Projet Clavardage

Hanne Oy

Nerea Pruneau Marin

Table des matières

[Introduction 2](#_Toc94367763)

[I. Conception de l’architecture du projet 2](#_Toc94367764)

[1. Modélisation UML 2](#_Toc94367765)

[2. Architecture du projet 4](#_Toc94367766)

[II. Processus de développement automatisé 6](#_Toc94367767)

[1. Le rôle de git dans la répartition du travail 6](#_Toc94367768)

[2. Mise en place de la méthode agile grâce à Jira 7](#_Toc94367769)

[3. Automatisation des test grâce à Jenkins 9](#_Toc94367770)

[III. Mode d’emploi de l’application 9](#_Toc94367771)

[Conclusion 10](#_Toc94367772)

# Introduction

Dans le cadre du projet de développement de l’application de clavardage, nous avons pu mettre en place des connaissances sur la Conception et Programmation Orientée Objet. En effet, nous avons tout d’abord réalisé des diagrammes de modélisation UML pour faciliter le développement à posteriori. Puis nous avons développé notre application en suivant l'architecture choisie lors de la conception des modèles UML. Nous avons également mis en place des méthodes de processus de développement pour faciliter l’organisation du travail. Pour cela nous avons utilisé des outils comme GitHub, Jenkins pour la partie développement mais également Jira qui nous a permis de mettre en place un moyen d'organisation de la charge de travail en mettant en place la méthode agile.  Premièrement, nous verrons la conception et l'architecture de ce projet, puis nous verrons l’organisation et les techniques d’automatisation des processus mis en place. Enfin, nous présenterons le mode d’utilisation de notre application.

# I. Conception de l’architecture du projet

## Modélisation UML

* **Diagramme de cas d’utilisations**

Une image contenant texte, carte, intérieur

Description générée automatiquementPremièrement, nous avons réalisé le diagramme de cas d’utilisations, pour bien prendre en compte les spécifications du cahier des charges mais également bien définir les fonctionnalités de notre application.

Comme il est possible de le voir sur le diagramme des cas d’utilisation, on y trouve les fonctionnalités principales de notre application. En effet, démarrer une session implique de choisir un pseudo et si celui-ci est valide alors il est envoyé aux autres utilisateurs. De même lorsqu’un utilisateur change de pseudo s’il et validé il est également envoyé aux autres utilisateurs.

Une fois connecté l’utilisateur a tout le temps accès aux autres utilisateurs connectés et peut ainsi choisir un de ces utilisateurs pour démarrer une conversation.

Lorsqu’une conversation est ouverte, il peut bien évidemment envoyer et recevoir des messages mais également avoir accès à l’historique des messages envoyés précédemment avec cet utilisateur et l’heure à laquelle ces messages ont été envoyés.

Pour assurer ces mécanismes, l’application utilise une base de données décentralisée et commune à tous les utilisateurs.

