

# *Rapport de projet*

Rédigé par : Malak Senhaji Encadré par : EL Azhari Khadija

2023-2024

# *Introduction*

### L'objectif de ce rapport est de présenter un projet de développement d'une application de serveur multi-thread composée de trois classes interconnectées.

Ce projet vise à faciliter la communication simultanée et fluide entre un serveur et plusieurs clients, en proposant diverses fonctionnalités interactives.

### L'intérêt majeur de ce projet réside dans sa capacité à créer un environnement dynamique et interactif de communication entre le serveur et les clients, grâce à l'utilisation de l'architecture multi- thread.

Cette approche permet au serveur de traiter simultanément les requêtes de plusieurs clients, offrant ainsi une expérience utilisateur réactive et fluide.

### Dans ce rapport, nous analyserons les résultats obtenus lors de l'exécution de chaque classe, en mettant en évidence leurs points forts ainsi que les améliorations possibles.

Enfin, nous explorerons les perspectives d'évolution de ce projet en envisageant l'ajout de fonctionnalités supplémentaires et en discutant des utilisations potentielles dans des scénarios réels.

*Les étapes pour la réalisation du projet*

*La phase d'analyse des exigences :*

A joué un rôle essentiel dans la compréhension approfondie des besoins et des fonctionnalités attendues de l'application. Cette étape a permis de définir clairement les objectifs du projet et d'identifier les fonctionnalités principales à mettre en œuvre. Une analyse approfondie a été réalisée afin de garantir une compréhension complète des exigences, ce qui a servi de base solide pour le développement ultérieur de l'application. En comprenant pleinement les besoins des utilisateurs, nous avons pu définir les priorités et les points clés à aborder dans la conception et l'implémentation du projet.

## *La conception de l'architecture :*

De l'application de serveur multi-thread. Cette étape cruciale a impliqué la création d'une structure appropriée, comprenant la définition des classes principales, leurs relations et les mécanismes de communication essentiels entre le serveur et les clients. En élaborant cette architecture, nous avons veillé à ce qu'elle soit bien adaptée pour répondre aux besoins fonctionnels et assurer une communication fluide et efficace. La conception a été réalisée en tenant compte des bonnes pratiques et des principes de conception logicielle, afin de garantir la scalabilité, la modularité et la maintenabilité de l'application.

## *Implémentation de la classe "MyServerMultiThread" :*

Cette classe a joué un rôle crucial dans l'établissement du serveur en utilisant la classe ServerSocket, ainsi que dans l'acceptation des connexions des clients et l'attribution d'un identifiant unique à chaque client. En mettant en place cette classe, nous avons pu créer un environnement stable pour la communication entre le serveur et les clients. Elle a été conçue pour gérer de manière efficace les connexions entrantes, permettant ainsi une interaction simultanée avec plusieurs clients de manière ordonnée. L'implémentation de cette classe a constitué une étape fondamentale pour la mise en place de la fonctionnalité de base de notre application de serveur multi-thread

## *Implémentation de la classe "Communication" :*

Au sein de la classe "MyServerMultiThread", nous avons développé la classe interne "Communication". Cette classe a été spécifiquement conçue pour gérer la communication bidirectionnelle entre le serveur et chaque client individuel. Son rôle principal était de faciliter l'échange de messages en utilisant les flux d'entrée et de sortie. Grâce à cette classe, le serveur a pu recevoir les requêtes et les données provenant des clients, et également envoyer des réponses ou des informations spécifiques à chaque client. La classe "Communication" a joué un rôle essentiel dans l'établissement d'une communication fluide et réactive, garantissant ainsi une interaction efficace entre le serveur et les clients. Son développement a permis de créer un mécanisme robuste pour la transmission des informations et a contribué à l'expérience utilisateur globale de l'application.

## *Implémentation des fonctionnalités spécifiques*

Dans la classe "PlayWithServer", nous avons implémenté un jeu de devinettes qui permet aux clients de proposer des nombres et au serveur de donner des indications pour les aider à deviner correctement.

Dans la classe "ChatWithServer", nous avons mis en place un système de chat qui permet aux clients de communiquer entre eux en envoyant des messages textuels au serveur, qui les redistribue ensuite à tous les autres clients connectés. Cela crée un environnement de chat en temps réel, favorisant l'interaction et le partage d'informations entre les utilisateurs.

## *Test et débogage :*

Après avoir terminé l'implémentation, des tests approfondis ont été effectués pour garantir le bon fonctionnement de l'application. Divers scénarios ont été couverts afin de tester la stabilité et la fiabilité de l'application. Des erreurs éventuelles ont été identifiées et corrigées pendant le processus de débogage. Ces tests rigoureux ont permis de vérifier que l'application fonctionne comme prévu, en assurant une expérience utilisateur fluide et sans dysfonctionnement. Les corrections apportées ont permis d'améliorer la qualité de l'application et de garantir sa robustesse lors de son utilisation réelle.

## *Documentation :*

Une documentation complète a été créée pour décrire l'architecture, les fonctionnalités et l'utilisation des classes. Cette documentation inclut des explications détaillées du code, des exemples d'utilisation et des instructions pour l'exécution de l'application.

