messages
Pour “écouter” la reseaux et des messages des autres utilisateurs nous avons créé des “listener” services, qui exécutent dans deux autres threads créés par le message service, une pour l’UDP et autre pour le TCP. Ces services sont toujours en attente de nouveaux messages, qui sont tous traités ensuite de son arrivée.

Actualisation d'utilisateurs actifs
Pour ne pas rater la possible connexion/déconnexion d’un utilisateur au cas où son message de changement d'état n’est pas reçu par les autres utilisateurs, ce qui peut arriver en raison de la méthode UDP en mode broadcast utilisé pour la diffusion de ce information, nous avons créé un service responsable pour renvoyer le statut actuel du client chaque N seconds.

Utilitaires Réseaux
Pour faciliter l' interaction avec des éléments du réseau au niveau des autres classes, nous avons créé un classe singleton qui contient des fonctions qui font l’abstraction de plusieurs de ces fonctionnalités au niveau réseaux, comme par exemple l’obtention de l’IP et le MAC de la machine utilisateur et l’envoie des messages UDP et TCP dans la reseaux locale.

Classe Central du Systeme
Nous avons centralisé la création de threads et la gestion des ressources à une seule classe principale afin de simplifier ce processus complexe, pour ça, la création et la gestion de plusieurs fonctionnalités et routines sont centralisées à la classe MessageService.

Structure de Données et sa transmission
Pour la structure de données utilisées, les classes dans le directoire "models" peuvent être consultées. Pour la transmission et réception de ses données, nous avons utilisé des fonctionnalités de la classe java “serializable” qui font la “conversion” de tout la classe, et plus importants, de ses propriétés, en bytes, qui sont encodé dans le format base64 et envoyé à des autres utilisateurs. Après la réception de la message, il faudrait juste faire l'inverse pour obtenir l’objet original et toutes ses données.

4. Procédure d'installation et de déploiement

Avec le JAR

Pre-requis
- Java Version 11 ou plus

Instructions
- Télécharger le fichier .JAR du repo git du projet.
- Il faut seulement exécuter le fichier .JAR bien configuré.
- Si la reseaux d'exécution du système est différent de l'IP 192.168.0.0/24 il faut recompiler le source-code comme indiqué ci-dessous.

Compiler le source-code

Pre-requis
- Java Version 11 ou plus
- Maven

Instructions
- Télécharger ou faire une git clone du repo git du projet, et accéder au directoire "projet".
- Il y a une makefile contenant plusieurs commandes utiles pour la utilisation du programme, c'est possible de l'exécuter, par exemple, avec le command "make" ou sinon un "make install" pour générer le fichier .JAR.
- Il faut aussi changer l'IP de la reseaux qui va être utilisée pour l'envoi et réception des messages s'elle est different de la reseaux locale (192.168.0.0/24) par default du systeme, ce qui peut être fait en changeant le valeur de la variable "BASE_IP" du fichier *src/main/java/utils*.

5. Manuel d'utilisation du produit

En lançant l'application l'écran de login vous sera présenté. (Fig 1). Il faut remplir le champ affiché avec un pseudo de votre choix, comme montré à la figure 1. Ensuite, en appuyant sur le bouton "LOGIN" ou sinon en appuyant sur la touche "Entrée" c'est possible de se connecter au système.

