**Projet de Genie Logiciel**

ChatRoom En Sous Reseau

Un projet de :

Majuran

David

Lohan

Mathis

Luiz

Adam

Readme :

Avant de lancer le projet il faut savoir que pour le bon fonctionnement du projet nous avons crée 2 fichiers jar exécutables : sample-JavaGUI et sample-ServerLauncher

Pour avoir un fonctionnement correct il faut d’abord lancer sample-ServerLauncher et ensuite sample-JavaGUI

NB : IL FAUT LANCER sample-ServerLauncher SEULEMENT UNE SEULE FOIS par contre pour simuler plusieurs client vous pouvez lancer javaGUI plusieurs il vous demandera un nom d’utilisateur à chaque fois

Nous avons seulement reussi à générer un script maven pour build par contre attention il peut seulement générer ServerLauncher ( Expliqué en detail dans la page suivante )

Nous avons pas réussi à générer des scripts pour les test unitaires

Liste des étudiants du Projet + Rôle :

Majuran Murugananthan -> Chef de projet : Correction des Bugs , relecture , mise en place des Packages pour la structure.

Mathis Denis -> Mise en place de l’interface graphique + test Junit

David Ngin -> Implantation du Patron de conception Strategy

Lohan Soreau -> Création du Script Maven et creation de la méthode SEND()

Luiz Ferreira -> Création de la méthode listen pour la classe Server

Adam Dubois -> Création de la méthode isCommand()

Majuran s’est occupé de la mise en place de l’architecture des packages pour faciliter la compréhension du code pour chacun , ensuite la relecture et correction des bugs après que tout le monde ai envoyé leur parti du code , de plus il a travaillé sur les methodes broadcast et listuser qui envoient les réponses de chaque commande à tous les utilisateurs en ligne et aussi la methode listen pour le client qui fonctionne avec les commandes.

Mathis s’est occupé de l’interface graphique pour cela il a utilisé une fonctionnalité présente dans Eclipse qui est le awt swing designer qui permet de gérer les fenêtres plus simplement. De plus il a mis en place le système qui permet de nous informer quand la fenêtre d’un client se ferme qui nous a permis de savoir quel utilisateur s’est déconnecté. Il a mis en place les test Junit mais c’était compliqué avec maven et on savait pas très bien comment tester une interface serveur/client donc on a pu faire seulement 2 tests unitaires

David a conçu le patron de conception , il a attendu la fin de la creation des classes Server et Client pour implémenter ce patron car il a observé que y’a des méthodes qui se répètent mais pas de la même manière il a implémenté ce patron pour les méthodes isCommand() et listen() en créant une nouvelle interface appelé Strategy et une classe appelée Context

Lohan s’est occupé de créer le Projet Maven ainsi que le script en ajoutant toutes les dépendances nécessaires pour le projet : Junit. Il a aussi fait la génération du fichier Jar mais le problème étant que c’est un système client Server on savait pas trop quoi rendre executable donc on a cloné notre projet et créer 2 executables et on a mis les 2 executables dans un seul projet : 1 executable qui lance le SERVER ( IL FAUT LE LANCER QUE UNE SEUL FOIS ) un autre qui lance le GUI Du client celui la on peut le lancer plusieurs fois ce qui nous permettre de faire une conversation entre plusieurs utilisateurs.

Ensuite il a mis en place la partie send pour le serveur client qui permet d’envoyer des socket entre le client et le serveur

Luiz s’est occupé de la méthode listen() pour la classe Server qui utilise un thread pour écouter le port ensuite le socket pourra recevoir le contenu du message contenu dans le packet et convertir ces données en message lisible.

Adam a mis en place les différentes commandes nécessaires pour le bon fonctionnement de l’interface connexion déconnexion. En effet grâce à la commande isCommand() on peut déterminer quelle message correspond à une commande ou non actuellement nous avons 3 commandes : une commande de connexion , deconnexion et une commande qui permet de savoir les utilisateurs connectées.

**Méthode de développement :**

Pour la méthode de développement de projet nous avons choisis un Cycle en V pour faire cela nous avons eu une grande période de réflexion qui consistait à savoir quelles sont les prérequis nécessaires pour la réalisation du projet grâce à ça on a pu définir l’intégralité du projet dès le départ.

Il faut aussi savoir que ici les décisions du clients n’évoluent pas donc le choix du cycle en V est le plus adapté.

Une fois avoir mis en place toutes les fonctionnalités nécessaires nous avons pu commencer la phase de développement qui fonctionnait de manière séquentielle.

