Rapport Chat System

Gauthier Cassany, Arnaud Panont

5 février 2020

Table des matières

1 Choix d'architecture et design

1.1 Design Client MVC

Le client est structuré de façon à suivre le design pattern "Model-View-Controller". Ce pattern permet de séparer la partie code logique de l'interface graphique, le controller effectuant la passerelle entre les deux, diminuant ainsi l'interdépendance et améliorant la flexibilité de l'application.
Par exemple, il serait possible de changer la technologie utilisée pour l'interface graphique en changeant peu ou pas le code gérant la logique de l'application (communication, interactions avec les bases de données, ect) .

1.2 Webservice REST & Servlet

Le service de présence a été conçu comme une API REST. Une API REST permet de faciliter la communication entre un client et un serveur en utilisant des requêtes http. Ainsi, nous avons pu facilement faire communiquer l'application de bureau client avec le service de présence de cette façon, en utilisant le format de donnée Json pour transférer les données nécessaires entre les deux modules.

Ce type d'architecture est également flexible, car il serait par exemple possible de réutiliser le service de présence pour une potentielle application mobile ou application web, l'API REST étant juste une interface, qui n'est pas liée à un certain type d'application.

2 Choix de technologies

2.1 Interface Utilisateur : Swing

L'interface graphique a été programmée avec la technologie swing. Cette technologie est mature et bien documentée, ce qui nous a permis de nous familiariser rapidement avec les méthodes d'utilisations et les bonnes pratiques.
En complément, nous avons également utilisé le plugin eclipse "Windows Builder" pour faciliter la conception des fenêtres et l'agencement des composants.

2.2 Base de données centralisée MySQL

Pour s'assurer que les utilisateurs sélectionnent bien un nom d'utilisateur unique lors de la création du profil ou lors d'un changement de nom d'utilisateur, il est nécessaire de stocker l'entièreté des utilisateurs utilisant l'application dans une base de données accessible à tous. C'est pourquoi nous avons choisis d'utiliser une base de données centralisée MySQL pour stocker les utilisateurs.

2.3 Base de données locale Sqlite

Il n'est pas nécessaire que l'historique des messages spécifique à chaque utilisateur soit mis en commun à tous les utilisateurs. Une base de données locale étant suffisante pour sauvegarder l'historique, nous avons choisi d'utiliser une base de données SQLite pour cela. Ce type de base de données est facile à utiliser, légère et ne nécessite pas de configuration au préalable. Cela permet également d'être moins dépendant de la base de données centralisée dans les cas où la liaison avec cette dernière n'est plus accessible.

2.4 Sérialisation avec les librairies Jackson et Gson

Nous avons utilisé le format Json pour les communications avec le service de présence et les communications locales et distantes entre les utilisateurs. Ce format est très populaire et particulièrement adapté pour les transferts de données avec les API REST. Nous avons utilisé la librairie "Jackson" pour les communications entre utilisateurs et "‘ Gson "‘ pour les communications avec le service web, car les contraintes de tailles pour déployer le service web sur le serveur tomcat du Gei nous permettaient d'utiliser uniquement la librairie "Gson" qui était plus légère. Nous avons préféré utiliser jackson pour le reste des sérialisations car plus stable et plus documentée.

3 Manuel

3.1 Déploiement Client

Éléments requis :
- Maven
- Git
- Java JDK 11 ou plus récent
- MySQL remote server

Client :
- Aller sur https ://github.com/Mozenn/ChatSystem et cloner le projet avec la commande git clone
- Dans le dossier Chatsystem/src/main/resources, le fichier app.properties permet de configurer le client en fonction des spécificités de l'environnement de déploiement.
- Les propriétés importantes à configurer sont les suivantes :
  - dbURL : l'url jdbc complète de la base de données sous la forme jdbc :mysql ://hostname :Port/DatabaseName
  - dbLogin : login du compte utilisateur utilisé pour cette application
  - dbPassword : mot de passe du compte utilisateur utilisé pour cette application
  - presenceServiceURL : url pour atteindre le service de présence sous la forme :

http ://hostname/presenceservice/users
  - downloadPath : définit où les fichiers téléchargés seront placés
  - embeddedDB : si "true", le stockage des utilisateurs ne sera plus effectué sur la base de données centralisée mais dans la base de données locale. L'unicité des noms d'utilisateurs n'étant plus assurée si cette propriété est placée à "true"
- Dans le dossier ChatSystem où se trouve le fichier pom.xml, exécuter la commande "mvn package"
- Récupérer le .jar avec "‘ with -dependencies "‘ dans le nom, généré dans le dossier ChatSystem/target
- Le client peut s'ouvrir en double cliquant sur le .jar ou en exécutant la commande "java -jar client.jar"