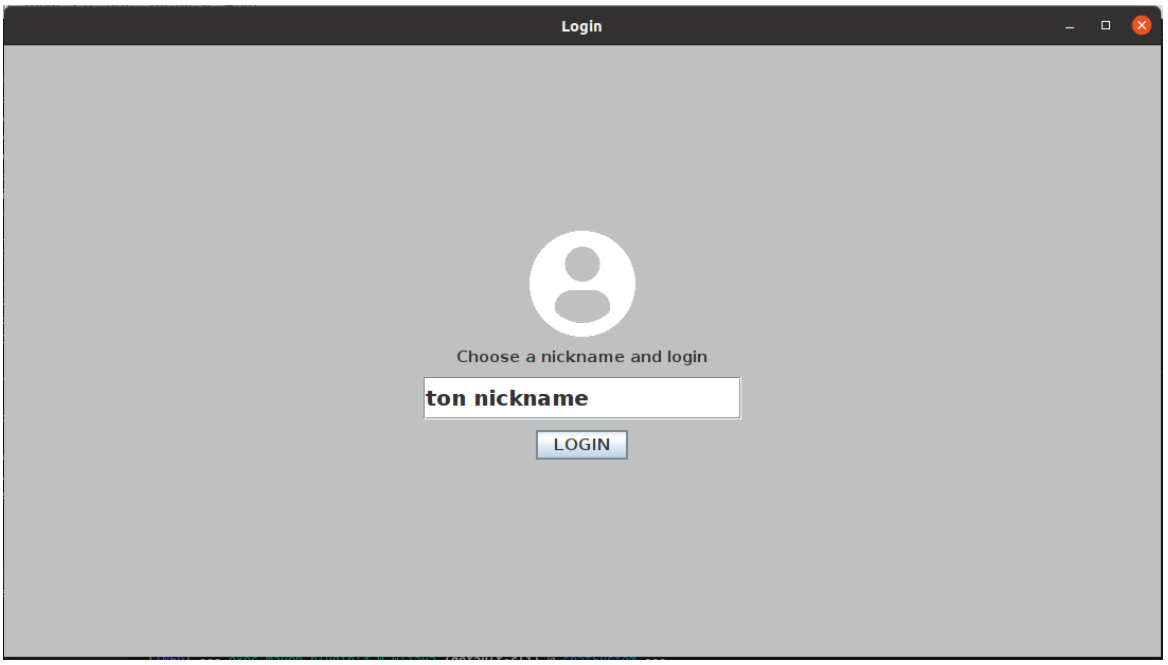


Fig 1. Écran du login

En écrivant un prénom déjà choisi pour un autre utilisateur, un message d'avertissement s'affiche (Fig 2). Il faut cliquer sur le bouton “OK” et chosir un autre pseudo.

