Rapport de COO

Java Clavardage

Maud Pennetier et Damien Molina

14 février 2021

Table des matières

Introduction

Le présent document est un rapport de COO concernant le projet d'application bureau de Clavardage développée en Java dans le cadre de l'UF COO/POO de quatrième année à l'INSA Toulouse. Ce document a été réalisé par Maud Pennetier (4IR - SC) et Damien Molina (4IR - SI).

Contexte Un client contacte notre équipe de développement pour développer une application de clavardage pour son entreprise. Certains de ses employés sont connectés sur un réseau interne à l'entreprise, d'autres en dehors de ce réseau (depuis leur domicile par exemple). L'application sera développée en Java.

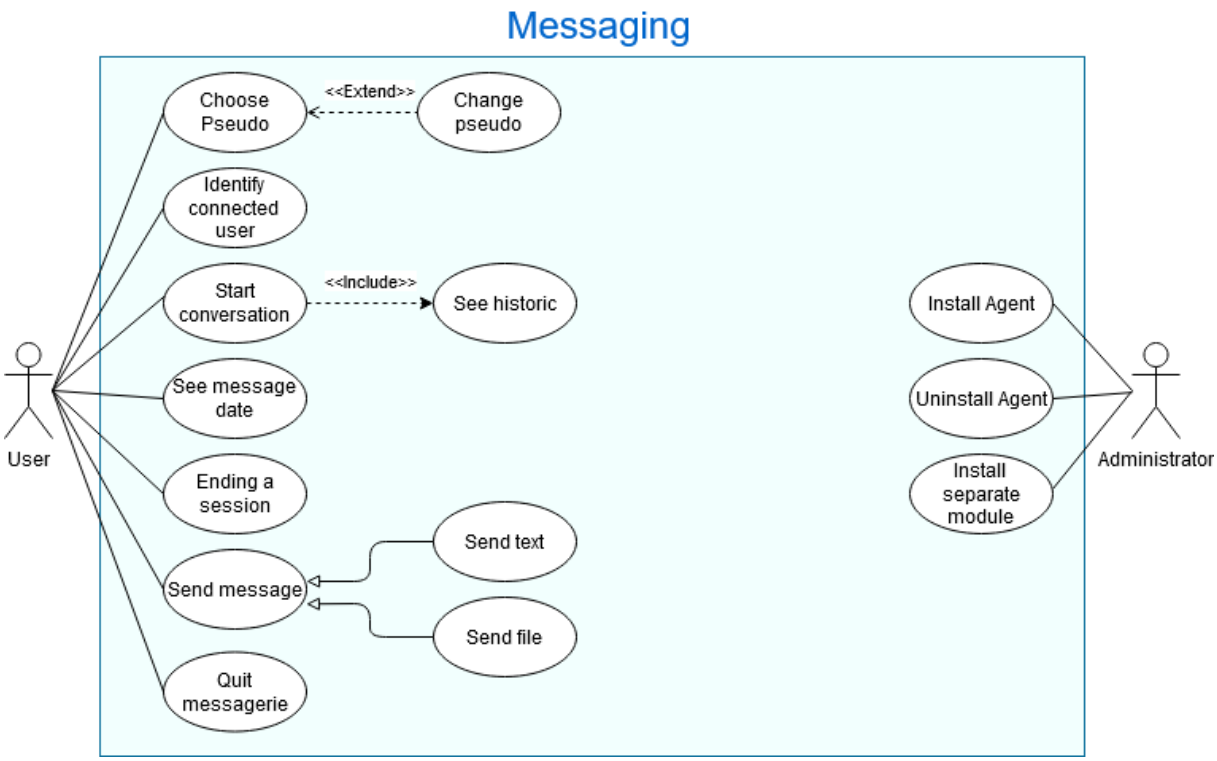
L'ensemble des diagrammes est disponible sur un répertoire Git hébergé sur la plateforme GitHub accessible à l'adresse :
https://github.com/Pythagus/java-messenger/tree/master/docs/modeling.

Chapitre 1

Compréhension préalable du projet

Dans cette partie, nous nous intéressons à la démarche première de conception, le diagramme des cas d'utilisations

Le projet est représenté par le diagramme des cas d'utilisation ci-suivant. On distingue deux utilisateurs : User, la personne qui utilise le client, et Administrator, celui qui installera le client. Les messages envoyés peuvent être de deux types : du texte ou bien un fichier (image, png, pdf, etc).