Par exemple nous avons commencé par créer la classe Server , ensuite on la relisait pour la correction des bugs ensuite on validait cette étape . La validation nous permet de passer à la suite du développement. Une fois avoir réalisé tous les fonctionnalités on pouvait considérer le projet comme terminé.

**Cahier de charges :**

*Développement d’une application de chat en java entre plusieurs utilisateurs.*

**Objectif :** *Offrir une expérience de chat instantanée entre plusieurs utilisateurs se trouvant sous le même sous-reseau. Un tel projet peut être intéressant pour créer un chat intranet au sein d’un établissement scolaire ou une entreprise ce qui pourrait faciliter la communication d’un groupe de personne.*

**Fonctionnalités du Programme :**

*Création d’un chatroom permettant la communication avec tous les membres présents du chat.*

*Les messages écrits doivent êtres accessibles même lorsqu’on est plus présent dans le chatroom.*

Une fois ces fonctionnalités réalisées on peut améliorer le projet ou rajouter des fonctionnalités comme notamment le cryptage des données pour éviter qu’une personne malveillante puisse avoir accès à nos messages sans notre autorisation.

Liste des Exigences Fonctionnelles :

* *Connexion de plusieurs utilisateurs*
* *Conversations entre plusieurs utilisateurs*
* *Interface graphique avec pseudo*
* *Liste des utilisateurs connectées se mettant à jour à chaque connexion et déconnexion*

Liste des Exigences Non Fonctionnelles :

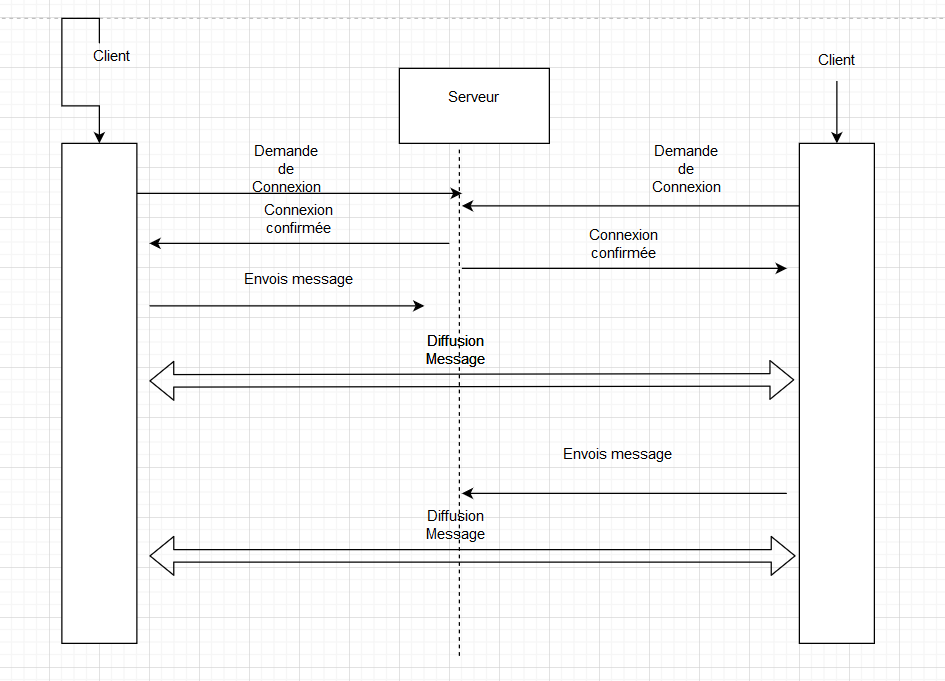
* *Lorsqu’on quitte la salle nos messages sont encore présents mais si on se reconnecte on aura plus accès à ces messages*
* *Pas de sauvegarde de message dans une base de données*
* *Pas de chatroom privé entre 2 personnes*

|  |  |  |
| --- | --- | --- |
| Nom | *Type* | *Description* |
| *Connexion des users* | *F* | *Le logiciel permet la connexion des users dans la salle* |
| *Conversations* | *F* | *Le logiciel permet la communication entre plusieurs utilisateurs* |
| *Interface Graphique* | *F* | *Mis en place d’un interface graphique qui nous permet de dire qui parle* |
| *Liste d’utilisateurs en ligne* | *F* | *Affichage d’une liste d’utilisateurs qui s’actualisent en temp réel* |
| *Sauvegarde Message dans une base de donnée* | *NF* | *Permet la sauvegarde de la conversation lorsque tout le monde a quitté la salle dans un fichier* |
| *Chatroom prive* | *NF* | *Permet la conversation entre 2 personnes MAX sans aucune intrusion d’un autre utilisateur* |

