

Projet JMessenger

Alexandre Thiroux, Franciszek Dobrowolski

16 mars 2019

Table des matières

1	Conception	2
1.1	Répartition	2
1.2	Développement	2
1.3	Devops	2
2	Architecture du projet	3
2.1	Serveur	3
2.1.1	Technologies utilisées	3
2.1.2	Structure des packages	4
2.1.3	Diagramme des classes	4
2.2	Client	5
2.2.1	Technologies utilisées	5
2.2.2	Structure des packages	5
2.2.3	Diagramme des classes	5
3	Fonctionnement	6
3.1	Serveur	6
3.1.1	Fonctionnement général	6
3.1.2	Tests	8
3.1.3	Fonctionnalités	8
3.2	Client	10
3.2.1	Enregistrement	10
3.2.2	Connexion	11
3.2.3	Envoi de message	12
3.2.4	Les salons de discussion	13
3.2.5	Fonctionnalités administrateur	14
3.2.6	Les favoris	14
3.2.7	Notifications	14
3.2.8	Déconnexion	15
3.3	Tests	15
4	Déploiement	16
4.1	Déploiement du serveur	16
4.1.1	Déploiement local	16
4.1.2	Déploiement docker	16
4.2	Déploiement du client	17
5	Annexes	18

Chapitre 1

Conception

1.1 Répartition

La répartition des rôles au sein du projet s'est effectuée selon un découpage serveur/client. De fait, la répartition suivante a été adoptée :

- Alexandre Thiroux - Conception de l'interface graphique, du client et du cache local
- Franciszek Dobrowolski - Conception du serveur (API REST et serveur push), de la base de données distante

1.2 Développement

Le développement du client et du serveur a été effectué à l'aide de l'IDE IntelliJ (JetBrains). Ce dernier a été choisi pour sa facilité d'utilisation et sa souplesse face à ses principaux concurrents tel que Eclipse. Pour la création des interfaces graphique nous avons utilisé Scene Builder. Ce logiciel permet de concevoir des interfaces en sélectionnant et en ajoutant des composants de manière visuelle.

1.3 Devops

Nous avons utilisé le gestionnaire de versionning Gitlab. Celui-ci était hébergé sur un serveur privé nous appartenant. Gitlab nous a permis de travailler en commun, et de produire des versions fonctionnelles de notre code en implémentant partie par partie les différentes classes et composants.

Chapitre 2

Architecture du projet

Le projet est divisé en deux exécutables distincts, le client et le serveur. Le serveur centralise les données de tous les utilisateurs, garantit le respect des contraintes de gestion et assure la distribution des messages. Le client permet aux utilisateurs de communiquer avec le serveur à travers une interface graphique afin d'envoyer et recevoir des messages, consulter des statistiques ou parcourir les salons. Le système d'échanges repose sur le protocole HTTP à travers une API REST. Une deuxième source destinée aux notifications en temps réel est mise en place grâce aux java sockets.

Le projet a été réalisé en JAVA en version JDK 8.

2.1 Serveur

2.1.1 Technologies utilisées

Dans la réalisation du projet nous nous sommes engagés à utiliser le moins de dépendances possible. Ainsi, dans la mesure du raisonnable, le serveur est réalisé en utilisant uniquement les outils disponibles dans la distribution de développement Java.

HttpServeur

L'échange des données à travers l'API est réalisée grâce au `HttpServer`, un module embarqué dans le JDK. Cette classe permet de définir des contextes menant vers d'autres classes chargées de traiter la requête. Elle assure la bonne transmission des paquets sur internet et l'identification des composants d'une requête pour le traitement.

Java sockets

Les notifications push, nécessitent une communication bidirectionnelle. Pour cela les sockets Java nous ont paru être la solution la plus adaptée. Ils permettent d'établir des flux d'entrée et de sortie, pouvant être écrits et lus en temps réel, dans les deux sens.

PostgreSQL

Pour stocker les messages des utilisateurs et d'autres informations nécessaires pour le fonctionnement du chat nous avons choisi d'utiliser une base de données de type PostgreSQL.

Bcrypt

Afin de garantir la sécurité de conservation des mots de passe, nous avons mis en place un chiffrement utilisant l'algorithme Bcrypt, qui garantit une très haute sécurité.

Jackson

Le mappage du corps des requêtes est réalisé par la bibliothèque Jackson de `fasterxml`.

Maven

Afin de gérer les dépendances nécessaires pour la JDBC, le Bcrypt, la bibliothèque Jackson, ainsi que le packaging nous avons utilisé Maven 3.

2.1.2 Structure des packages

Les classes sont organisées selon le type de fonctionnalité qu'elles réalisent. La séparation choisie a pour but d'isoler les tâches, pour qu'une classe n'ait pas à traiter l'intégralité d'une requête. De telle façon on peut modifier un comportement sans affecter la chaîne entière. C'est une façon de garantir une isolation technique entre les différentes parties du programme.

Système

Dans la package système on trouve les classes partagés, responsables du bon fonctionnement du programme intégral. On y trouve le log, la configuration, et les outils comme le générateur de token.

Notifications push

Contient les classes responsables du service des notifications.

Exceptions

Contient les exceptions personnalisées pour le serveur. Notamment la `ProcessingException` permettant le retour verbeux lors d'une requête que ne peut pas être complété.

Entités

Les entités sont les objets personnalisés dont le serveur va manipuler. Dans le sous-package `api` se trouvent les classes prévues à être mappées pour l'interface `api`. Le sous-package `enums` contient les énumérations comme les types d'erreur personnalisés pour le programme.

Handlers

Les handlers sont les classes responsables de l'extraction des données des requêtes, l'envoi des ordres de notification, la construction des réponses `http` et le choix de service amené à traiter la requête. Elles ne se soucient pas du traitement de la demande, mais assurent les bonnes transformations et le bon routage des demandes. Ils vérifient la conformité technique des données. Il existe un handler par type d'entité attendu en entrée.

Services

Le rôle des services est d'apporter l'intelligence métier sans se soucier de l'échange `http`. Ils s'occupent à traiter les données fournies par les handlers. Ils construisent les éléments de réponse à partir des données externes et internes, sans se soucier de comment ces données sont fournies. Ils vérifient également la conformité fonctionnelle des données. Il existe un service par handler, plus des services supplémentaires.

Dépôts (repositories)

Ces classes sont destinées à interagir avec la base des données. Elles construisent les requêtes pour répondre aux demandes des services, sans se soucier de leur utilisation. Elles garantissent que les bonnes données sont fournies aux services. Elles sécurisent également les requêtes contre l'attaque `SQL injection`. Il existe un dépôt par table dans la base des données. En vu du caractère relationnel de la donnée manipulée, il se peut qu'un dépôt fasse des requêtes jointes sur plusieurs tables.

2.1.3 Diagramme des classes

Comme nous avons pu voir dans structure des packages il existe des groupes de classes ayant des fonctionnalités communes avec des parties spécifiques. Dans ce cas une architecture d'héritage a pu être mise en place pour rassembler les fonctionnalités communes aux classes du même package pour n'implémenter les méthodes qu'une génériques et spécifiques qu'une seule fois.

Les diagrammes des classes par package se trouvent dans les annexes.