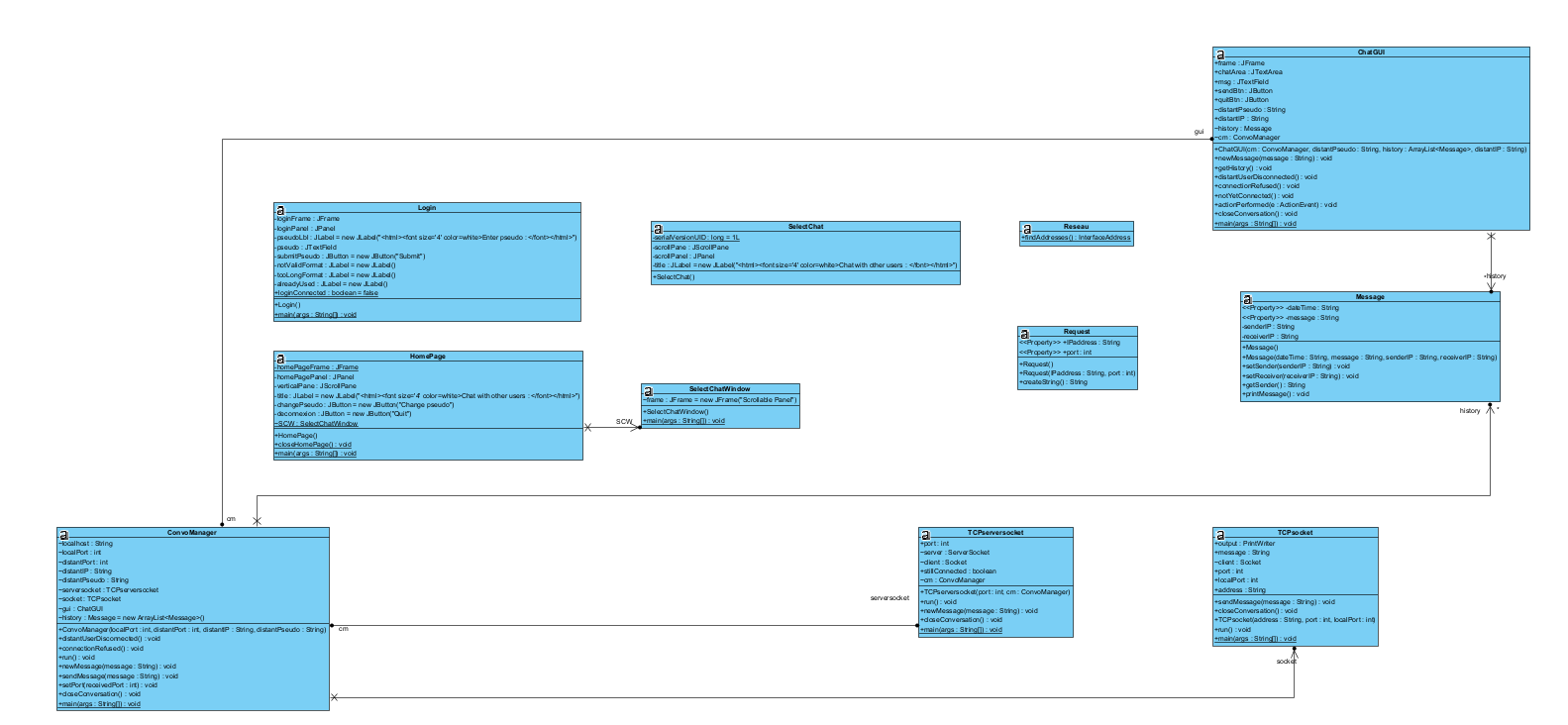
Ainsi, le diagramme des cas d’utilisation nous a permis de définir les fonctionnalités accessibles à l’utilisateur mais également de poser les limites de notre application.

* **Diagramme de classes**

Ensuite, nous avons réalisé le diagramme de classe. En effet, avec ce diagramme nous avons pu faire le lien entre les cas d’utilisations, c’est-à-dire les actions que peut réaliser l’utilisateur sur son application, et les fonctionnalités auxquelles l’utilisateur n’a pas accès, comme la gestion de la connexion au réseau, la gestion des connexions faites avec les différentes bases de données, etc.

Une image contenant texte, capture d’écran, intérieur

Description générée automatiquement



* **Autres modélisations**

Nous avons également réalisé d’autres diagrammes par exemple les diagrammes de classe détaillé, les diagrammes de séquence en boîte noire et en boîte blanche.

Grâce au diagramme de classe détaillé, il a été plus simple pour nous de développer le code puisque les méthodes étaient déjà définies, il ne fallait qu’implémenter le code de ces dernières. Nous avions également modélisé les interactions entre les différentes classes et ainsi nous avons pu faire le lien plus facilement entre les différentes classes à l’heure d’implémenter le code.

De plus, les diagrammes de séquences nous ont permis de vraiment définir l’ordre des événements et comment les mettre en place dans les différentes méthodes. Cela a été très utile au moment de la conception des threads puisque nous avons pu plus facilement gérer la réception de paquets ou de segments lors des connexions UDP et TCP.

Ainsi, la modélisation UML du projet nous a grandement facilité la tâche lors de l’implémentation du code puisque nous avions bien défini les fonctionnalités de l’application, les classes dont nous aurions besoin et la gestion des threads.