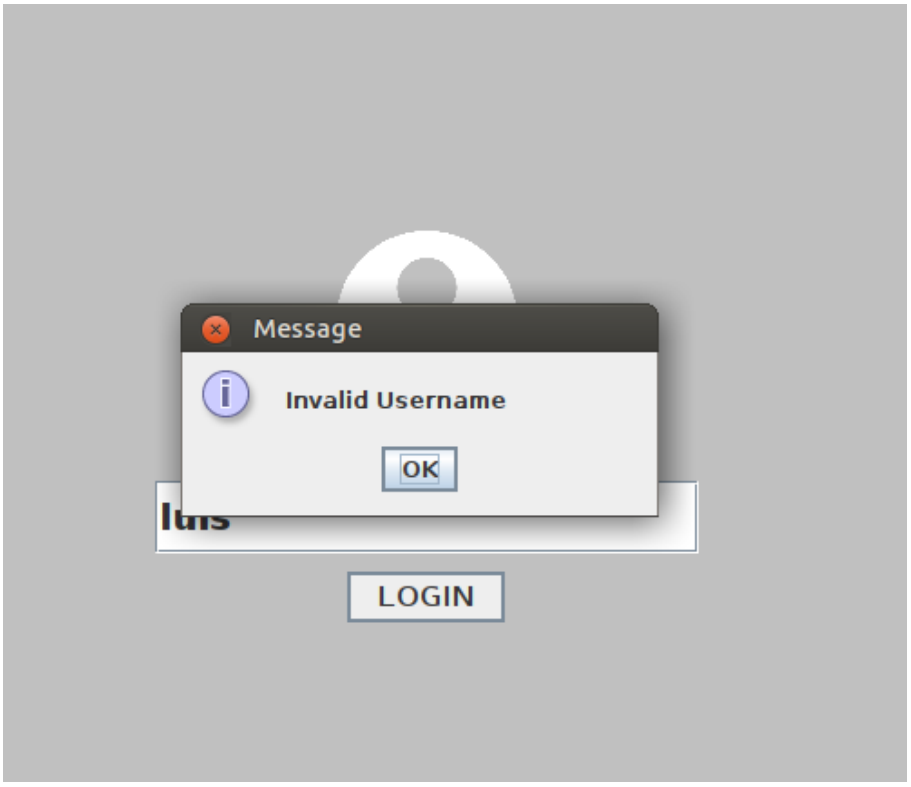


Fig 2. Message de surnom invalide

Ensuite de faire le login, l'écran de chat vous sera présenté. Dans ce écran, 3 panneaux sont présent (Fig 3):

- À gauche, le panneau des utilisateurs connectés. En cliquant sur un utilisateur, le panneau de chat actif ouvrira la conversation avec cet utilisateur.
- À droite le panneau de chat actif: au-dessus de ce panneau se trouve un champ de texte pour envoyer des messages à l'utilisateur sélectionné.

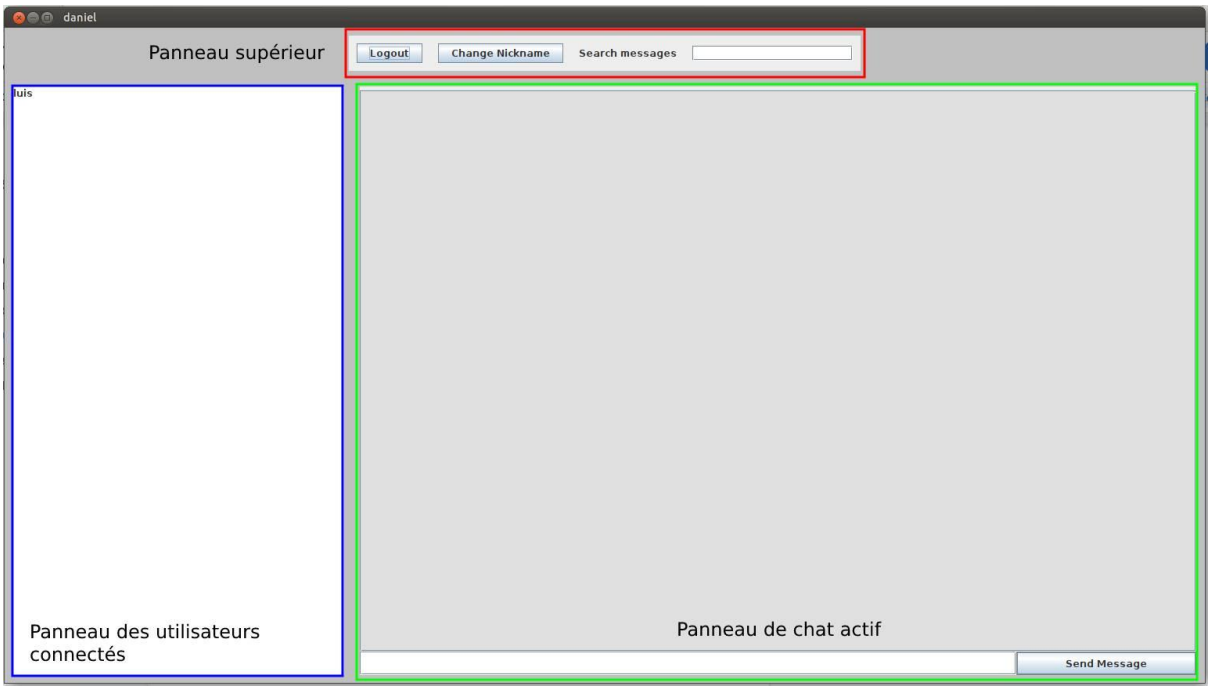


Fig 3. Écran de conversation.