Chapitre 2

Développement du projet : fonctionnement du code

Cette partie présente les diagrammes de séquence et de classe propre au code écrit. Tout les diagrammes sont consultables en .png et en taille originale sur le répertoire Git.

2.0.1 Interactions globales

Le développement s'est fait en utilisant de très nombreuses classes (plus d'une quarantaine pour le seul client), rendant plus simple leur lecture, mais réduisant la visibilité du graphe des interactions totales. Nous ne présentons donc pas un diagramme de classe général dans ce rapport (un exemplaire est tout de même présent sur le Git). En revanche, un diagramme de séquence en boite noire permet une meilleure compréhension. Ici **Actor** représente un utilisateur, **System** décrit le client d'application de ce dernier, **Network** les autres clients, **TomcatServer** le serveur de présence et **MessageServer** la base de données des messages.

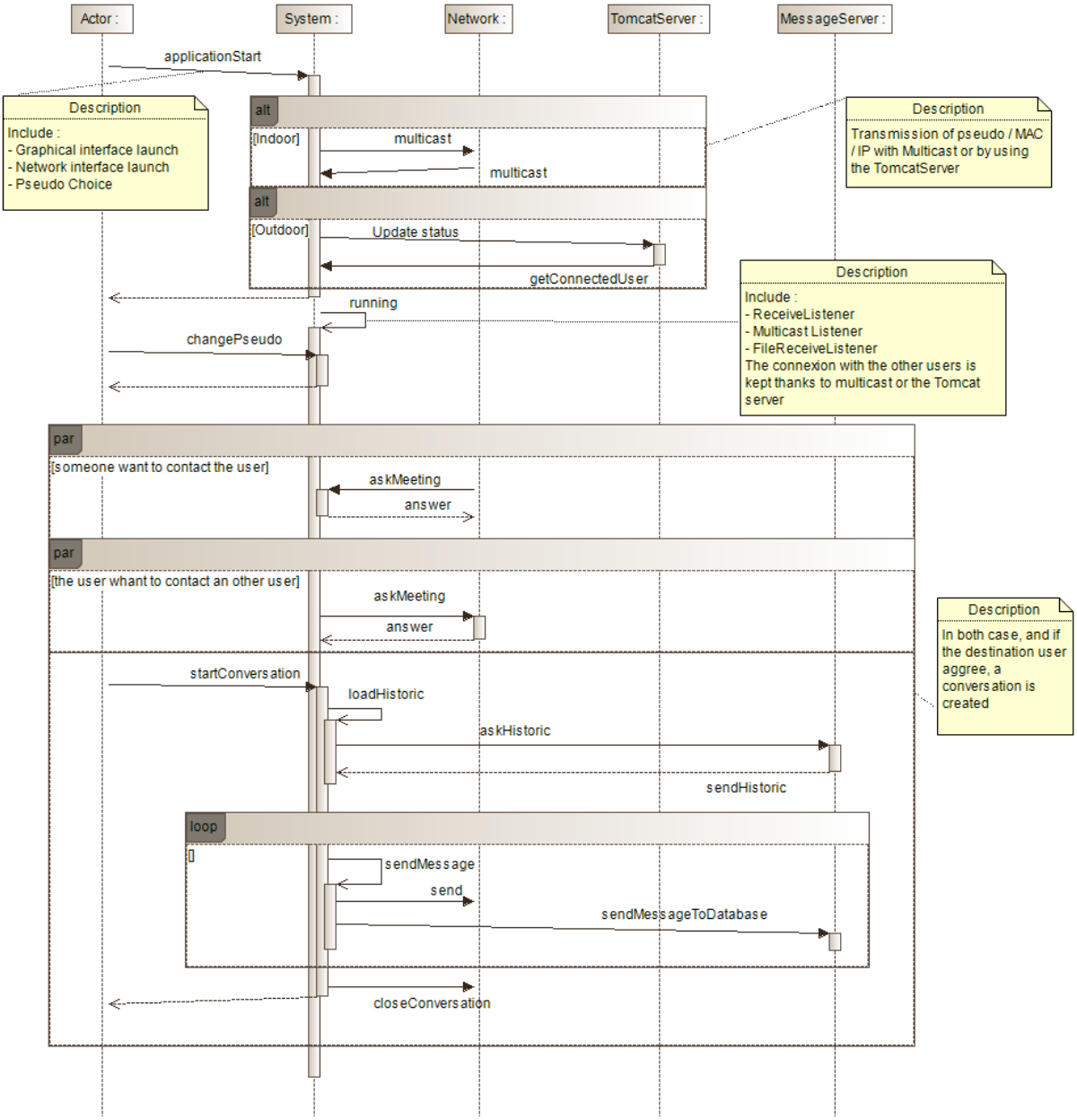


FIGURE 2.1 – Diagramme de séquence global

2.0.2 Diagrammes de séquences

Les diagrammes de séquence suivant représentent le déroulement précis de certaines actions notables de l'application.