Vous pouvez trouver le reste des diagrammes dans le fichier Annexes.

## Architecture du projet

A présent, nous allons voir l’architecture en soit de notre projet. Pour réaliser notre application, nous avons définis 5 paquets principaux contenant des classes liées à chaque paquet. Premièrement nous avons un paquet réunissant toutes les classes nécessaires à l’établissement des connexions TCP et UDP. Puis des classes propres à la construction et la récupération d’informations nécessaires au projet comme la récupération des différents types d’adresses d’une machine sur un réseau, la construction d’un utilisateur, la construction des messages, etc. Troisièmement, nous avons un paquet dédié à l’établissement de connexions avec les bases de données, que se soit la base de données locale ou la base de données distante. Ensuite, nous avons un paquet contenant toutes les classes nécessaires à la création d’interfaces et la gestion de celles-ci. Enfin, le dernier paquet contient les classes qui contrôlent le fonctionnement général de l’application en faisant le lien entre toutes les autres classes.

* **Serveurs et clients TCP et UDP**

Premièrement, nous avons développé les serveurs et les client UDP et TCP puisque ce sont les éléments essentiels à la communication sur un réseau entre des machines déployant la même application.

En effet, les connexions UDP permettent de gérer les connexions et déconnexions des utilisateurs du réseau. Le serveur UDP est ouvert en permanence dès le lancement de l’application, il distingue les paquets indiquant qu’un utilisateur est connecté ou au contraire un avertissement de déconnexion. C’est pour cela que nous avons également 2 types de clients UDP, l’un lançant un envoi d’avertissement de connexion en broadcast chaque 5 secondes (ainsi l’envoie est automatisé et chaque 5 secondes tous les utilisateurs du réseau sont en mesure de percevoir la connexion d’un nouvel utilisateur) et un autre client envoyant un message de déconnexion à la fermeture de l’application.

D’autre part nous avons les classes liées aux connexions TCP, utilisées dans le cadre des demandes de sessions de clavardage et lorsqu’un chat est ouvert. Ainsi, le serveur TCP est ouvert en permanence pour pouvoir écouter lorsqu’une demande de clavardage est lancée et ainsi pouvoir avertir l’utilisateur qu’un autre utilisateur veut communiquer avec lui. Puis lorsqu’une conversation est ouverte, le serveur TCP est ouvert pour pouvoir recevoir des messages en permanence et un client s’ouvre à chaque envoi de message.

* **Classes d’objets du projet**

Nous avons également créé des classes pour définir des objets dont nous aurions besoin pour la gestion complète du fonctionnement de l’application et également de certaines classes nous permettant de faciliter la récupération d’informations.

En effet, premièrement nous avons défini une classe Réseau nous permettant de récupérer plus facilement les adresses IP liées à la machine sur le réseau, en effet grâce à une librairie java nous pouvons récupérer l’adresse liée à l’interface connectée au réseau et ainsi pouvoir plus facilement récupérer cette adresse IP mais également récupérer l’adresse de broadcast.

Ainsi, cette classe sert notamment à la classe user puisque nous avons défini une méthode permettant d’initialiser un utilisateur en lui passant une chaîne de caractère représentant le pseudo et grâce à la classe réseau récupérer l’adresse de la machine et ainsi construire un objet utilisateur.

De plus, nous avons également définit une classes message permettant de gérer toutes les informations liées à l’envoi d’un message comme son expéditeur, son destinataire, la chaine de caractères en soit, la date du message, etc. Et ainsi gérer plus facilement l’envoi et la réception de ces derniers mais également facilitant la récupération de l’historique des conversations.

* **Contrôleur de connexion aux bases de données**

D’autre part, nous avons également dû mettre en place des bases de données pour stocker toutes les informations nécessaires au fonctionnement de notre application. Pour cela nous avons décider de mettre en place une base de données locale à chaque machine pour le stockage d’informations liées aux autres utilisateurs connectés au réseau. Mais aussi d’une base de données centralisée pour le stockage des informations liées à l’envoi de messages.