- En haut, se trouvent deux boutons et un champ de texte. Le premier bouton à gauche "Logout" sert à se déconnecter, une fois déconnecté l'écran du login vous sera présenté à nouveau (Fig 1). Le deuxième bouton, "Change nickname" sert à modifier le pseudo, il faut que remplir le nouveau pseudo et ensuite cliquez sur "OK" (Fig 4), si le pseudo a été déjà pris par une autre utilisateur. Un message vous sera présenté indiquant que le prénom n'a pas été modifié (Fig 5).
  Le champ de texte à droite sert à chercher des messages contenant des lettres/mots spécifiques d'une conversation. En écrivant une lettre/mot et appuyant sur la touche Entrée, c'est possible de filtrer des messages contenant ce sequence de lettres (Fig 6).

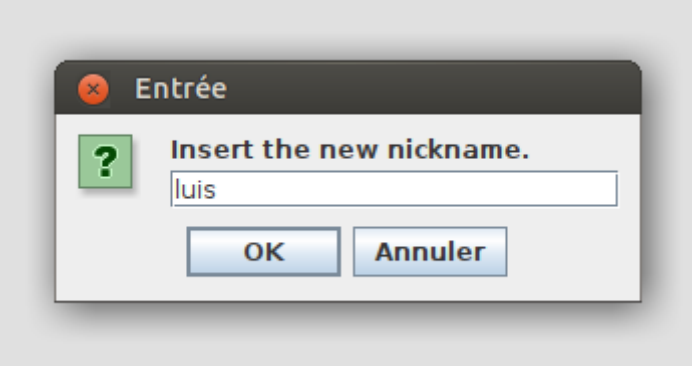


Fig 4. Message pour modifier le prénom.

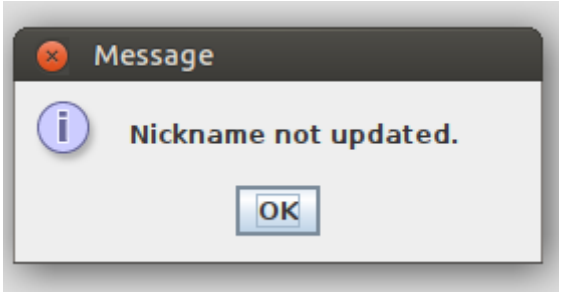


Fig 5. Message pour notifier prénom non modifié.