— **Lancement du client - Figure 1.2** Ce diagramme présente la séquence suivant le démarrage de la fonction *start()*, comprise dans la classe Launcher. Le but de cette fonction est de démarrer les instances principales du client (l'interface graphique et l'interface réseau notamment).

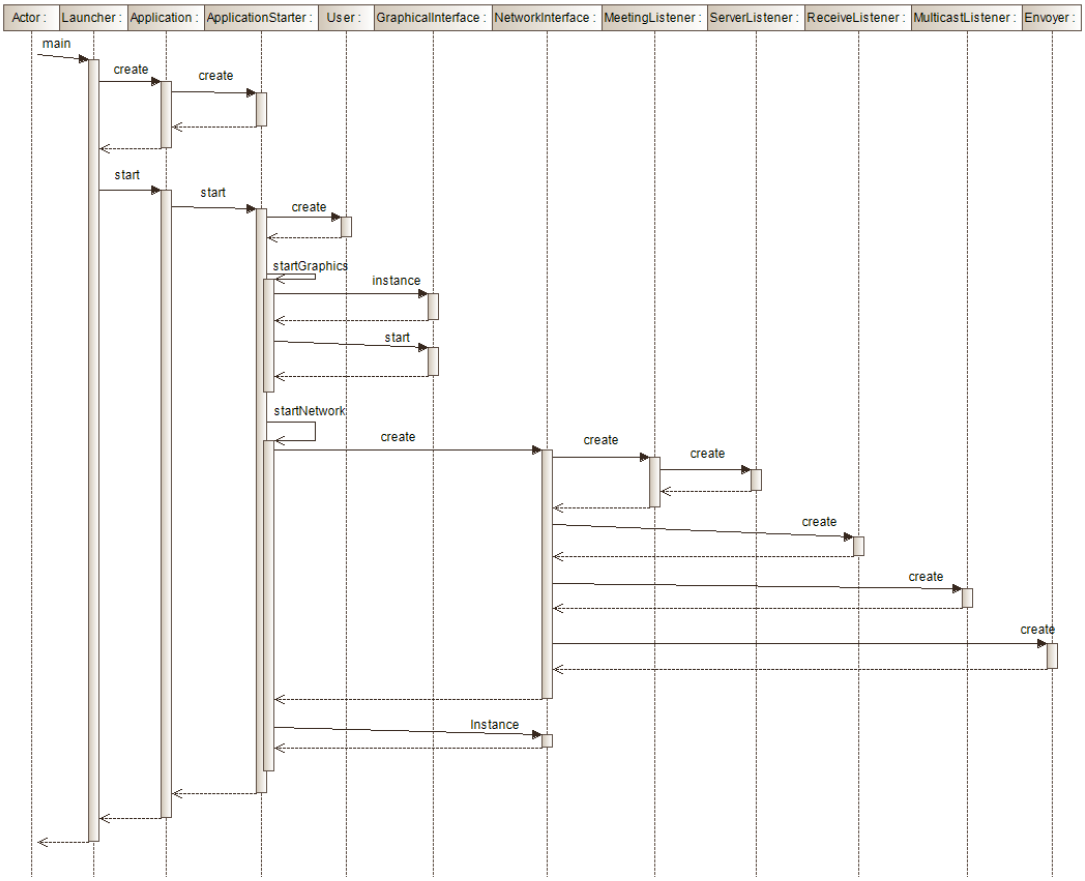


FIGURE 2.2 – Diagramme de séquence de la fonction start()

— **Envoi d'un message - Figure 2.3** L'envoi d'un message avec la fonction send() (présente dans le package MessageEnvoyer et implémentée dans l'interface graphique) est divisée en deux parties successives. Premièrement la création du *packet* qui est lié à un *User*). Secondement, le packet est transmis à l'*Envoyer* qui gère le Stream vers l'autre *User*.