Premièrement, pour la base de données locale nous avons opté pour SQLite qui permettait de créer simplement un fichier texte à l’intérieur du projet dans lequel il créer la table et ajoute les informations de tous les utilisateurs. Ces tables contiennent un identifiant unique étant la clé de la table puis le pseudo de l’utilisateur et son adresse IP. Lors de la déconnexion d’un utilisateur la base de données et effacée.

Concernant la base de données distante, nous avons utilisé la technologie MySQL puisque nous avons utilisé la base de données du LAAS qui implémente MySQL. Cette base de données stocke les adresses IP des émetteurs et des destinataires, l’heure d’envoi du message et le message en soit. Ainsi, tous les utilisateurs ont accès au même historique de messages à tout moment.

* **Interfaces graphiques**

De plus, nous avons réalisé des interfaces graphiques pour que l’utilisateurs puisse utiliser les fonctionnalités de l’application. Ainsi, dès le lancement de l’application il accède à une interface sur laquelle il peut soumettre un pseudo et si celui-ci est validé alors il accède au menu principal sur lequel il peut choisir de changer de pseudo ou de se déconnecter. Une autre fenêtre apparait également permettant de choisir un utilisateur connecter avec qui ouvrir une session de clavardage. L’autre utilisateur reçoit alors un message d’avertissement et peut accepter ou non la session de clavardage. Dans le cas où celui-ci accepte alors la fenêtre de chat s’ouvre dans laquelle il peut accéder à l’historique des messages et envoyer de nouveaux messages.

* **Manager général de l’application**

Enfin, nous avions besoin d’un paquet manager réunissant toutes les classes manager permettant de faire le lien entre tous les autres paquets et avoir une gestion centralisée de l’application. Les méthodes, les objets et les fonctionnalités définies dans la classe manager.java permettent d’être utilisées dans les GUI pour implémenter les fonctionnalités du projet au niveau de l’interface graphique.

Ainsi, nous avons vu qu’une modélisation UML bien définie permet de faciliter le processus de développement d’une application. Puis nous avons tenté d’organiser notre projet en suivant des groupes de fonctionnalités essentielles à un bon fonctionnement de notre projet.

# II. Processus de développement automatisé

## Le rôle de git dans la répartition du travail

* **Début du projet**

Dès le début du projet nous nous sommes mis d’accord sur l’organisation que nous allions mettre en place pour ce projet. Nous avons tout d’abord ouvert un projet sur GitHub et ouvert une page sur Jira pour organiser notre travail.

Concernant le début du projet nous avons principalement développé ensemble dans le cadre des TPs. En effet, nous devions développer les bases du projet, être d’accord sur le fonctionnement des bases de l’application en respectant le cahier de charges et en s’aidant de la modélisation réalisée dans le cadre de la COO. Ainsi, les fonctionnalités de bases étaient en accord avec nos points de vue et pouvaient être plus facilement utilisées par la suite.

Nous suivions les sprints que nous nous étions fixés et en suivant l’outil Jira. Puis à chaque fin de séance de TP nous pushions le code sur le GitHub du projet.

* **Travail à distance**

Une fois les fonctionnalités de bases développées ensembles nous nous sommes réparti le travail restant puisque toutes les fonctionnalités essentielles étaient claires et nous avions spécifié plus précisément les fonctionnalités de plus haut niveau.

Ainsi l’une s’est occupée de développer les fonctionnalités liées à la connexion au réseau, c’est-à-dire la gestion des connexions UDP et la mise en place d’une base de données locale puis au lancement de l’application grâce à l’authentification. D’autre part, il fallait mettre en place le service d’envoi de messages, c’est-à-dire la gestion des connexions TCP et de la mise en place d’une base de données distante pour le stockage de messages.

Ainsi, nous avons eu une organisation globale qui nous a permis d’avancer efficacement tout en discutant des points précis de développement pour être d’accord sur les fonctionnalités à mettre en place et leur utilisation. Maintenant, nous verrons les outils que nous avons utilisés pour pouvoir garantir cette organisation.