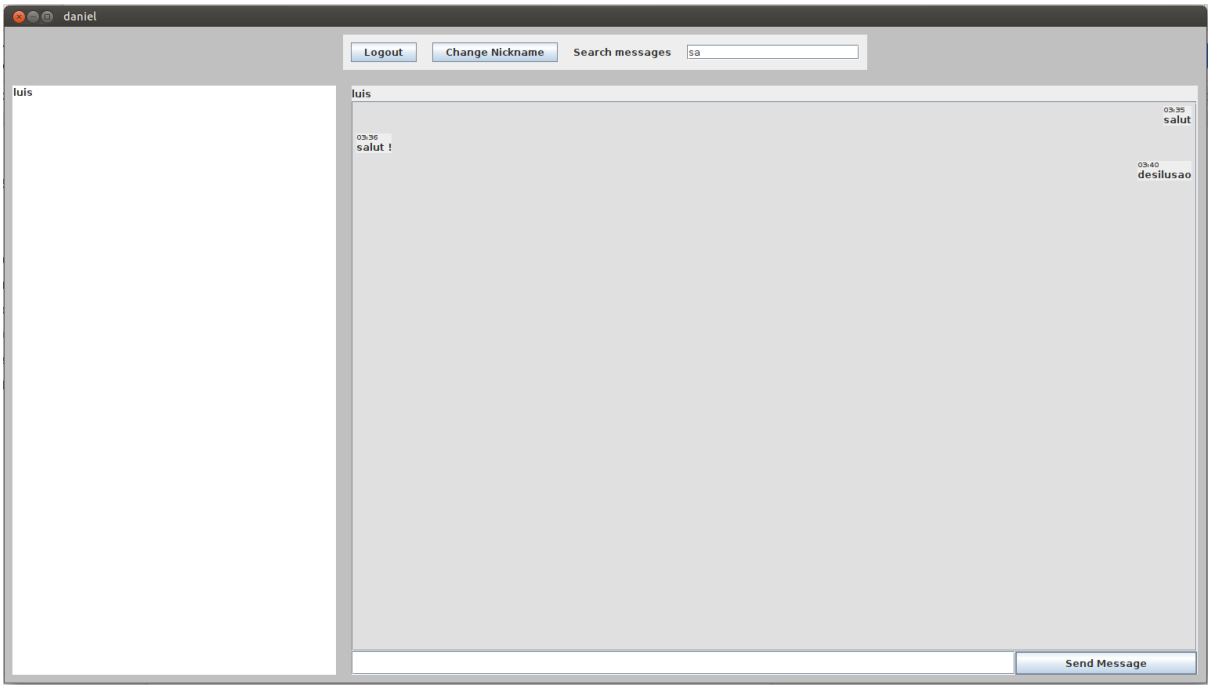


Fig 6. Exemple de filtrage de messages.

6. Procédures de tests et de validation

Pour tester le système, nous avons utilisé le concept de mock, qui consiste à créer des utilisateurs simulés pour vérifier des fonctionnalités déjà existant sur le système sans nécessairement le besoin d'une autre client connecté, et de cette façon, c'était possible aussi de tester la majorité des fonctions indépendamment de la couche réseau du système, qui a été testé séparément.

Pour la procédure de création de nouvelles fonctionnalités sur github, nous avons utilisé la méthodologie de git workflow, dans ce contexte spécifique le plus important qu'il faut relever c'est l'utilisation branches de développement pour chaque nouvelle fonctionnalité spécifique qui vont être développées. Nous avons créé des branchés dédiées à chaque grosse ensemble relationnel de fonctionnalités, qui ensuite seront sujet d'une "pull request", qui seront vérifiés par les autres membres du groupe et alors mergés à la branche principal, après possibles corrections, discussions et approbation du code.

Pour les procédures de validation, nous avons utilisé l'outil jenkins avec GitHub pour faciliter l'intégration des changements dans le code source.

En jenkins nous avons créé un tâche pour automatiser la création d'un jar avec la dernière version de le projet (la dernière version de la branche main). De cette manière, nous avons configuré une machine local ou nous avons installé jenkins et maven pour créer une tâche (Fig 7). Ensuite nous avons le configurée:

1. Pour l'intégrer a le projet GitHub nous avons indiqué le lien du git (Fig 8),
2. Pour spécifier un temp pour faire le "build" la construction automatisée de le jar nous avons utilisée la scrutation de de l'outil de gestion de version(Fig 9),
3. Pour faire le jar nous avons utilisée le comand clean install (Fig 10)