***Modèle-Vue-Contrôleur***

## *Modèle (Model) :*

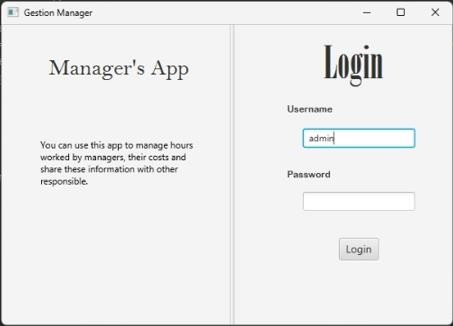
La classe "MyServerMultiThread" assume le rôle central en tant que modèle principal de notre application, responsable de la gestion du fonctionnement du serveur multi-thread. Elle coordonne les différentes fonctionnalités du serveur et assure une communication fluide entre les clients et le serveur.

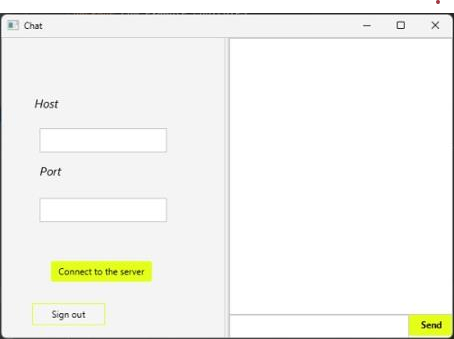
D'autre part, la classe "PlayWithServer" est dédiée à la mise en œuvre d'un modèle spécifique de jeu de devinettes. Elle se charge de la logique du jeu, incluant la génération d'un nombre secret et l'évaluation des réponses fournies par les clients.

La classe "ChatWithServer" constitue également un modèle spécifique pour le système de chat. Elle assume la responsabilité de gérer la communication et la diffusion des messages entre les clients connectés, favorisant ainsi les échanges entre les utilisateurs.

En somme, ces trois classes interconnectées, "MyServerMultiThread", "PlayWithServer" et "ChatWithServer", jouent des rôles essentiels dans notre application. La première assure le bon fonctionnement du serveur multi-thread, tandis que les deux autres se concentrent sur des fonctionnalités spécifiques telles que le jeu de devinettes et le système de chat, permettant ainsi une expérience interactive et engageante pour les utilisateurs.

## *Vue (View) :*





*Contrôleur (Controller) :*

Dans chaque modèle, la classe "Communication" joue le rôle de contrôleur. Elle agit comme une interface entre le modèle et la vue, gérant les interactions entre le serveur et les clients.

Les instances de la classe "Communication" sont responsables de la réception des demandes des clients, telles que les devinettes ou les messages de chat, et effectuent les opérations nécessaires sur le modèle correspondant. Elles traitent les requêtes des clients et communiquent avec le modèle pour obtenir les réponses appropriées.

Les réponses du modèle, telles que les indications de devinettes ou les messages de chat, sont renvoyées aux clients via les flux de sortie appropriés. La classe "Communication" assure ainsi la transmission bidirectionnelle des informations entre le serveur et les clients, garantissant une communication fluide et réactive.

En résumé, la classe "Communication" remplit un rôle essentiel en tant que contrôleur dans chaque modèle. Elle facilite les interactions entre le serveur et les clients en gérant les demandes, en effectuant les opérations nécessaires sur le modèle et en renvoyant les réponses appropriées aux clients. Cette structure assure une communication efficace et transparente dans l'application.

***Explications des fonctionnalités de mon application***

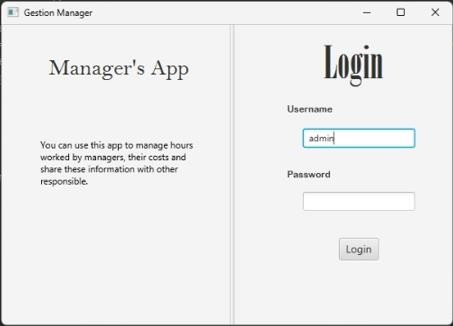
*Serveur multi-thread (MyServerMultiThread):*

La fonctionnalité du serveur multi-thread permet au serveur de gérer simultanément plusieurs connexions clientes en utilisant des threads dédiés. Chaque client connecté reçoit un numéro de client unique pour identifier ses interactions distinctes. Le serveur établit la communication avec les clients en échangeant des messages via les flux d'entrée/sortie.

*Jeu de devinettes (PlayWithServer):*

Le jeu de devinettes est une fonctionnalité offerte aux clients. Le serveur génère un nombre secret aléatoire. Les clients soumettent leurs propositions de nombres, et le serveur compare ces propositions au nombre secret. En fonction de la comparaison, le serveur envoie des indications (supérieur, inférieur, correct) aux clients pour les guider vers la bonne réponse. Lorsqu'un client trouve la bonne réponse, le serveur annonce le gagnant et met fin au jeu.

*Authentification :*





L'application utilise des sockets pour établir les connexions réseau entre le serveur et les clients. Lorsqu'un client se connecte au serveur, une nouvelle instance de la classe Communication est créée pour gérer la communication avec ce client spécifique. Chaque instance de Communication s'exécute dans un thread séparé, permettant ainsi un traitement asynchrone des requêtes des clients.

La communication entre le serveur et les clients s'effectue par le biais de flux d'entrée et de sortie. Le serveur reçoit les messages provenant des clients et les transmet aux destinataires appropriés en fonction du contenu du message. Si un message contient le symbole "=>", il est analysé pour extraire le numéro du client destinataire et le contenu du message. Dans le cas contraire, le message est envoyé à tous les clients connectés, assurant ainsi une diffusion à large échelle des messages.

Cette approche de communication basée sur les flux d'entrée et de sortie permet une interaction fluide et en temps réel entre le serveur et les clients. Chaque client peut envoyer des messages, et le serveur se charge de les acheminer aux destinataires appropriés, facilitant ainsi les échanges d'informations entre les utilisateurs connectés à l'application.