* **Mise en place de git en local**

Dans un premier temps nous avons chacune créer un répertoire sur nos ordinateurs personnels sur lequel nous avons mis en place git localement. Ainsi, nous pouvions avancer de notre côté et pouvoir accéder à des versions précédentes de nos projets.

De plus nous avons également mis en place git directement sur Eclipse, ainsi il était plus simple pour nous d’accéder à l’historique des versions que nous avions faites tout au long du projet.

* **Utilisation d’un repository GitHub**

Néanmoins, nous avons surtout utilisé GitHub puisque cela nous permet d’avoir accès aux versions créées par l’autre personne. En effet, ce que nous avons fait, c’est qu'en plus de la branche Main, nous avons créé une branche chacune pour pouvoir ajouter au fur et à mesure nos versions, et lorsque nous étions satisfaites avec une version pour laquelle nous avions fait des avancements nous l’ajoutions à la branche principale.

Ainsi, git a été un outil très utile pour garder une trace des avancements que nous faisions mais également pour pouvoir travailler à distance et avoir accès aux nouvelles fonctionnalités ajouté par l’autre personne.

Voici un lien vers notre dépôt GitHub : <https://github.com/PMnerea/POO.git>

## Mise en place de la méthode agile grâce à Jira

* **Les rôles**

Tout d’abord, il fallait attribuer les rôles du Scrum. Pour cela nous devions définir notamment les rôles de Scrum Master et de Product Owner. Comme ce projet était à réaliser en suivant un cahier des charges et encadré par des enseignants et nous n’avions pas de réel client nous avons donc défini que le “Product Owner” serait le cahier des charges et que nous pouvions nous aider des conseils et directives supplémentaires donnés par les encadrants du projet. Par conséquent, nous avons pris le rôle de Scrum Master puisque nous étions les responsables du développement complet de l’application.

* **User stories**

Une fois les rôles du Scrum attribués il fallait définir précisément les users stories pour mettre en place une organisation claire et simple. Nous avons donc défini 3 user stories principales :

1. Pouvoir choisir un pseudo et le changer à tout moment
2. Avoir accès à tous les utilisateurs connectés pour discuter
3. Pouvoir discuter avec un utilisateur en particulier et avoir accès à l’historique des messages

Ainsi, nous avons pu bien définir les users stories nécessaires à ce projet, pour ces users stories principales nous nous sommes également aidées des modèles développés en COO, notamment le diagramme de cas d’utilisation.

* **Gestion des sprints**

Néanmoins, nous avions besoin de développer des fonctionnalités sous-jacentes communes à ces users stories, nous avons donc baser nos sprints sur le développement de ces fonctionnalités essentielles au fonctionnement de l’application mais non accessibles directement par les utilisateurs. Ainsi, nous avons divisé le développement de notre projet en 4 sprints :

1. Mettre en place les serveurs et les client TCP et UDP
2. Développer les classes définissant les objets nécessaires à l’application comme la gestion des utilisateurs, leur pseudo et leurs adresses IP mais également la gestion des messages, etc.
3. Mise en place des bases de données, locale et distante.
4. Développement des interfaces graphiques pour permettre la mise en place des fonctionnalités accessibles aux utilisateurs

À la fin de chaque sprint nous faisions un point sur ce qui avait été fait et nous mettions d’accord sur les points à développer pour réaliser le sprint suivant.

* **Utilisation de Jira pour la gestion des sprints**

Comme nous avons vu précédemment, nous définions précisément les objectifs à réaliser à chaque sprint, de cette manière Jira nous a grandement aidé à cela. En effet, nous pouvions définir les tickets dont nous avions besoin dans le backlog et les ajouter au sprint en cours. Ainsi, nous avions des tâches précises à réaliser que nous pouvions nous assigner pour répartir la tâche de travail mais également définir des tâches sous-jacentes à celles déjà présentes, puis il était aussi possible de marquer l’avancement de chaque tâche en cours. De plus, Jira nous permettait également de définir la priorité de chaque ticket pour savoir quelles tâches il fallait réaliser en priorité pour pouvoir avancer.