- Le package des entités `api` - Figure 5.1
- Le package des énumérations - Figure 5.2
- L'entité utilisateur - Figure 5.3
- Le package des exceptions - Figure 5.4
- Le package des handlers - Figure 5.5
- Le package du service push - Figure 5.6

- Le package des dépôts - Figure 5.7
- Le package des services - Figure 5.8
- Le package système - Figure 5.9

2.2 Client

2.2.1 Technologies utilisées

FasterXML

Permet la sérialisation et la désérialisation des chaînes de caractère vers ou depuis le format JSON. Le JSON est utilisé par les communications entre le serveur et le client.

JavaFX

JavaFX est une librairie Java permettant la création d'interfaces graphique cliente grâce à un ensemble de composants.

Socket

JMessenger utilise les sockets afin de dialoguer avec le serveur Push.

Maven

Utile pour l'installation des librairies, la gestion des dépendances ou encore la phase de déploiement, Maven est un outil de gestion et d'automatisation de production des projets logiciels Java.

JDBC

La librairie JDBC permet l'utilisation de base de données et du langage SQL afin d'exécuter des requêtes.

2.2.2 Structure des packages

Le client utilise est construit selon le modèle MVC (Modèle / Vue / Contrôleur).

La partie Modèle d'une architecture MVC encapsule la logique métier ainsi que l'accès aux données. Il s'agit ici d'un modèle orienté objet.

La partie Vue s'occupe des interactions avec l'utilisateur : présentation, saisie et validation des données. C'est ici qu'intervient JavaFX.

La partie Contrôleur gère la dynamique de l'application. Elle fait le lien entre l'utilisateur et le reste de l'application. Les classes présentes dans le contrôleur permettent d'instancier et d'accéder aux objets du modèle.

2.2.3 Diagramme des classes

Étant donné l'utilisation du format MVC, il n'y a que trois différents packages présent dans le logiciel. Les diagrammes des classes par package se trouvent dans les annexes.

- Le package des contrôleurs - Figure 5.10
- Le package des modèles - Figure 5.11
- Le package des vues - Figure 5.12

Chapitre 3

Fonctionnement

Comme vu précédemment, le projet est constitué de deux entités distinctes. Cela rend possible le fonctionnement du serveur et du client sur des systèmes séparés. Le serveur est autonome, tandis que le client requiert un serveur pour fonctionner.

3.1 Serveur

3.1.1 Fonctionnement général

Service API

Le fonctionnement général peut-être décrit avec le schéma si-dessous :

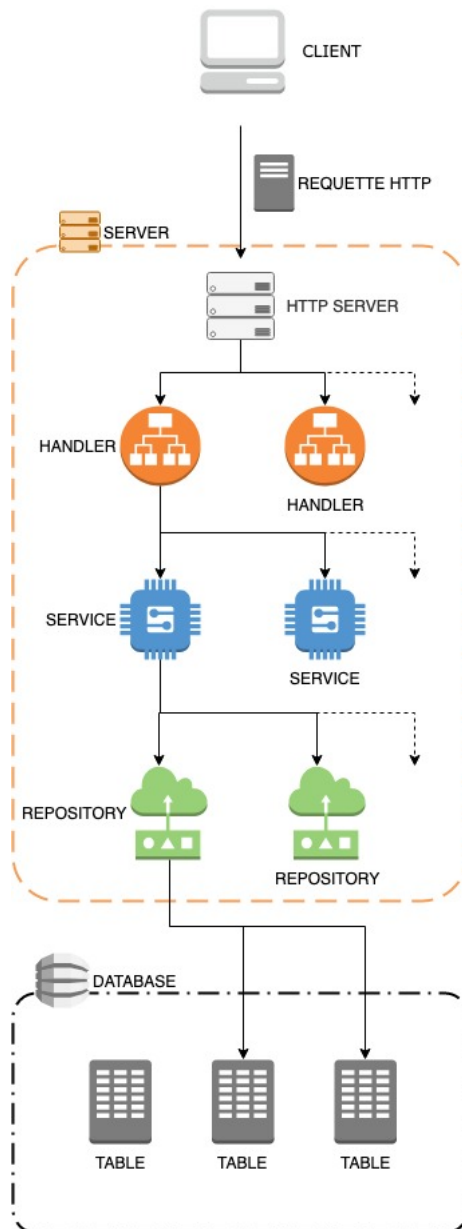


FIGURE 3.1 – Schéma de fonctionnement du serveur

1. Le serveur http réceptionne une requête, la transforme en objet Java et appelle le handler correspondant
2. Le handleur mappe la donnée, et décide lequel service va la traiter
3. Le service traite la demande et demande des données au dépôt
4. Le dépôt demande les information à la base des données
5. La base des données fournit les informations
6. Les informations et le résultat de la requête remontent la chaîne jusqu'au handler, qui ordonne une notification au serveur push pour les notifications et au serveur pour un retour de requête.

Service push

Le service des notification est basé sur un processus, qui écoute des nouvelles connexions et crée des processus à chaque nouveau utilisateur connecté. Lors d'un ordre de notification, le service push reçoit la liste des utilisateurs à notifier, et transmet la notification aux processus leur correspondants. Ces derniers se chargent de transmettre les données dans les java sockets.

3.1.2 Tests

Pour tester l'api nous avons utilisés l'application Postman, qui permet de créer des requêtes http et observer le résultat dans une interface graphique. Le déploiement a été testé sur un serveur dédié privé, équipé d'un client Docker.

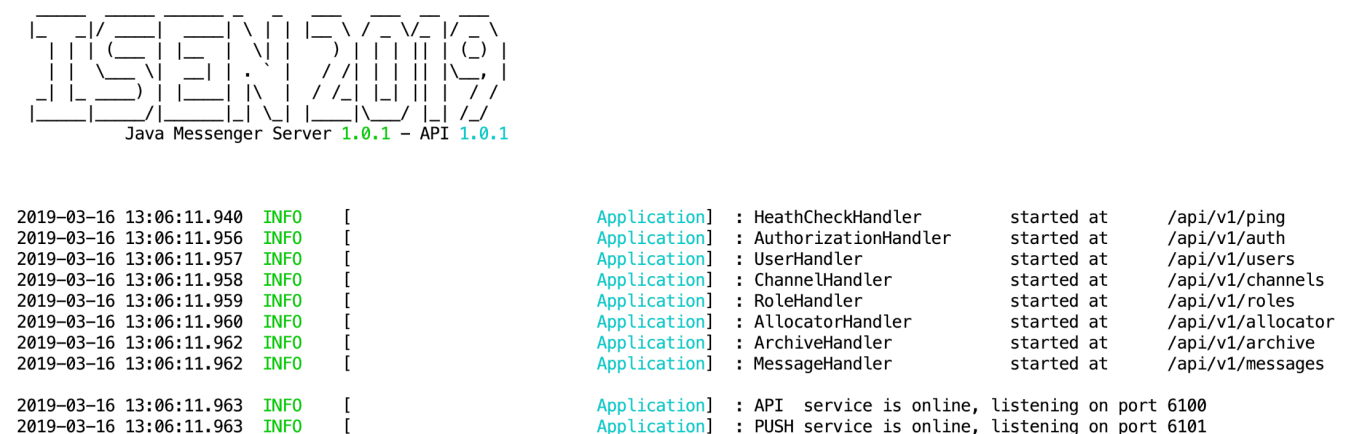
3.1.3 Fonctionnalités

La fonctionnalité principale est de garantir l'accès aux données à travers une interface web de type API. En plus de cela le serveur possède des fonctionnalités supplémentaires, pour offrir un meilleur diagnostic en cas d'erreur, l'authentification, et une installation facile.