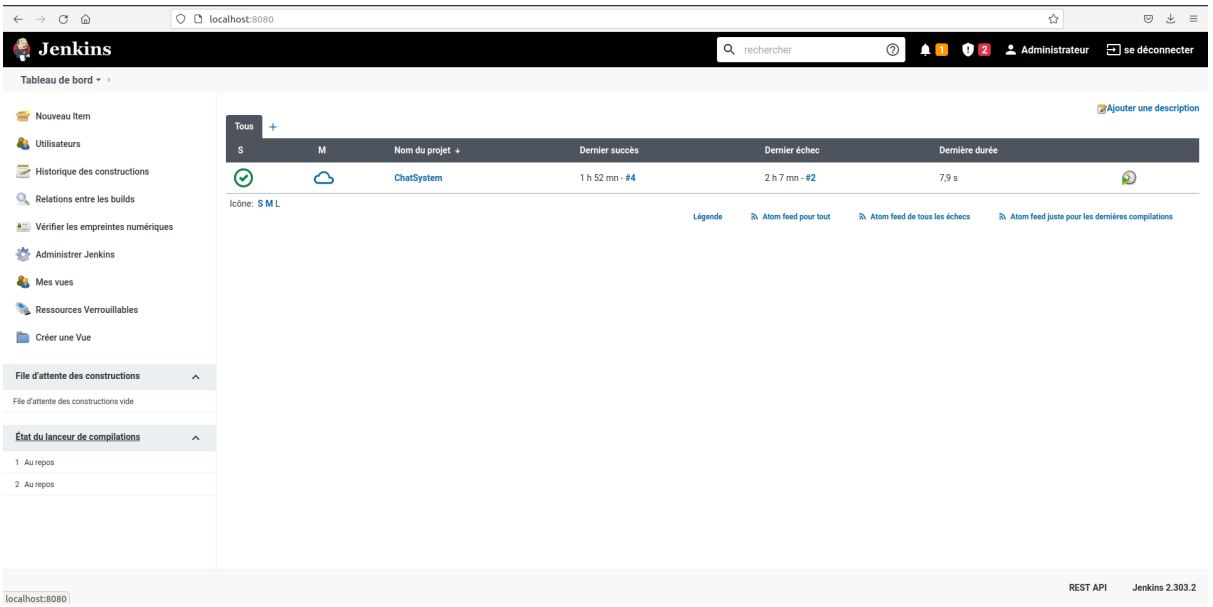


Fig 7. Tâche du projet sûr Jenkins.

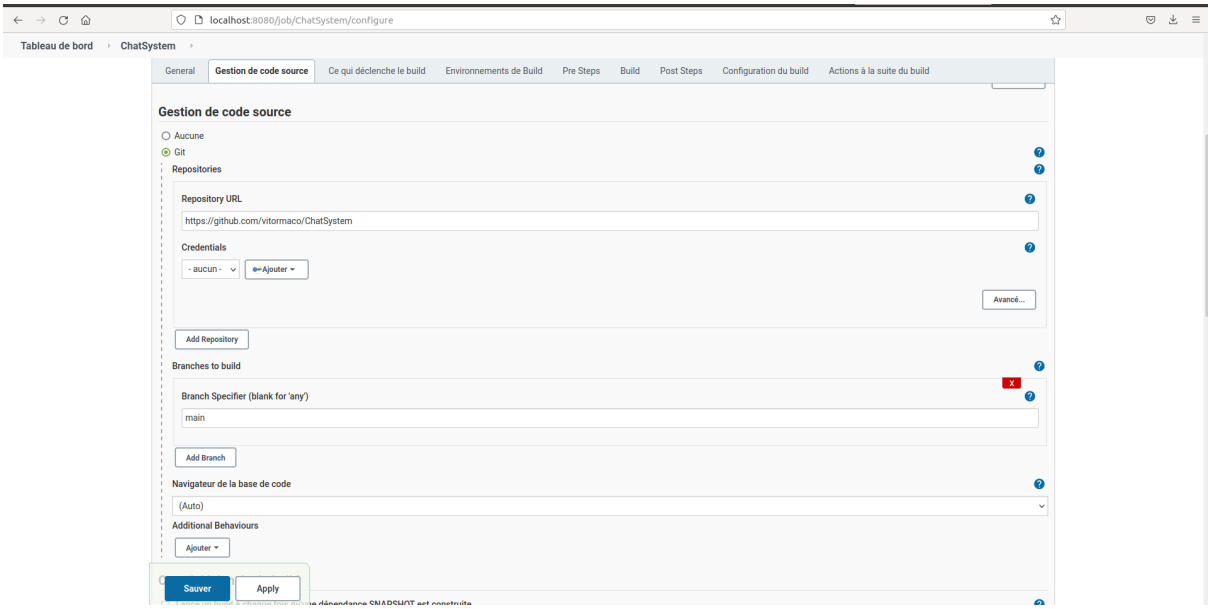


Fig 8. Configuration du tâche concernant Github.