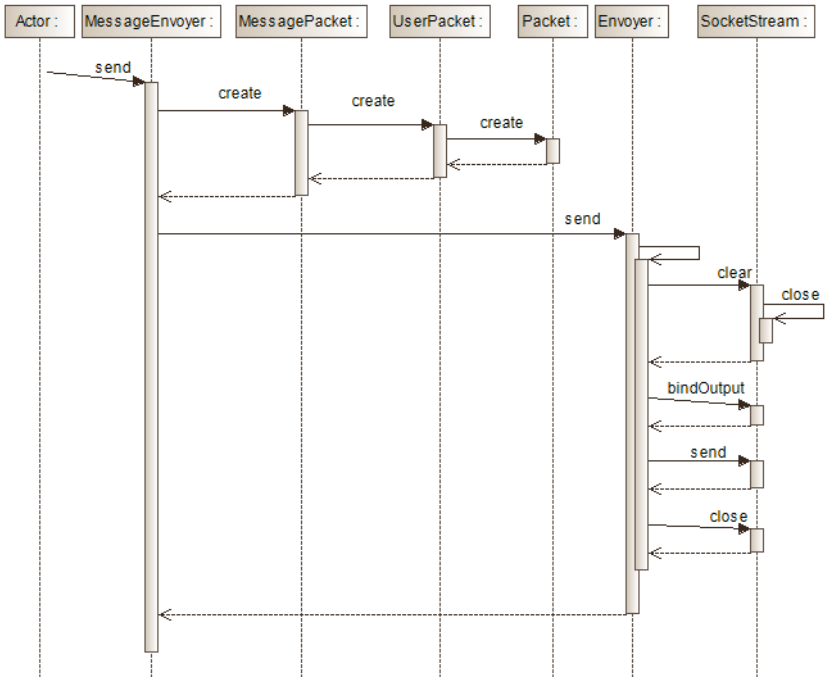


FIGURE 2.3 – Diagramme de séquence de la fonction send()

— **Changement de pseudo - Figure 2.4** Le choix du pseudo est plus contraignant que le changement, le diagramme suivant permet donc d'expliquer les deux actions. La première partie de la séquence concerne la validation du pseudo (i.e. personne d'autre n'utilise le même). Après cette phase, le client charge le reste de l'interface graphique et envoie le pseudo aux autres utilisateurs. Finalement l'utilisateur est connecté au serveur de présence.