Ainsi, l’outil Jira a été le moyen de mettre en place plus efficacement toute la méthode agile et d’ainsi organiser plus clairement le travail à réaliser.

Voici le lien vers la feuille de route de notre projet sur Jira : <https://projetpo.atlassian.net/jira/software/projects/PROJ/boards/2/roadmap>

## Automatisation des test grâce à Jenkins

* + **Jenkins**

Tout d’abord, nous avons dû paramétrer les jobs. Premièrement, nous avons intégré les paramètres permettant de relier les projets Maven à Jenkins. Ensuite, nous avons défini que le déclanchement du « build » se ferait à chaque push sur le GitHub, ainsi, nous avons ajouté le lien vers notre dépôt du projet et nous avons mis en place la suppression automatique de l’espace mémoire alloué à un job dès lors que celui-ci avait été exécuté. Enfin, à chaque « build » nous associons le lancement d’une commandante shell nous permettant d’exécuter la compilation du projet Maven.

Sur Jenkins, nous avons créé deux jobs correspondants au fonctionnement des bases de données car c’était les deux classes qui nécessitaient le plus de test et dont nous pouvions vérifier le résultat. Puis nous avons créé le pipeline permettant de connecter ces deux jobs. Le projet est déployé seulement si le « build » automatique lancé lors d’une actualisation du git est un succès.

* + **Tests des connexions UDP et TCP**

Concernant les tests des connexions que ce soit en TCP ou UDP, nous avons ouvert les serveurs sur des terminaux différents pour vérifier que les ports et les buffers étaient bien ouverts et récupéraient les informations dont nous avions besoin. De ce fait, nous savions également que les clients étaient fonctionnels puisqu’ils envoyaient les informations nécessaires.

# III. Mode d’emploi de l’application

Voici le mode d’emploi de l’application :

Une image contenant texte

Description générée automatiquement

Ainsi, il suffit de télécharger le fichier chatApp.jar, changer ses droits d’exécutions depuis un terminal, puis il suffit de l’exécuter à partir de Java 11 en faisant un clic droit. Par la suite, vous arrivez directement sur la page de login sur laquelle vous pouvez choisir votre pseudo. Si le pseudo ne convient pas un message d’alerte vous indiquera qu’il faut que vous changiez le pseudo. Si vous avez réussi à vous connecter vous accédez à la page principale sur laquelle vous pouvez décider de changer de pseudo ou de quitter l’application, de plus vous avez une autre interface graphique sur laquelle vous pouvez voir les utilisateurs actuellement connectés et leur envoyer une requête de démarrage de chat. Si votre interlocuteur accepte cette conversation alors vous pouvez voir et lui envoyer des messages comme sur une messagerie classique.

Dans le cas ou vous vouliez avoir accès au code source et lancer l’application depuis Eclipse c’est toujours possible. Il vous suffit d’avoir Eclipse pour java developpers, importer ce projet en tant que Maven Projet, puis en allant dans le fichier src>main>java>gui>Login.java vous pouvez faire un clic droit et exécuter cette classe comme une java application et vous devriez accéder à la même interface et aux mêmes fonctionnalités. De plus, vous pouvez avoir accéder aux services sous-jacent grâce au terminal puisque vous pouvez voir en temps réel les méthodes appelées pour le fonctionnement de l’application.

# Conclusion

Ainsi, pour ce projet nous avons eu recours à de nombreuses techniques de domaines différents. En effet, nous avons pu travailler sur la modélisation complète du projet en UML pour définir les limites de l’application, l’architecture du projet et faciliter par la suite le développement du code en soit. De plus, grâce à ce projet nous avons pu remettre en place des techniques et des outils que nous connaissions déjà comme git ou encore le travail en équipe mais également de mettre de nouvelles méthodes de travail grâce à la mise en place de la méthode agile et d’outils pour le développement comme peut l’être Jenkins. En effet, git et jenkins permettent de faciliter le travail en équipe et de garder une trace de ce qui est fait, tout en étant bien organisées grâce à la méthode agile qui permet de faciliter la répartition de la charge de travail. De cette manière, nous avons pu mener à bien notre projet.