La documentation détaillée de chacun des noeuds de l'API, des exigences, ainsi que des exemples de requêtes et de réponses sont disponibles dans l'annexe.

Le gestionnaire de logs

La première fonctionnalité pour un bon développement est le gestionnaire des logs. Pour le serveur nous nous sommes inspirés du log4j afin de réaliser un fonctionnement et un rendu similaire. Le gestionnaire permet de contrôler globalement l'affichage des messages dans la console, ainsi que des niveaux de log, paramétrable pour ne garder que des information nécessaires à un moment donnée. Il permet aussi d'identifier le temps exacte et la classe dont le message provient. Ci-dessous un exemple du rendu final, au lancement du serveur :



```
Java Messenger Server 1.0.1 - API 1.0.1

2019-03-16 13:06:11.940 INFO [Application] : HeathCheckHandler started at /api/v1/ping
2019-03-16 13:06:11.956 INFO [Application] : AuthorizationHandler started at /api/v1/auth
2019-03-16 13:06:11.957 INFO [Application] : UserHandler started at /api/v1/users
2019-03-16 13:06:11.958 INFO [Application] : ChannelHandler started at /api/v1/channels
2019-03-16 13:06:11.959 INFO [Application] : RoleHandler started at /api/v1/roles
2019-03-16 13:06:11.960 INFO [Application] : AllocatorHandler started at /api/v1/allocator
2019-03-16 13:06:11.962 INFO [Application] : ArchiveHandler started at /api/v1/archive
2019-03-16 13:06:11.962 INFO [Application] : MessageHandler started at /api/v1/messages

2019-03-16 13:06:11.963 INFO [Application] : API service is online, listening on port 6100
2019-03-16 13:06:11.963 INFO [Application] : PUSH service is online, listening on port 6101
```

FIGURE 3.2 – Démarrage du gestionnaire des logs

Configurateur externalisé

En suite, pour permettre paramétrage sans édition de code, un configurateur externalisé a été mis en place. Le fichier app_config.json situé dans le dossier ressources permet de changer les options de du log, les accès à la base de données et les limites fonctionnelles, sans intervention des les classes java.

```

{
  "system_log_level": "INFO",
  "server_api_port": "6100",
  "server_push_port": "6101",
  "server_api_path": "/api/v1",
  "sql_host": "localhost",
  "sql_port": "5432",
  "sql_database": "postgres",
  "sql_schema": "jmsg",
  "sql_user": "postgres",
  "sql_password": "password",
  "default_channel_uid": "f2f62cc7-da3c-49b0-b7aa-79f7001c6afc",
  "auth_attempt_limit": "3",
  "auth_token_validity_hours": "5",
  "auth_ip_block_duration_minutes": "15",
  "auth_simultaneous_connection_limit": "5",
  "message_max_length": "500"
}

```

FIGURE 3.3 – Fichier de configuration serveur

Installateur de base des données

Le serveur dispose d'un installateur de schéma de base des données automatisé. Pour une première installation du serveur il suffit de spécifier les informations d'accès vers un schéma postgres vide avec des droits d'éditations suffisants. Le serveur créera des tables nécessaires avant de lancer le processus principal. De même avant chaque lancement du serveur la présence de la structure nécessaire est vérifiée. Les requêtes de création des tables sont chargées à partir d'un script sql. Nota bene le fonctionnement de ce module peut être observé en niveau DEBUG du log.

Authentification sécurisée

Grâce au stockage des données le serveur est capable d'offrir une authentification aux utilisateurs, pour une différenciation des accès aux ressources et actions spécifiques. Un token d'autorisation est fourni à l'utilisateur en échange d'une combinaison de login et de mot de passe correspondant à un utilisateur existant dans la base. Il existe un système qui empêche un utilisateur d'entrer un mauvais mot de passe trop de fois à la suite. Si la limite paramétrée dans la configuration est dépassée, alors l'IP de l'utilisateur est bloquée pour une durée déterminée, elle aussi paramétrable. Le token d'autorisation a une validité paramétrable. À chaque accès à l'api la validité est étendue. Si la date de validité d'un token est dépassée il est considéré comme étant invalide et est supprimé de la base des données. Le token permet au serveur de lier une requête à un utilisateur, et par conséquent d'accéder aux droits et données liées à son compte.

Retour verbeux

Lors des tests sur une API il arrive souvent que pour une raison la requête ne peut pas être complétée. Un champ manquant ou une donnée erronée peuvent empêcher le serveur de réaliser la demande. Il est très utile que le serveur précise au client la raison de l'échec de traitement pour que celui-ci puisse facilement corriger le problème. Afin de permettre un tel retour, nous avons mis en place un système d'exceptions api personnalisé. Avec des codes d'erreur et des messages, le serveur est capable d'informer l'utilisateur du problème rencontré.

Validation des requêtes

Comme l'api dans l'idée sera publiquement disponible, alors elle possède des multiples points de contrôle pour assurer un usage conforme à l'usage prévu. Les requêtes des utilisateurs sont vérifiées pour la méthode utilisée, le chemin appelé, le contenu transmis, l'autorisation requise et les droits d'effectuer une action.

Notifications

Lors de la réalisation du projet nous étions confrontés à la problématique d'envoi des notifications aux utilisateurs. Lors d'un nouveau message, d'un changement de salon et d'autres actions l'utilisateur doit être informé du changement. La solution consistant à redemander les données au serveur périodiquement nous a pas paru non adaptée de point de vue d'optimisation. Pour palier à ce problème nous avons décidé de créer un deuxième service parallèle, qui s'occupera à envoyer des notifications push. L'utilisateur peut s'enregistrer sur le serveur push avec son token d'autorisation, et les événements serveur le concernant lui seront transmis dans le temps réel grâce à une connexion bidirectionnelle.

Multi processus

Nous avons du prévoir aussi un système de multi-threading dans le cas ou plusieurs requêtes arrivent au serveur au même temps. Pour cela chaque requête api est traitée dans un nouveau thread, grâce à la fonctionnalité implémentée dans le `HttpServer`. Pour les requêtes push, le serveur gère les connexions lui-même en administrant les utilisateurs dans des processus séparés.

Déploiement Docker

En fin, un déploiement peut être facilement réalisé grâce à la technologie de Docker. Le serveur peut être facilement dockerisé et déployé sur une structure distante.