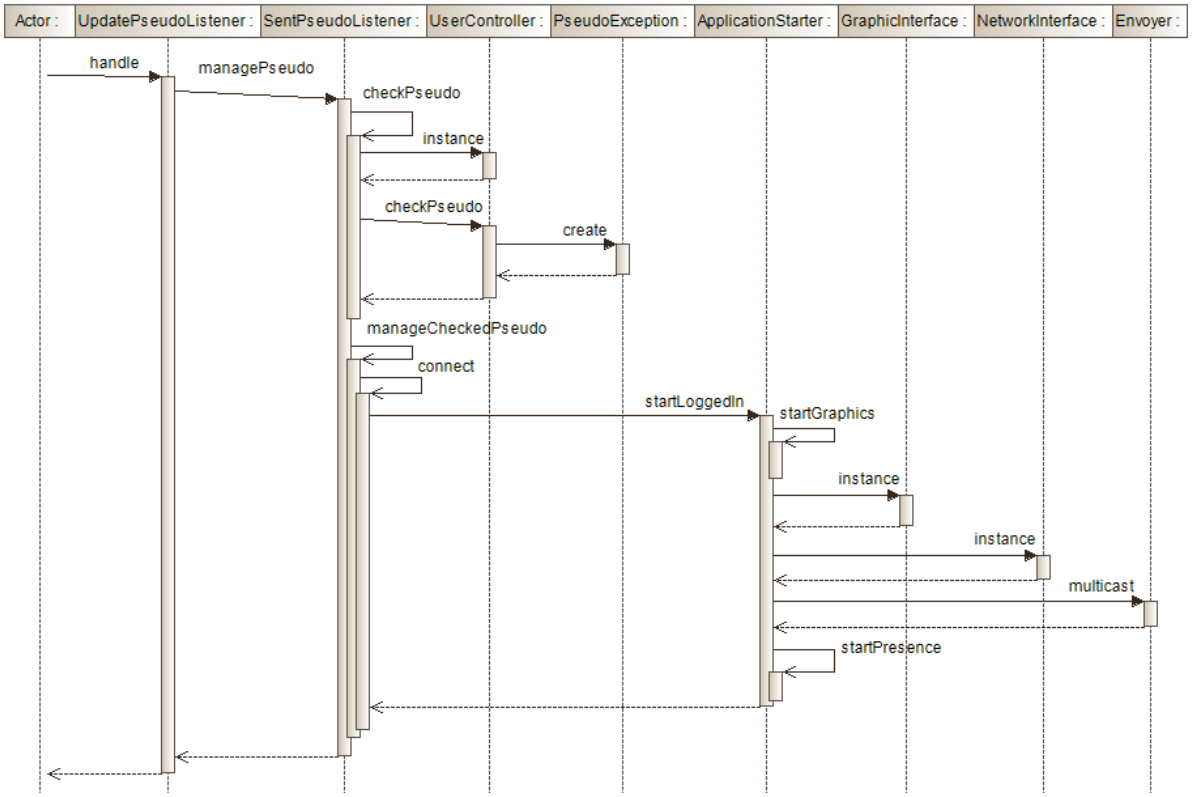


FIGURE 2.4 – Diagramme de séquence du choix de pseudo

2.0.3 Application bureau et diagrammes de classes

Le code est disponible sur le Git dans le dossier `messenger-client`. Les principaux packages du dossier `fr/insa/messenger/client` sont listés ci-après. Les diagrammes de classes de network, http, controllers, models et observers sont disponibles ci-suivant.

— **network** : package regroupant toute l'interaction de l'application bureau avec le réseau. Le package regroupe les *envoyers* et les *listeners* qui permettent respectivement d'envoyer et de recevoir des paquets envoyés par d'autres utilisateurs. La `NetworkInterface` permet de centraliser le démarrage des *threads* réseaux et d'obtenir l'instance d'`Envoyer` permettant de gérer l'envoi de paquets pour un type de message spécifique. Le diagramme étant grand, je conseille de regarder la version .png disponible sur Git.

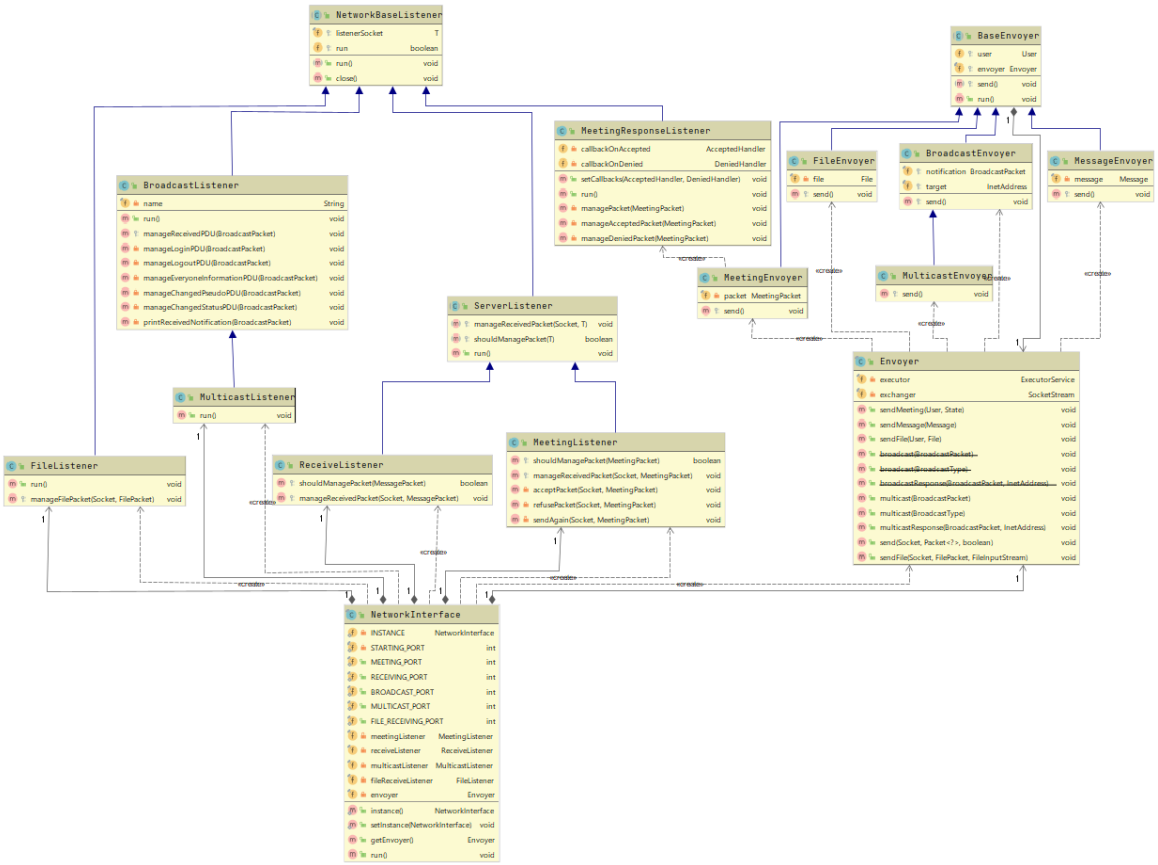


FIGURE 2.5 – Package network

— **http** : package utilisé pour communiquer avec le serveur de présence via requêtes HTTP. La classe `PresenceInterface` contient les deux seules méthodes accessibles par l'utilisateur : *subscribe* et *publish*.