Base de données :

- Installer le client mysql version 5.7 disponible ici : https ://dev.mysql.com/downloads/mysql/5.7.html
- Dans un terminal, connectez vous à votre serveur mysql avec la commande suivante : mysql -h hostname -u login -p
- Il est nécessaire au préalable de créer un utilisateur avec les droits pour créer une base de données ou avec une base de données déjà créée par l'administrateur du serveur
- Créer une base de données si nécessaire avec la commande suivante : Create Database databaseName

3.2 Déploiement Service de présence

Éléments requis :

- Maven
- Git
- Java JDK 11 ou plus récent
- Serveur Tomcat

Étapes :

- Aller sur https ://github.com/Mozenn/ChatSystem et cloner le projet avec la commande git clone
- Dans le dossier PresenceService où se trouve le fichier pom.xml, exécuter la commande "mvn install"
- Récupérer le .war, générer dans le dossier Presenceservice/target
- Se connecter à votre serveur tomcat en utilisant un navigateur web
- Déposer le .war précédemment généré

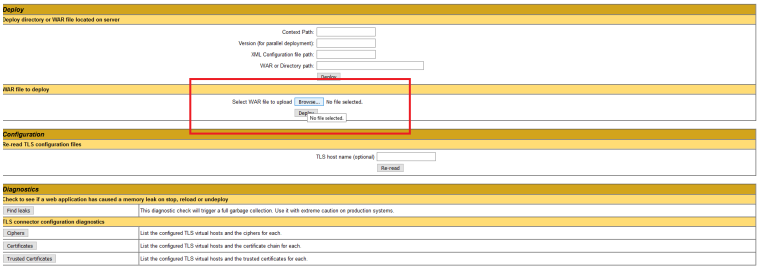

FIGURE 1 –

4 Utilisation

4.1 Création du profil

La création du profil utilisateur s'effectue lors de la première utilisation. Il suffit de rentrer le nom d'utilisateur dans le champ prévu à cet effet.

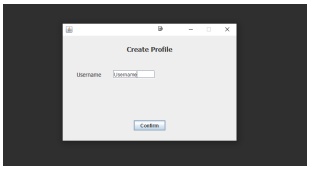

FIGURE 2 –

Le nom d'utilisateur doit avoir une longueur de 1 à 20 caractères et ne doit pas être utilisé par un autre utilisateur.

4.2 Changement de nom d'utilisateur

Dans le menu "Edit" de la barre des tâches, cliquer sur le bouton "Change Username"

Rentrer le nouveau nom d'utilisateur dans le champ prévu à cet effet.

Le nom d'utilisateur doit avoir une longueur de 1 à 20 caractères et ne doit pas être utilisé par un autre utilisateur.

4.3 Changement de l'emplacement de téléchargement des fichiers

Dans le menu "Edit" de la barre des tâches, cliquer sur le bouton "Settings"

Cliquer sur le bouton "Edit" et choisir un nouveau dossier

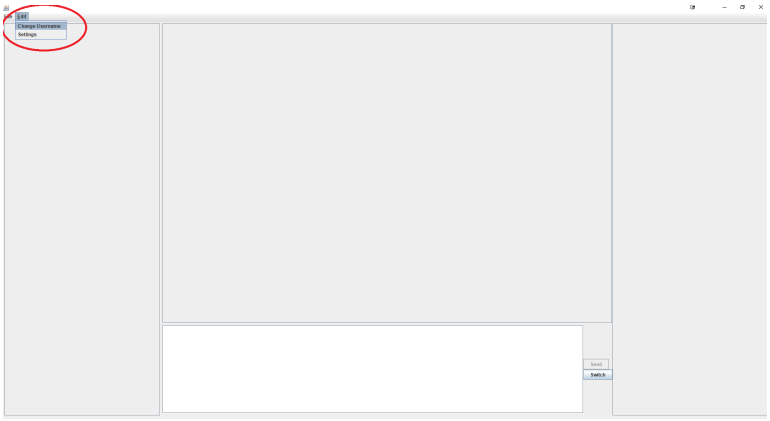

FIGURE 3 –

4.4 Démarrer une session de chat

A partir de la fenêtre principale, appuyer sur le bouton "Start Session" d'un utilisateur connecté