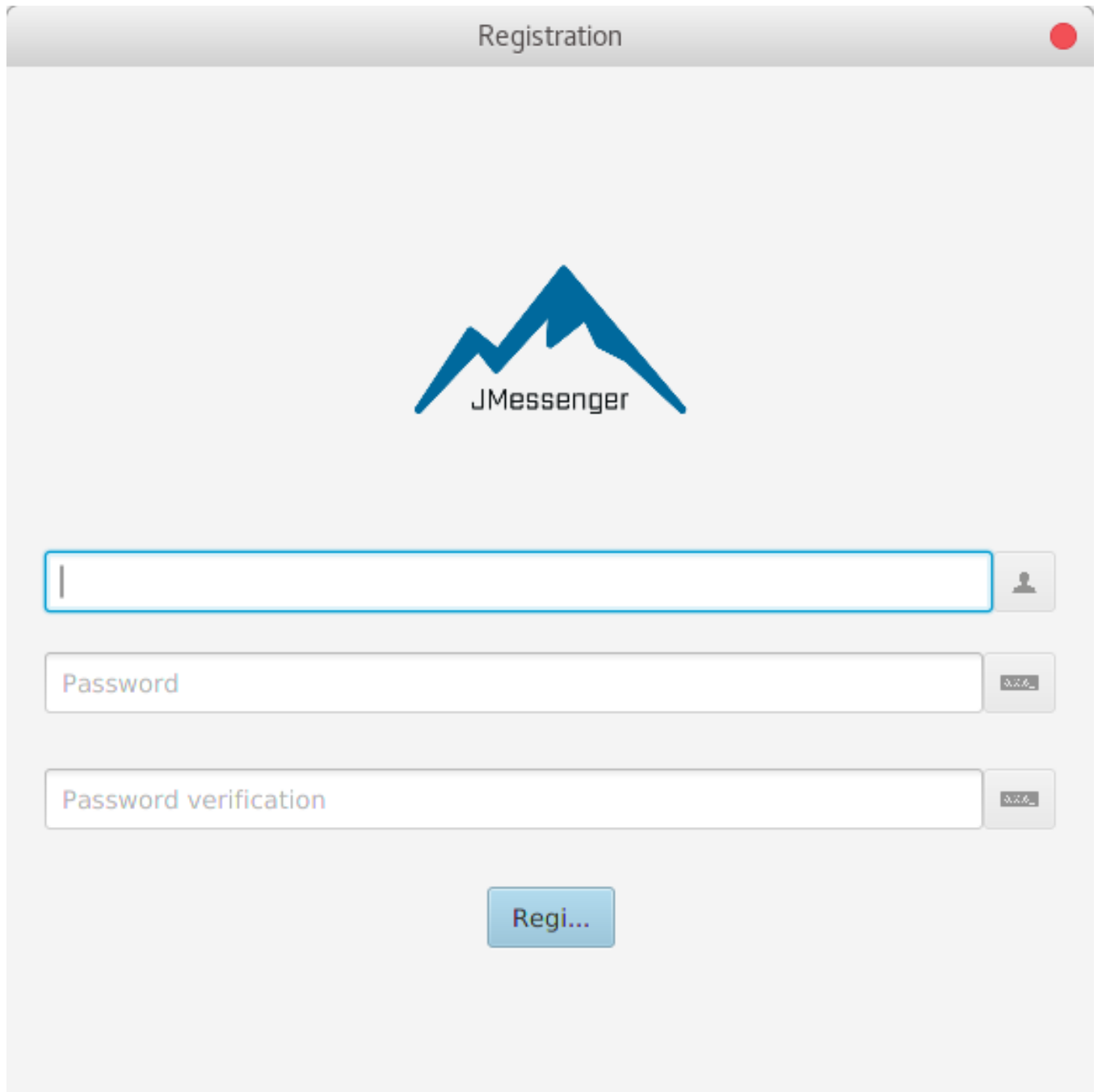
3.2 Client

Le client communique au travers d'une interface graphique avec l'API REST située sur le serveur. Il permet à un utilisateur de se connecter afin de pouvoir utiliser les services proposés par le logiciel. Ainsi, il sera en mesure de rejoindre les salons qui l'intéressent et de dialoguer avec d'autres personnes.

Une base de données locale réplique les informations reçues depuis le serveur. Cela évite l'accumulation de requêtes vers l'API.

3.2.1 Enregistrement

Pour pouvoir utiliser l'application, il est nécessaire de s'enregistrer. Pour cela il faut cliquer sur "Register" présent dans l'interface de connexion. L'enregistrement consiste à fournir un nom d'utilisateur ainsi qu'un mot de passe. Une vérification du mot de passe est effectuée.



The image shows a registration window titled "Registration" with a red close button in the top right corner. In the center, there is a blue mountain logo with the text "JMessenger" below it. Below the logo, there are three input fields: a username field with a person icon on the right, a password field with a "Show/Hide" icon on the right, and a password verification field with a "Show/Hide" icon on the right. At the bottom center, there is a blue button labeled "Regi...".

FIGURE 3.4 – Inscription

Après vérification, si le nom d'utilisateur n'est pas déjà utilisé par un autre utilisateur, l'enregistrement est effectué.

3.2.2 Connexion

Pour se connecter, l'utilisateur doit entrer son couple d'identifiants et appuyer sur le bouton "Login". Si les identifiants sont corrects et que le compte n'a pas été désactivé, le client est authentifié et l'interface principale apparaît. À sa première connexion, le client se trouve dans le salon de discussion par défaut, le salon "Général".

À chaque changement de compte, les données présentes dans la base locale sont effacées, empêchant le nouvel utilisateur d'avoir accès à des informations provenant d'un autre utilisateur.