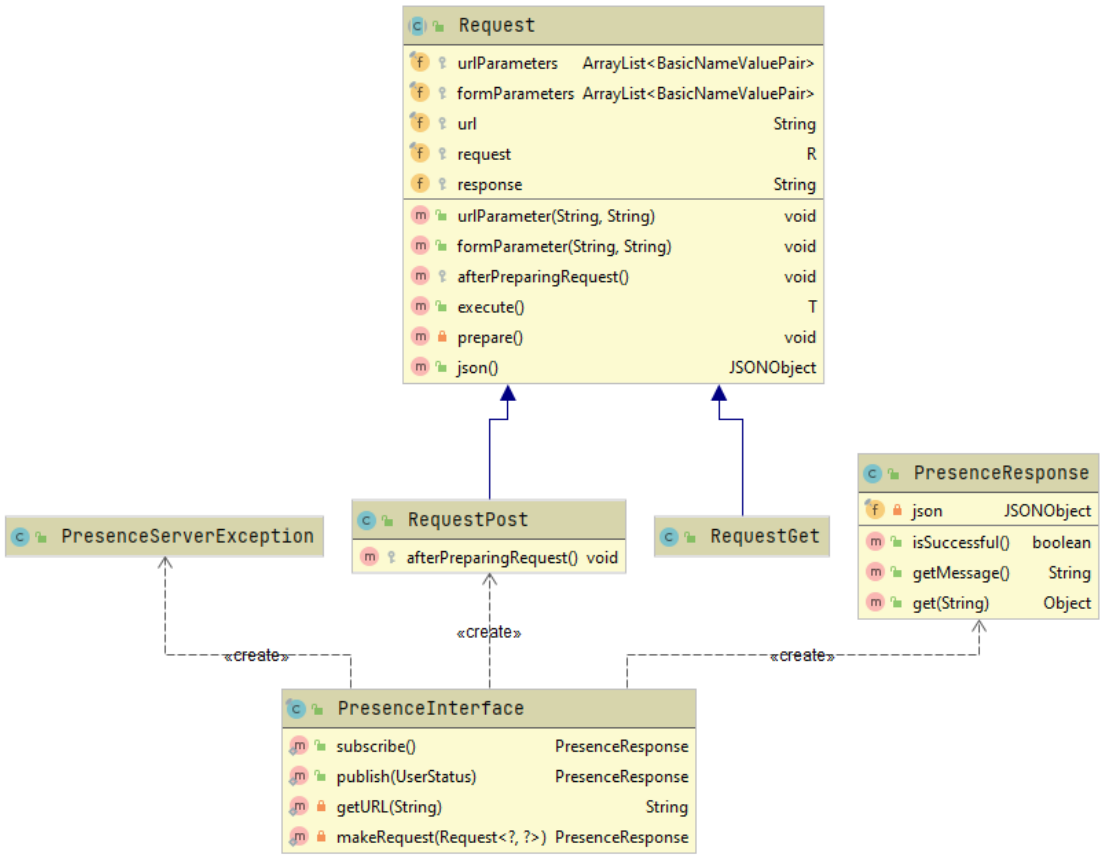


FIGURE 2.6 – Package http

— **controllers** : ensemble des contrôleurs. Ceux-ci sont responsables du contrôle de la validité des données ainsi que du traitement de celles-ci. Vous retrouverez donc les classes `UserController` pour gérer le choix du pseudo et le statut de l'utilisateur, et `ConversationController` pour la gestion des conversations.