La fenêtre de chat va s'actualiser et afficher cette nouvelle session en tant que session active

4.5 Fermer une session de chat

A partir de la fenêtre principale, appuyer sur le bouton "Close Session" d'une session en cours

La fenêtre de chat va s'actualiser et la session n'est pus active

4.6 Envoyer un message textuel

A partir de la fenêtre principale, et lorsque qu'une session de chat est en cours, saisir le texte à envoyer dans le champ textuel en bas de la fenêtre

Appuyer sur le bouton "Send" pour envoyer le texte saisi à l'utilisateur participant à la session en cours.

4.7 Envoyer un fichier

A partir de la fenêtre principale, et lorsque qu'une session de chat est en cours, appuyer sur le bouton switch pour faire apparaitre le menu d'envoi de fichier

Appuyer sur le bouton "Join" pour ajouter un fichier

Appuyer sur "Send" pour envoyer le(s) fichier(s) joint(s)

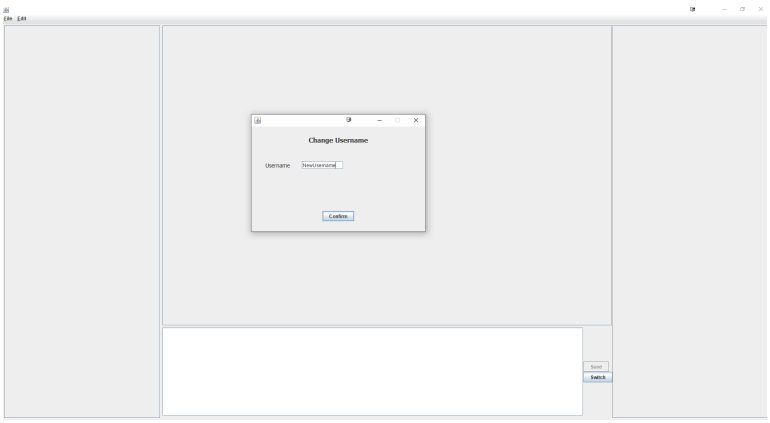

FIGURE 4 –

5 Procédures de tests et de validation

5.1 Tests Unitaires

Les tests unitaires permettent de tester des fonctions ou des unités de façon isolée, sans dépendance extérieure, pour s'assurer de leurs bons fonctionnements au fur et à mesure de l'évolution de l'application.

Nous avons effectué des tests unitaires sur les fonctions de sérialisation, car leur non-fonctionnement peut passer souvent inaperçu durant un long moment.

5.2 Tests d'intégration

Les tests d'intégration permettent de tester une fonctionnalité du système dans une situation similaire à une situation d'utilisation réel.

5.2.1 Test du service de présence

Le service de présence a d'abord été testé en isolation avec des outils générateurs de requêtes http tels que RESTED. Cela nous a permis de s'assurer du fonctionnement du webservice indépendamment du client, avec la possibilité d'envoyer un nombre important de requête pour tester la robustesse du service.

5.2.2 Test client

Le test des communications locales a été effectué avec plusieurs PC sur un même réseau local.

Pour tester les communications avec un client distant, utilisant le service de présence pour la phase de découverte, nous avons modifier notre programme de façon à ce qu'il n'utilise plus les fonctionnalités de découverte locale. Ce paramètre "test" peut-être changer de façons simple en modifiant une propriété

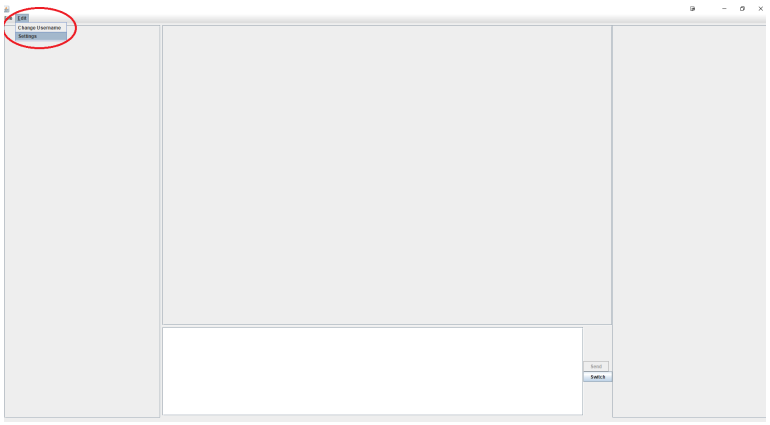

FIGURE 5 –

du fichier config.properties. Ce mode "test" nous permet également de lancer plusieurs clients distants sur une même machine, car toutes les dépendances liées à la machine locale comme l'id ou le dossier de l'application on était remplacées.