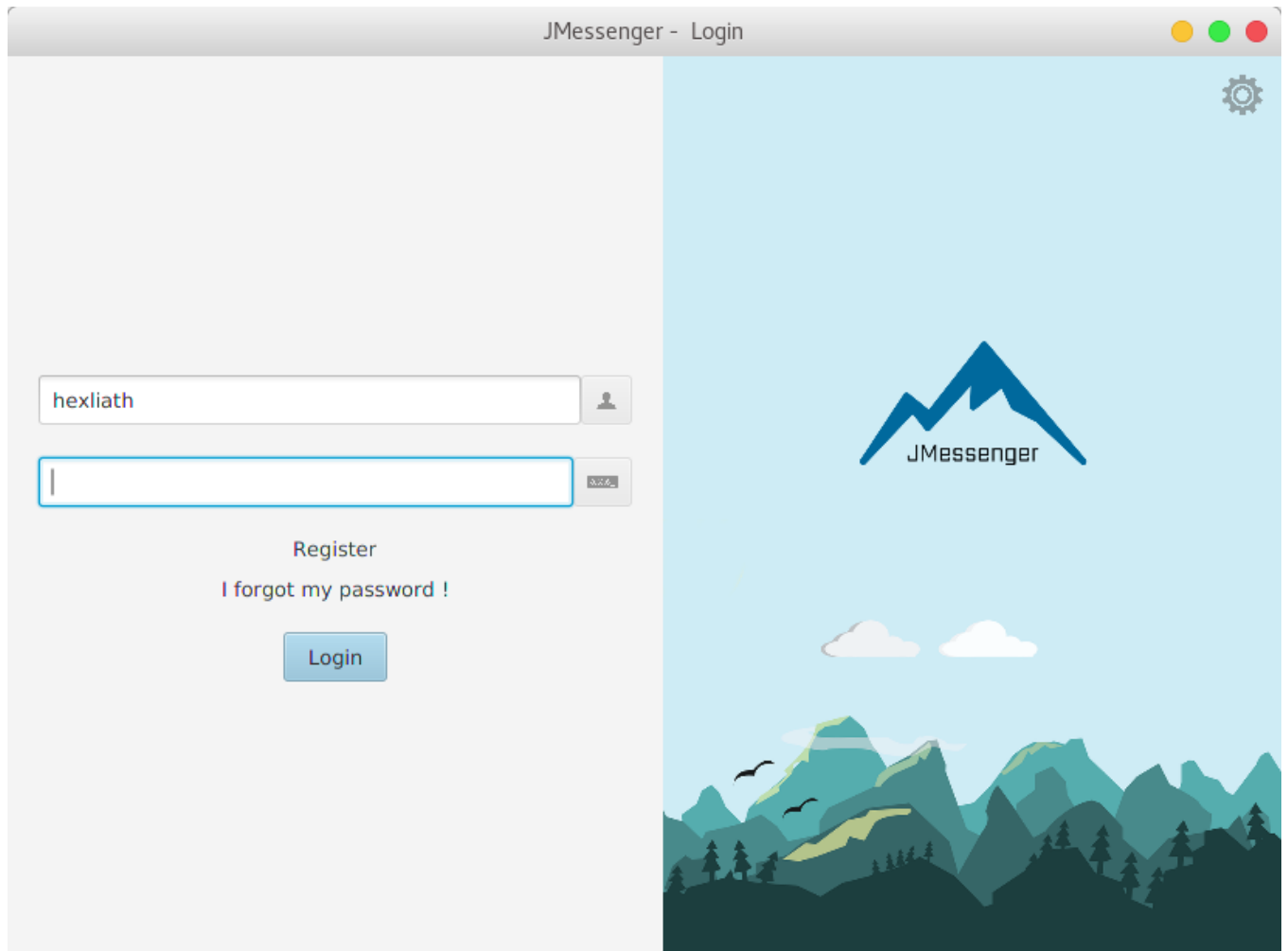


FIGURE 3.5 – Interface de connexion

3.2.3 Envoi de message

Pour envoyer un message, il est nécessaire d'avoir rejoint le salon dans lequel l'utilisateur se situe. Si c'est le cas, un message peut être envoyé en remplissant le champ de texte situé en bas de l'interface et en cliquant sur le bouton "Envoyer" représenté par l'icône associée.