Fig 9. Configuration du tâche concernant le “Planning”.

Fig 10. Configuration du tâche concernant le “Build”.

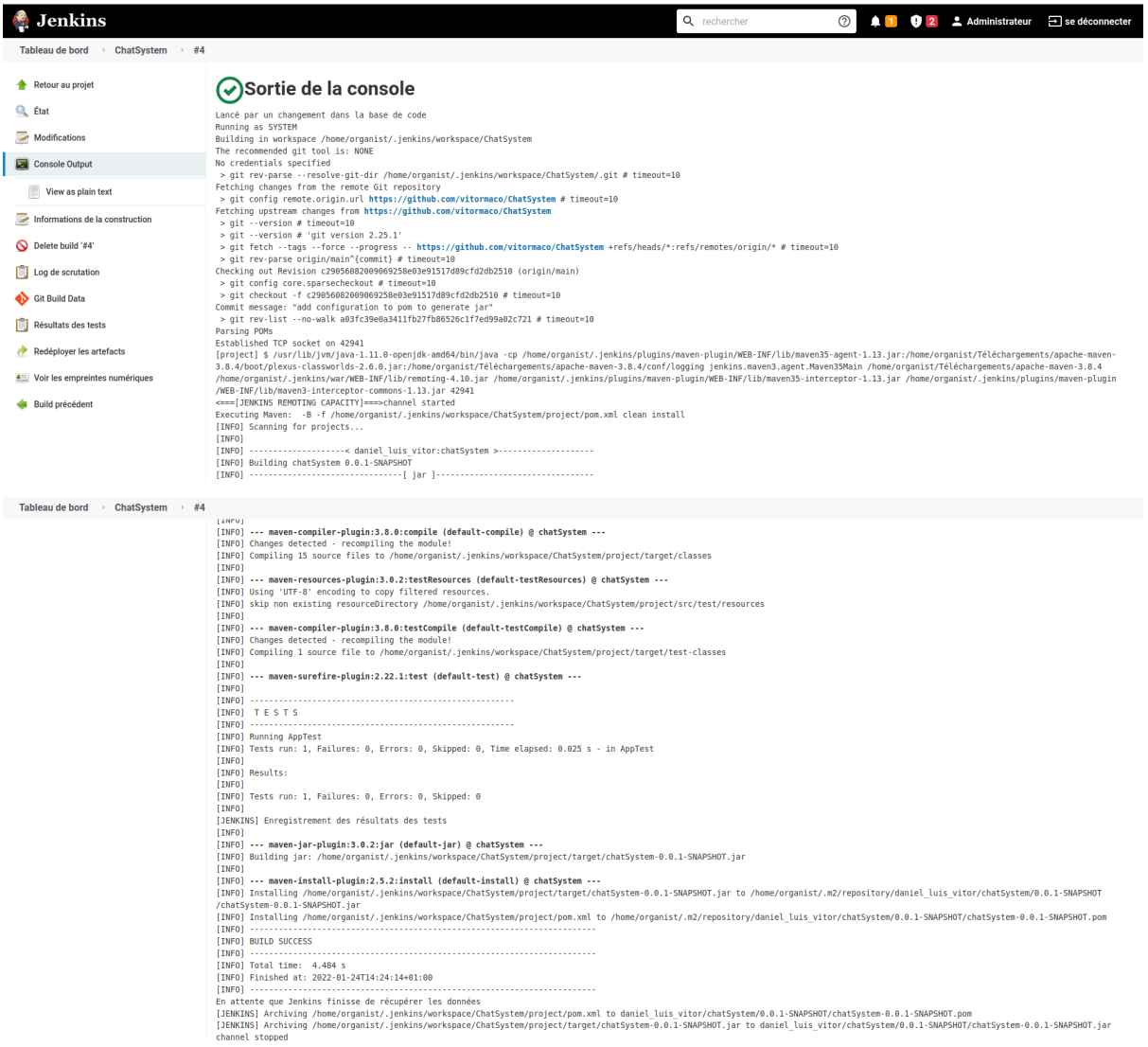


Fig 11. Sortie de la console une fois que la tâche se lance.