5.3 Automatisation avec Maven

Grâce à maven, nos tests s'exécutent automatiquement lors de la génération du .jar. Ceci nous assure que ce que nous testons fonctionnement sans manuellement lancer les tests.

6 Améliorations possibles

6.1 Authentification pour le service de présence

Une fonctionnalité d'authentification permettant de filtrer les requêtes arrivant au service web permettrait d'avoir un système plus sécuriser et moins vulnérable aux attaques de déni de service.

6.2 Test

L'ajout d'un panel de test unitaire plus important, couvrant la majorité des fonctions de l'application permettrait un débogage plus rapide. Ces tests pourraient être facilités avec l'utilisation de framework de mock tel que mockito.

6.3 Diminuer interdépendance et principes SOLID

Respecter et appliquer de façon plus stricte les principes SOLID permettrait d'avoir une application plus évolutive et plus facile à modifier. Par exemple,

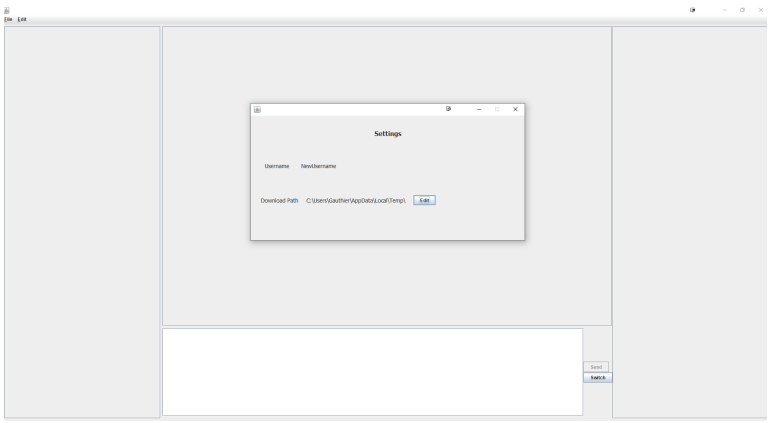

FIGURE 6 –

diminuer la taille des méthodes pour qu'elle n'effectue qu'une tache précise, et séparer des classes importantes comme CommunicationSystem en classe plus petites et générales.