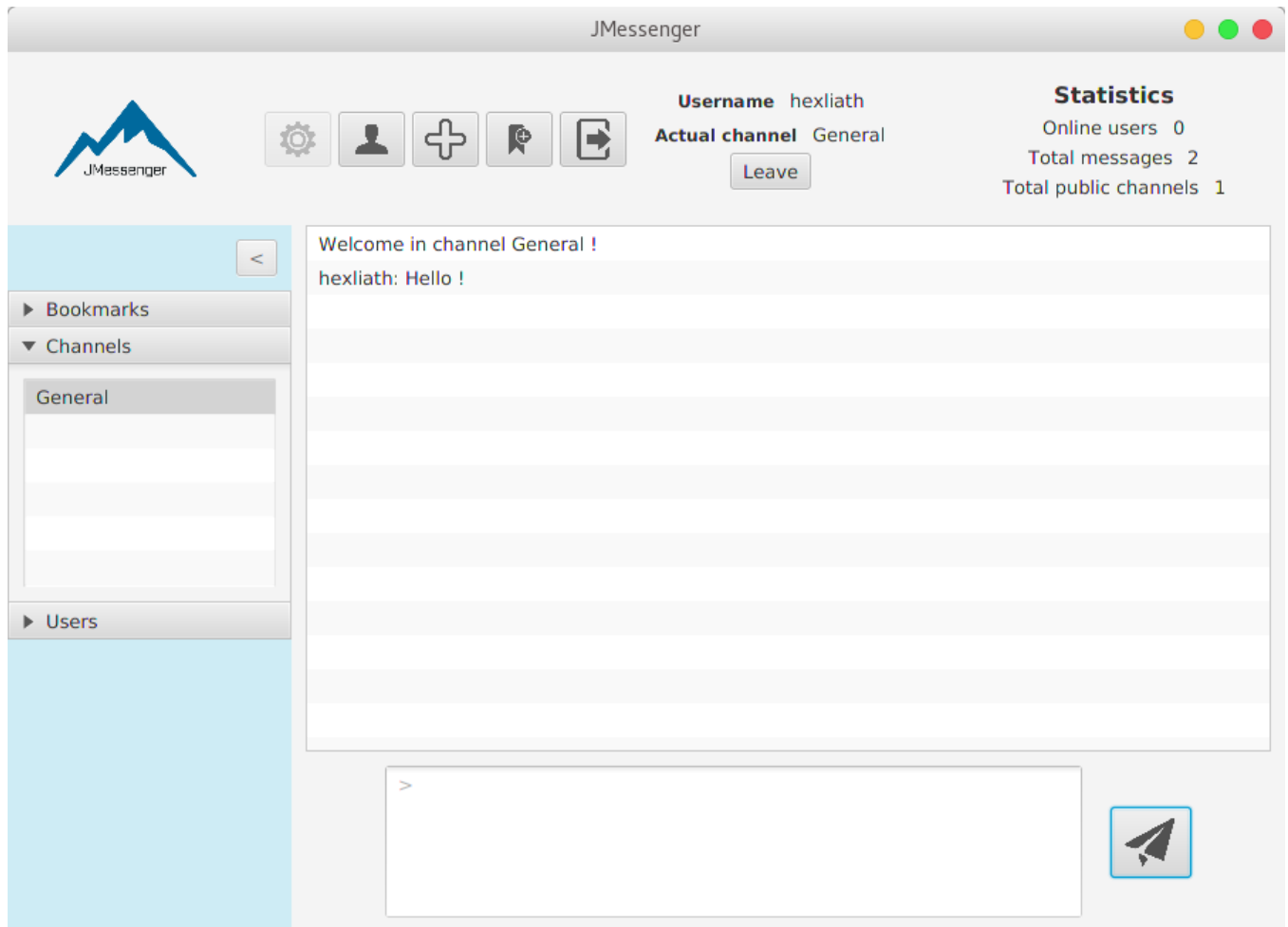


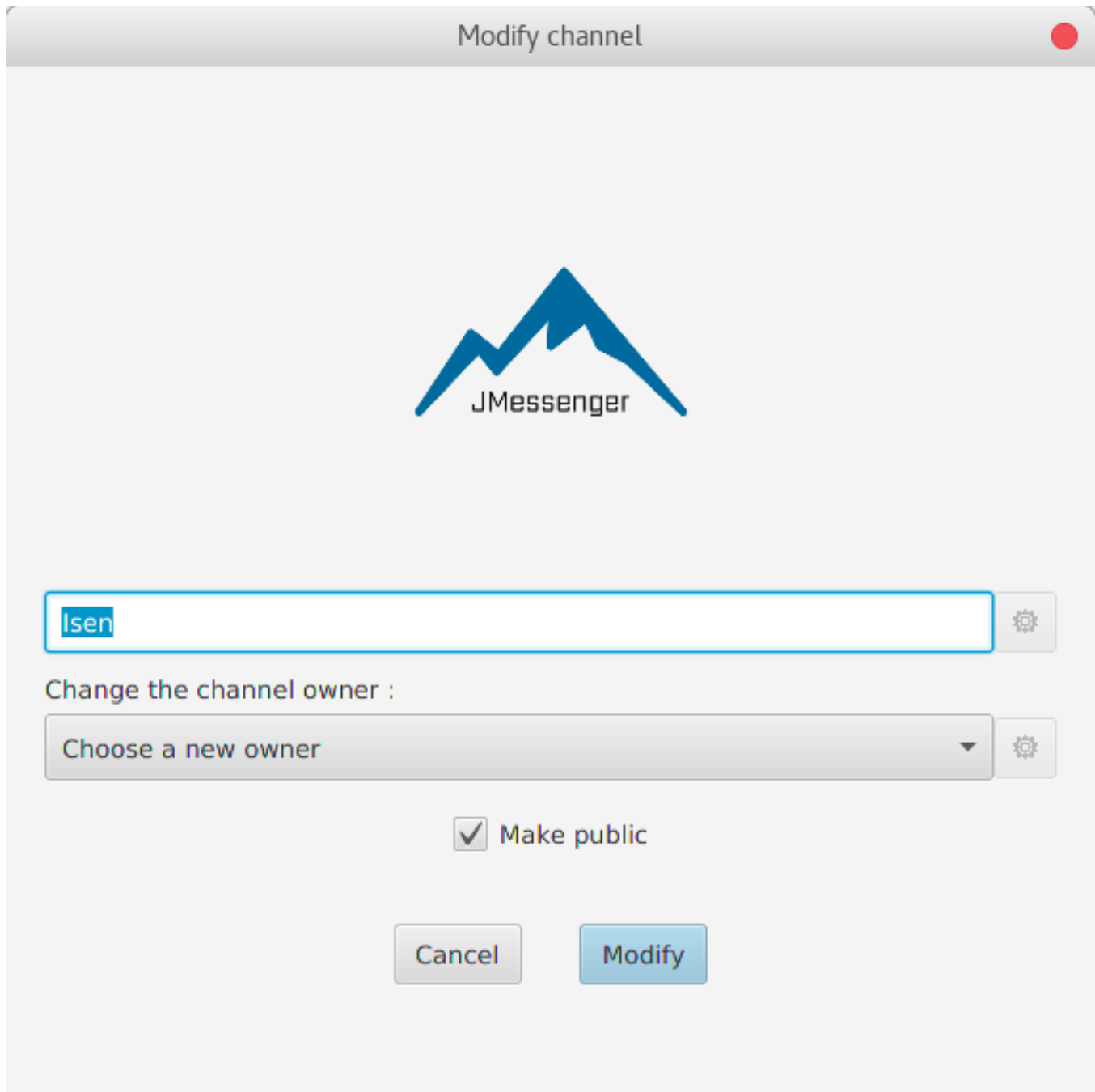
FIGURE 3.6 – Interface principale

3.2.4 Les salons de discussion

Les salons de discussion permettent aux utilisateur de discuter dans un espace ayant un thème dédié. Les salons peuvent être de trois types différents : Publique, Groupe, ou Privé.

Le premier est un espace que chacun peut rejoindre à sa convenance, tandis que le second nécessite d'y être invité. Le dernier quand à lui est une discussion privée entre deux utilisateurs. Chaque membre peut créer un nouveau salon en cliquant sur le bouton en forme de "+". Le créateur du salon peut désigner des Administrateurs et modifier le nom du salon.

Pour être en mesure de quitter un salon, son propriétaire doit transmettre ses droits à un autre utilisateur.



Modify channel

JMessenger

Isen

Change the channel owner :

Choose a new owner

☒ Make public

Cancel Modify

FIGURE 3.7 – Modification d'un salon

3.2.5 Fonctionnalités administrateur

Export XML

L'export en XML est disponible en cliquant sur le bouton associé. Il faut pour cela être administrateur du salon ou le créateur.

3.2.6 Les favoris

Il est possible d'ajouter un channel à ses favoris en cliquant sur le bouton "Favoris". L'ajout d'un salon à ses favoris permet de le retrouver plus facilement.

3.2.7 Notifications

En tâche de fond, un socket est en permanence connecté au serveur dans le but d'être informé en temps réel d'une notification (Utilisateur connecté, déconnecté, un nouveau message dans notre salon, création d'un nouveau salon

public). De cette manière, l'interface utilisateur est mise à jour en fonction du type de notification reçue.

3.2.8 Déconnexion

À la déconnexion, le salon actuel est enregistré dans la configuration. De ce fait, lors de la prochaine connexion, l'utilisateur retrouvera le dernier salon auquel il était connecté.

3.3 Tests

Lors du développement, il était alors impossible de tester le comportement du serveur de manière pragmatique, celui-ci n'étant pas encore au point. Nous avons utilisé pour tester le client le logiciel "Mockoon" permettant de simuler une interface REST. Grâce à la documentation de l'API établie en amont, il a été facile de simuler le résultat de requêtes en provenance du client et d'en tester les réactions.

Chapitre 4

Déploiement

4.1 Déploiement du serveur

4.1.1 Déploiement local

Base des données

Il est possible d'utiliser une base des données en local ou démarrer une base dans un container Docker

```
docker run --rm --name postgres -e POSTGRES_PASSWORD=password -d -p 5432:5432 postgres
```

Créer un schéma vide à l'usage du serveur

Configuration

Remplir le fichier json de configuration du serveur en y renseignant les données de connexion vers la base des données.

```
"sql_host": "localhost",  
"sql_port": "5432",  
"sql_database": "postgres",  
"sql_schema": "jmsg",  
"sql_user": "postgres",  
"sql_password": "password",
```

FIGURE 4.1 – Extrait du fichier de configuration serveur - accès base des données

Compilation

Créer un jar exécutable avec maven

```
mvn package
```

Lancement

Lancer le serveur en ligne de commande, en utilisant le jar avec dépendences

```
java -jar messenger-server-1.0.1-jar-with-dependencies.jar
```

4.1.2 Déploiement docker

Création de l'image

Ajouter un Dockerfile dans le fichier contenant le jar.

```
FROM java:8  
WORKDIR /  
ADD messenger-server-1.0.1-jar-with-dependencies.jar messenger-server-1.0.1-jar-with-dependencies.jar  
EXPOSE 6100  
EXPOSE 6101  
CMD ["java", "-jar", "messenger-server-1.0.1-jar-with-dependencies.jar"]
```

Lancement

Lancer l'image docker en arrière plan ou avec la sortie de console

```
docker run -d -p 6100:6100 -p 6101:6101 jmessenger
docker run -i -t -p 6100:6100 -p 6101:6101 jmessenger
```

4.2 Déploiement du client

Pour déployer le client, les commandes suivantes doivent être effectuées :

```
git pull https://github.com/Hexliath/JMessenger.git
apt-get install openjfx libswt-gtk-3-java
cd jmessenger/client
mvn package
java -jar target/jmessenger-front-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Il est aussi possible d'utiliser directement l'exécutable jar présent dans l'archive. Il faut pour cela utiliser la commande suivante :