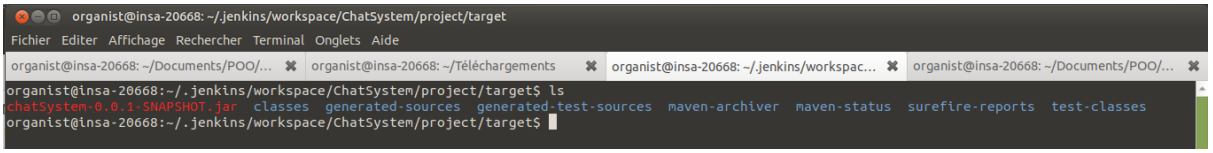


Fig 12. Jar crée localement pour Jenkins.

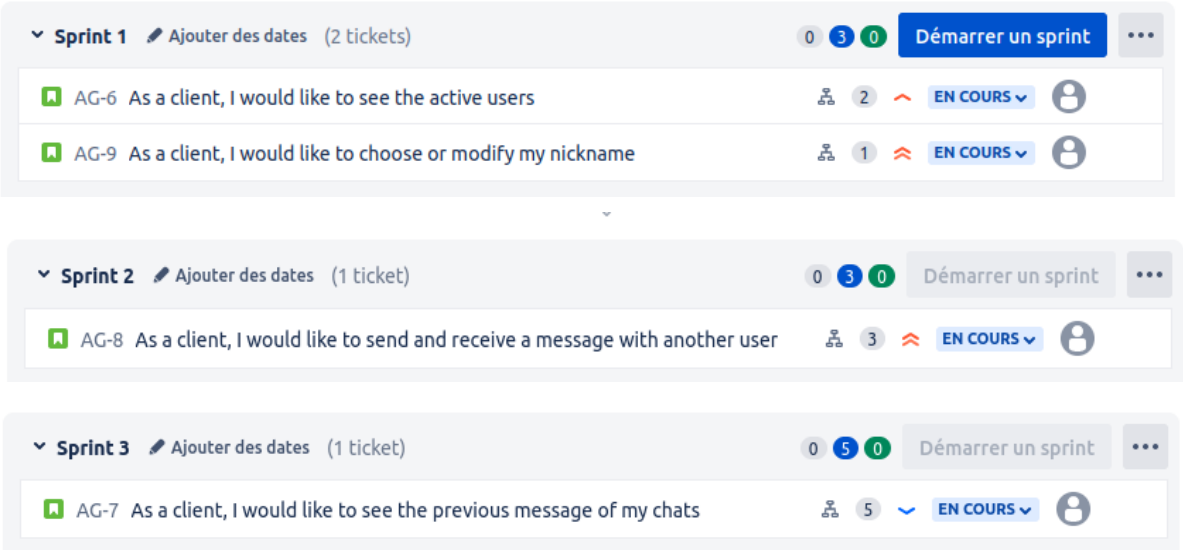
7. Méthodologie de gestion du projet

Pour une bonne gestion du projet, nous avons poursuivi la méthode Agile SCRUM, en utilisant le tool Jira de cette méthode.

Dans la méthodologie SCRUM nous proposons les suivants 4 users stories, chaque storie avec sa difficulté et priorité dans le projet :

User Story	Priority	Complex
As a client, I would like to see the active users	High	Easy
As a client, I would like to choose or modify my nickname	Higher	Most easiest
As a client, I would like to send and receive a message with another user	Higher	Difficult
As a client, I would like to see the previous message of my chats	Low	Most difficult

Ensuite, nous avons distribué ces 4 Users Stories en 3 sprints de 10 jours chacun :



Dans laquelle chaque user story a ces propres tâches techniques à remplir dans laquelle nous comme équipe on a poursuivi pour obtenir une meilleure communication.