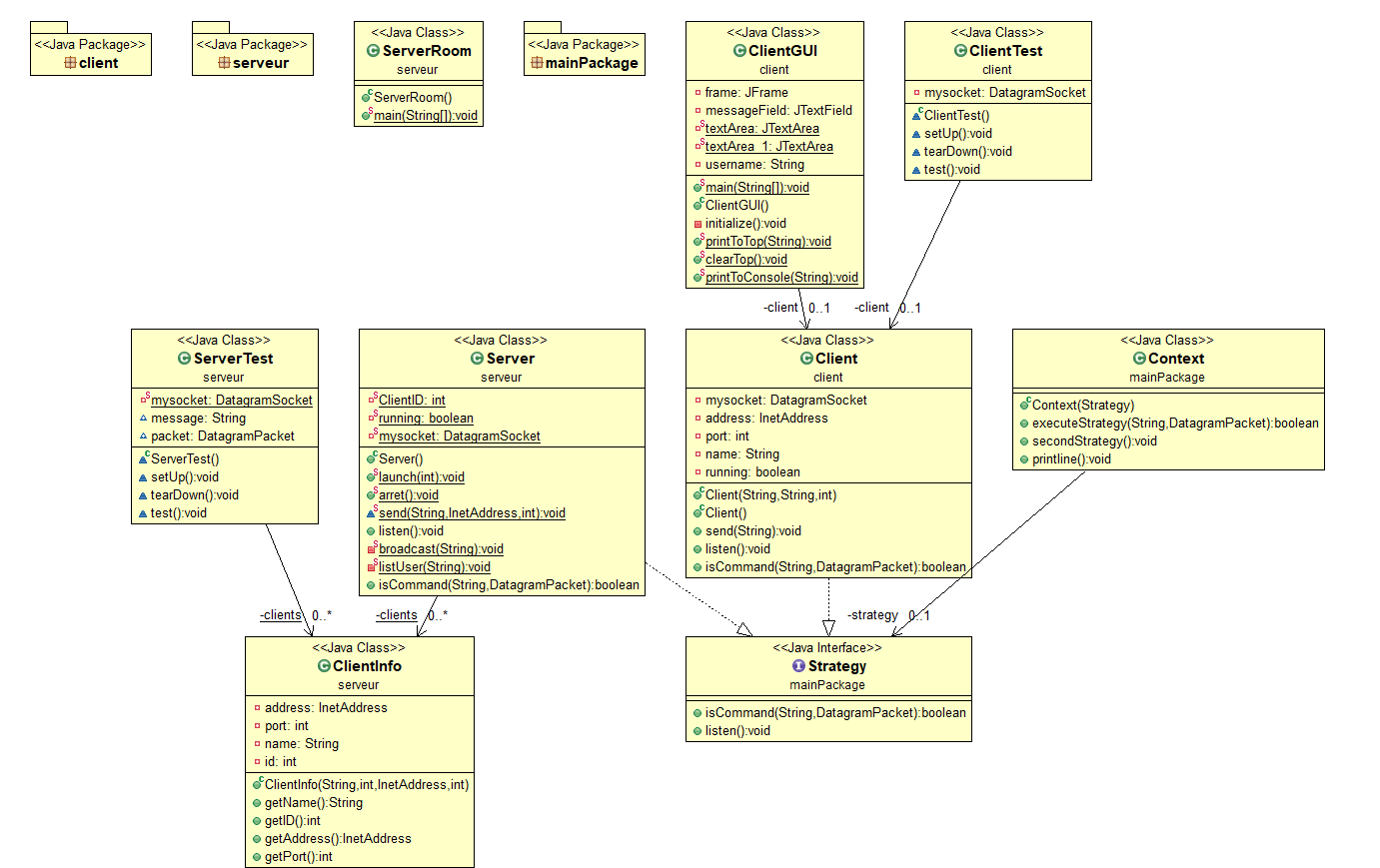
Pour produire la liste d’Exigence nous avons eu une longue période de réflexion car la liste d’Exigence correspond aux éléments attendus par le client donc par ce fait les éléments sur lesquels on doit se concentrer.

De ce fait lors de la lecture nous avons pu mettre en place notre liste d’éxigences

Ci-dessous il y a notre diagramme UML de séquence

**

*Voici maintenant notre diagramme de Classe*

**

Nous avons utilisé comme patron de conception : Le design Pattern strategy car grâce à celui-ci on peut sélectionner des algorithmes présents dans notre code qui se répète à la volée. Et sachant qu’on emploie plusieurs clients et qu’il y a beaucoup d’interchangeabilités entre serveur client grâce a strategy on peut facilement utilliser certaines methodes sans avoir d’erreur ce qui nous facilite le code .