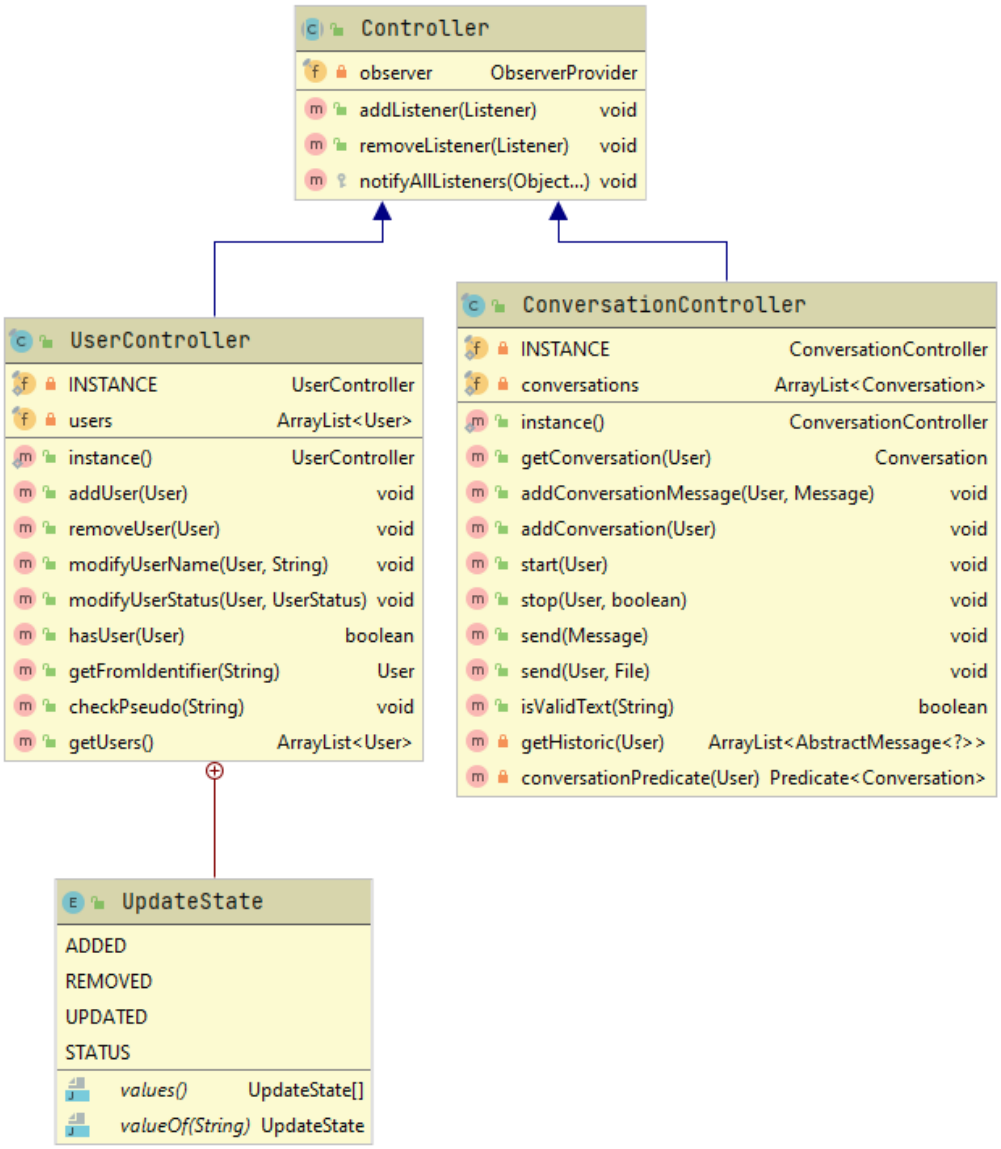


FIGURE 2.7 – Package controller

— **models** : contient les classes définissant les concepts de bases du système utilisés dans le reste des classes. On y trouve les classes `User`, `Conversation` et `Message`.