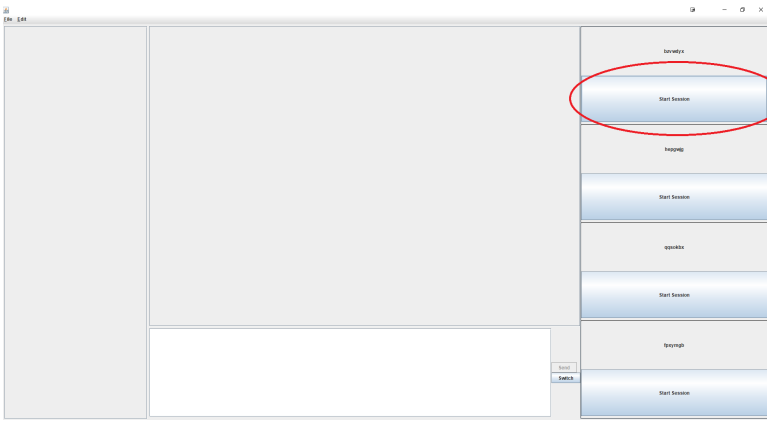

FIGURE 7 –

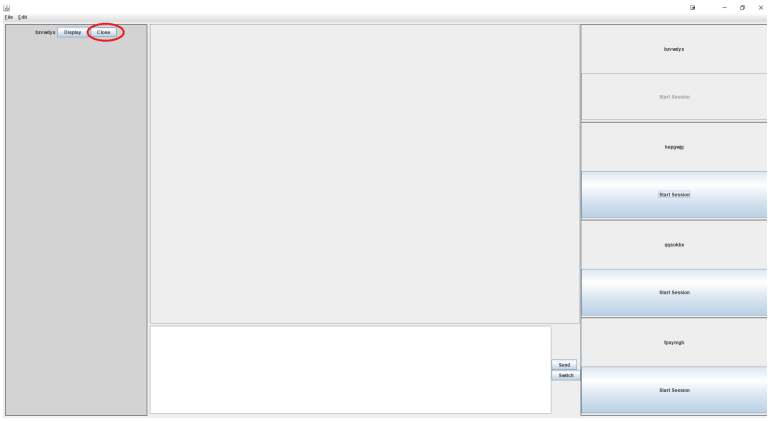

FIGURE 8 –

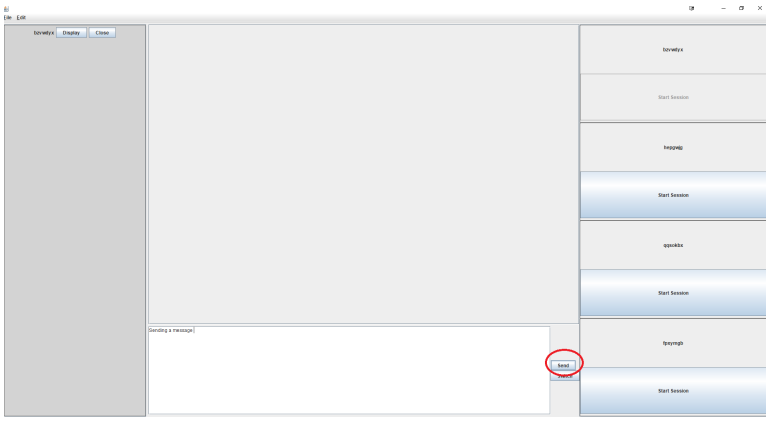

FIGURE 9 –

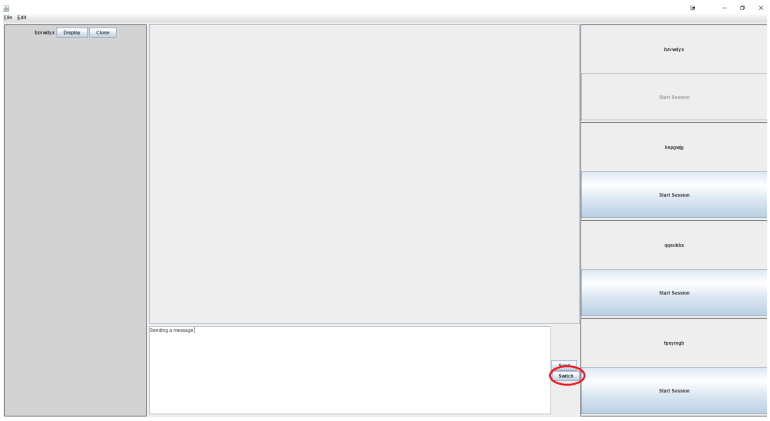

FIGURE 10 –

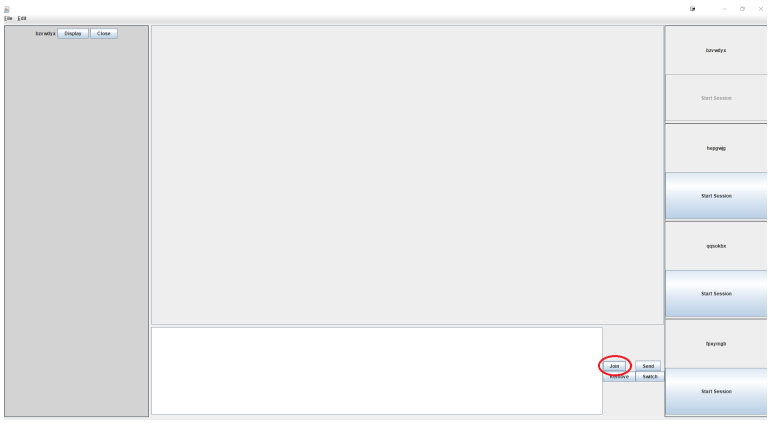

FIGURE 11 –

FIGURE 12 –