```
java -jar target/jmessenger-client.jar
```

La configuration s'effectue lors la connexion. En utilisant l'icone réglages située en haut à droite. L'adresse de connexion comprenant la partie "http :/" doit être rentrée à l'endroit prévu.

Annexes

Agrandir la page pour voir plus de détails sur les images.

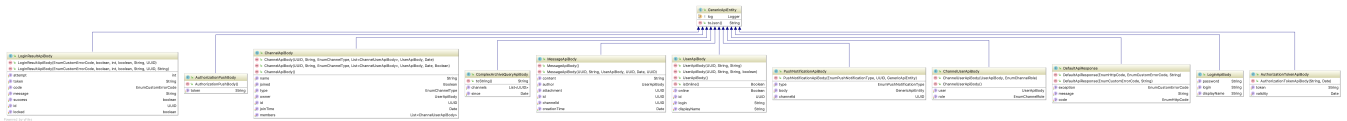


FIGURE 5.1 – Serveur - diagramme des classes - api entities package

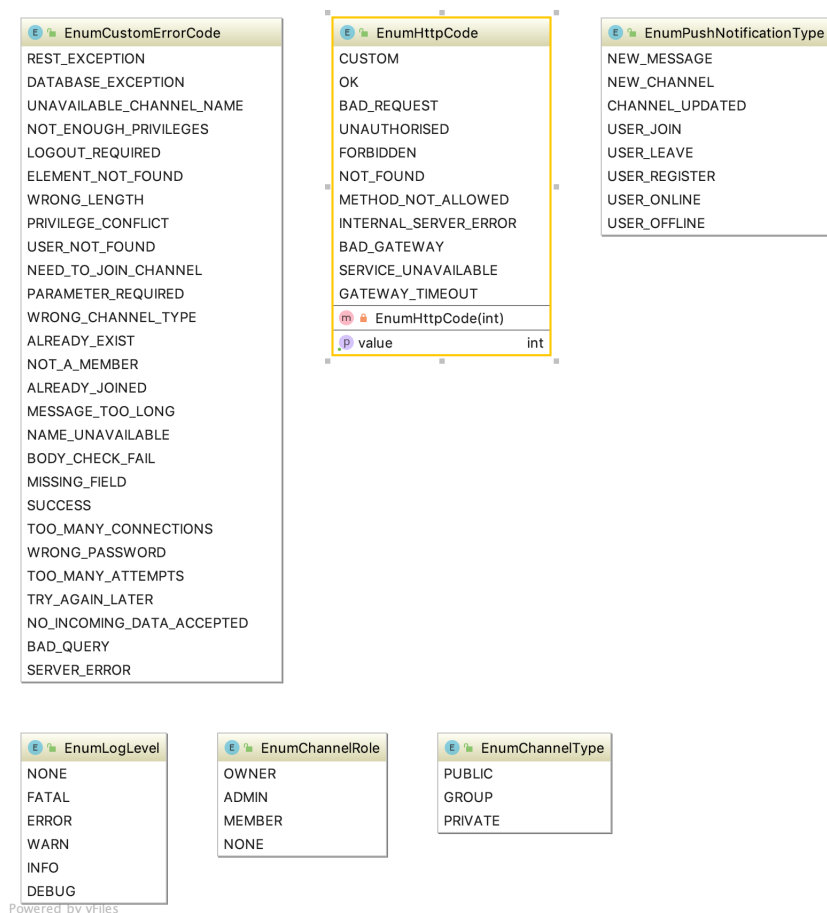


FIGURE 5.2 – Serveur - diagramme des classes - enums package

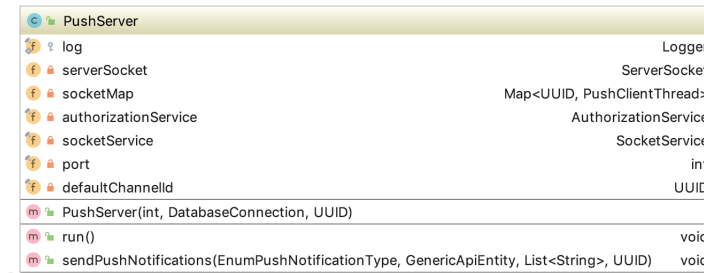
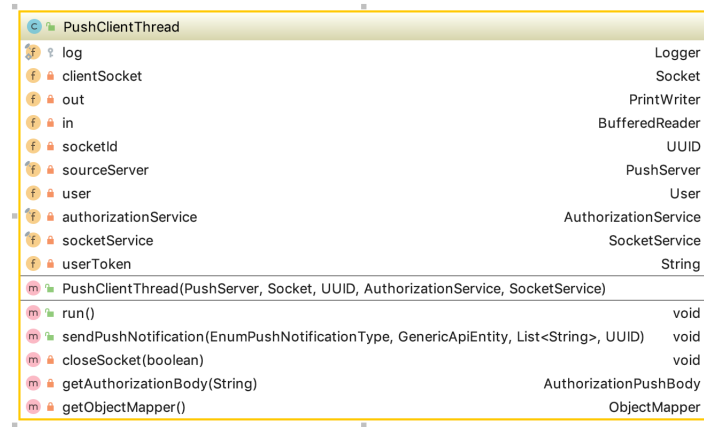


FIGURE 5.6 – Serveur - diagramme des classes - push service package