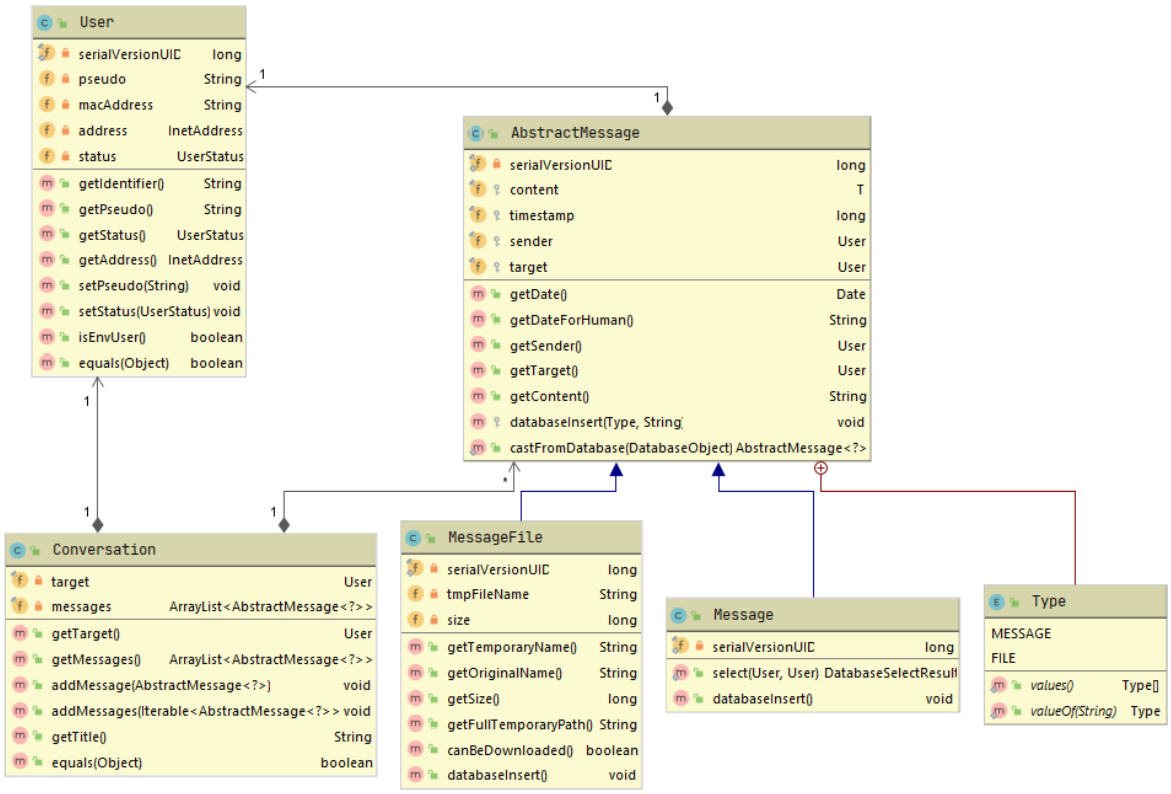


FIGURE 2.8 – Package models

2.0.4 Bases de données

Le code est disponible sur le Git dans le dossier `messenger-tomcat-server`. Le package est relativement léger. Le serveur de présence fonctionne via requêtes HTTP POST. Les routes disponibles sont présentes dans le package `routes`.

Stockage des status

Le serveur de présence contient la base de données ci-suivante. Elle affilie un utilisateur (identifié par une adresse MAC) avec un statut.

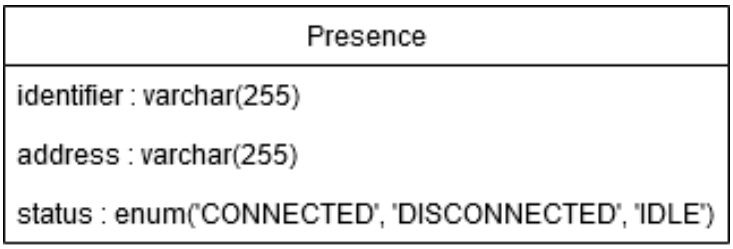


FIGURE 2.9 – Base de données du serveur de présence

Stockage des messages

Le stockage des messages, permettant la récupération d'un historique de conversation, est géré par une base de données centralisés. Les messages sont contenus dans une seule table. Chaque message est identifié par un entier. Il est aussi lié à la source et au destinataire, ainsi qu'à son type (du texte ou bien un fichier) et surtout un champ est destiné à son contenu.

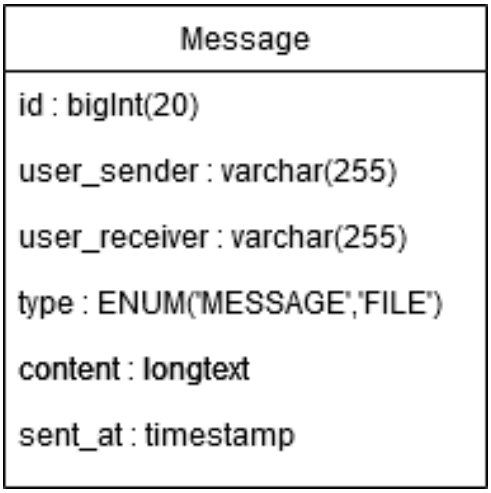


FIGURE 2.10 – Base de données du serveur de messages