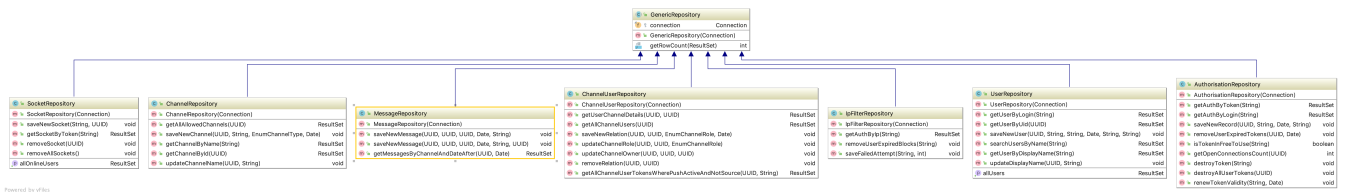


FIGURE 5.7 – Serveur - diagramme des classes - repositories package

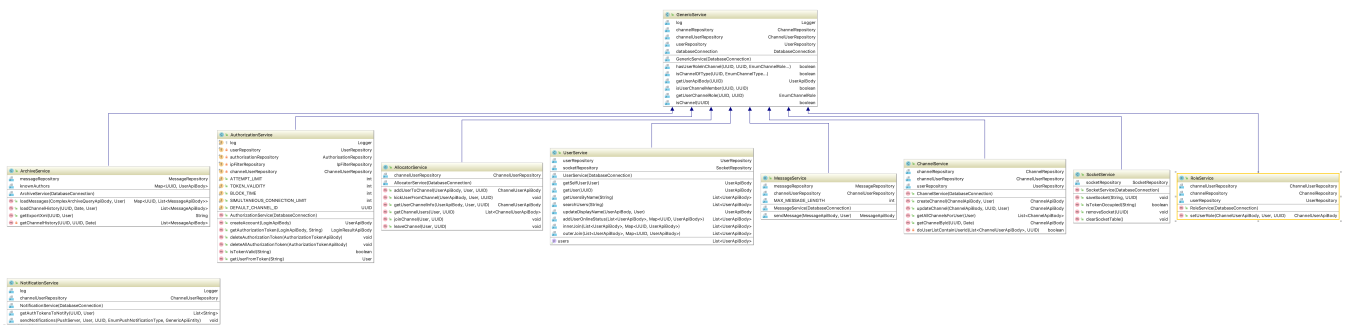


FIGURE 5.8 – Serveur - diagramme des classes - services package

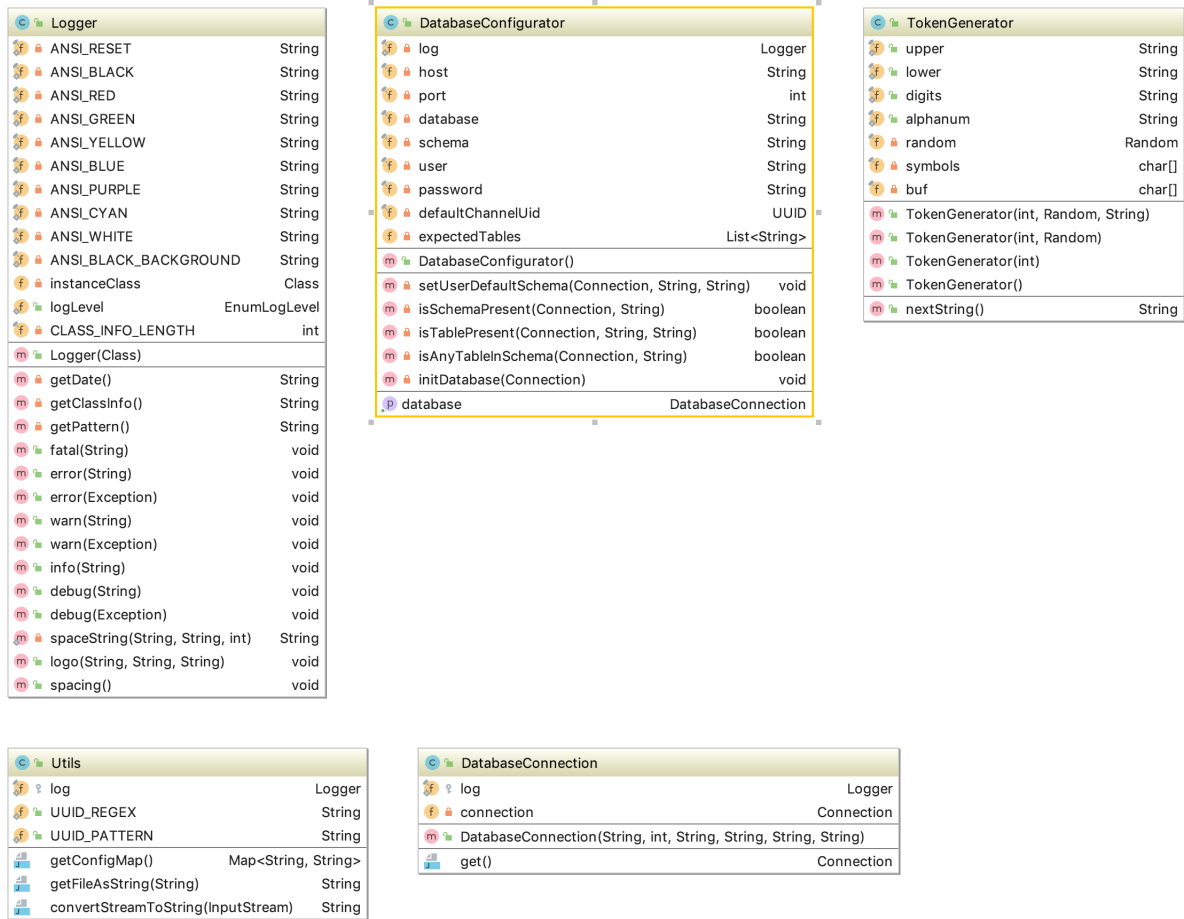


FIGURE 5.9 – Serveur - diagramme des classes - system package

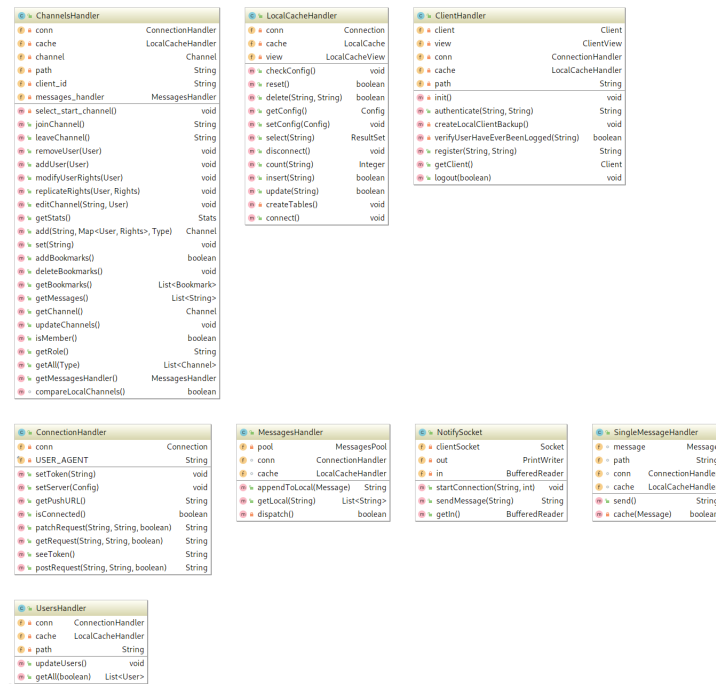


FIGURE 5.10 – Serveur - diagramme des classes - controller

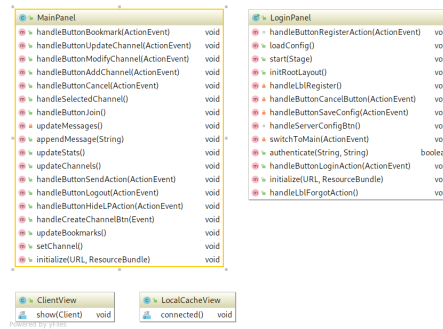


FIGURE 5.11 – Serveur - diagramme des classes - View

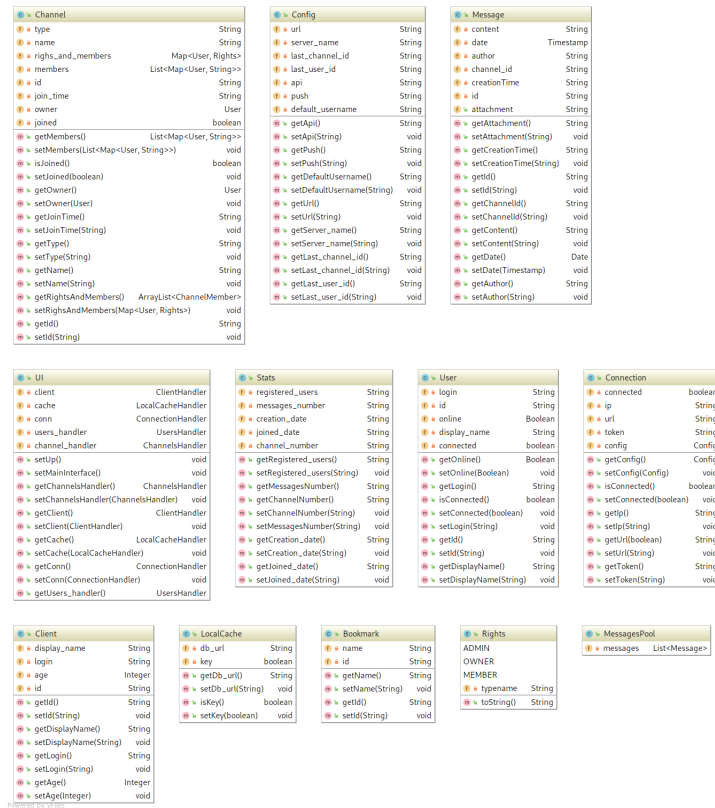


FIGURE 5.12 – Serveur - diagramme des classes - Model