DOCUMENTATION DÉVELOPPEUR

1. DESCRIPTION TECHNIQUE

Afin d'obtenir une architecture de projet structurée, nous avons séparé notre Application en trois modules principaux.

I. MODULES PRINCIPAUX & BASE DE DONNÉES

a. Client

Le premier module est le module ASAPP-Client et correspond à toutes les interactions du client à travers l'application qu'il aura installée. Ce module prend en considérations toute les actions effectuées par l'utilisateur à savoir le login, register, changement de langues, créations de règles (CFF/RTS/TWITTER/MÉTÉO), suppression de règles, modification de paramètres...
Une fois une action effectuée, une requête est interprétée par la classe ClientRequest puis envoyée au serveur qui va se charger d'exécuter cette requête afin de récupérer la réponse depuis la base de données.

Ce module correspond donc à ce qu'on appelle communément le "frontend".

b. Serveur

Le deuxième module est le module ASAPP-Server et correspond aux interprétations des actions effectuées par un utilisateur.
Il contient la définition des services, les différentes règles, le gestionnaire de notification via Telegram, le gestionnaire des tâches automatisées et bien évidemment le serveur, qui effectue la connexion avec la base de données.

Ce module correspond donc à ce qu'on appelle communément le "backend".

c. Common

Le troisième module est le module ASAPP-Common et comprend tous les éléments partagés par le module client et le module serveur. Essentiellement celui-ci contient tout le protocole de communication entre le client et le serveur, à savoir toutes les commandes effectuées par un utilisateur afin qu'elle soit interprétée par le serveur et qu'il exécute l'action correspondante aussi et afin d'éviter une duplication de code entre les deux modules principaux (serveur et Client) nous y avons ajouté les éléments suivants : les parseurs des JSON, et la cryptographie (simplement la fonction de hachage), les regex (pour contrôler les saisies de l'utilisateur ainsi que la politique de mot de passe).

d. Base de données

La base de données est composée d'une seule table contenant les champs suivants :

Champs	Descriptions
id	Identifiant de l'utilisateur
username	Nom de la personne sur notre application (nous sert à lier l'utilisateur à ses règles)
telegramUsername	Pseudo permettant d'associer la personne à son compte telegram (Ex : @Guy)
idTelegram	Entier permettant de lier une personne à son compte telegram
password	Password hashé et « salé » de la personne
rules	JSON stocké sous la forme d'une string contenant les différentes règles créées par un utilisateur.
langue	Langue choisie par l'utilisateur (Français ou anglais pour le moment)

La base de données est déployée sur le Cloud via le service Amazon Web Service (RDS). L'accès à la base de données est protégé par un mot de passe. Afin d'obtenir un droit d'accès en tant que développeur souhaitant améliorer notre application, merci de prendre contact avec notre Administrateur Réseau Adam Zouari qui vous fournira des credentials pour y accéder directement si vos motifs sont valables :adam.zouari@heig-vd.ch .

Si néanmoins vous souhaitez effectuer des essais de notre application avec votre propre base de données, nous vous fournissons un fichier docker-compose.yaml dans le serveur afin de pouvoir déployer rapidement une base de données locale. Afin de la créer vous devez avoir préalablement avoir installé docker avant d'exécuter dans le dossier courant la commande : docker-compose up

II. MODULES EXTERNES

a. API

CFF

L'API des CFF demande un lieu de départ, une destination et une heure et renvoie les différents trajets possibles. Il est aussi spécifié dans la requête que nous ne prenons en compte que les trajets, direct, de train. Ils sont renvoyés sous la forme d'un objet JSON contenant aussi un « delay » qui nous permet de savoir s'il y a eu une perturbation sur le réseau et de pouvoir avertir l'utilisateur.

Météo

L'API météo demande un lieu et renvoie un objet JSON contenant une vingtaine de champs comme l'heure du lever et du coucher de soleil, l'humidité, la température maximale, etc. Nous avons décidé de ne garder que les informations suivantes (avec notre parser), la température actuelle, le temps et le lieu.

RTS

Pour récupérer le programme TV proposé par l'API RTS, nous utilisons la librairie OkHTTP. Nous envoyons une requête formée d'une URL et d'une requête contenant les noms des chaines que nous souhaitons avoir dans notre programme.

L'API va nous renvoyer un objet JSON contenant différents champs, comme le type d'émission, l'id de l'émission, la première diffusion, etc.

Nous avons décidé de garder le titre et l'horaire de diffusion.

Remarque : L'utilisation de cette API n'est pas recommandée, il n'y a pas de documentation ni de support. Certaines fonctions ne sont pas utilisables.

Twitter

L'API Twitter demande un pseudo d'un compte auquel l'utilisateur souhaiterait s'abonner. Cette dernière va nous renvoyer un objet JSON contenant les 10 derniers tweets écrits par ce compte.

b. Telegram

Dès le début de ce projet, nous avons souhaité intégrer une intercommunication entre notre application et Telegram. L'objectif étant que les résultats des règles spécifiées par un utilisateur lui soient transmis par message directement sur son compte Telegram de façon automatisée.

Pour mettre en place cela nous avons utilisé l'API Telegram (très bien documentée et facilement appréhendable) afin de rediriger les résultats des requêtes du Task manager (exécutant les règles spécifiées de façon automatisée et a un horaire spécifié par l'utilisateur) vers Telegram.

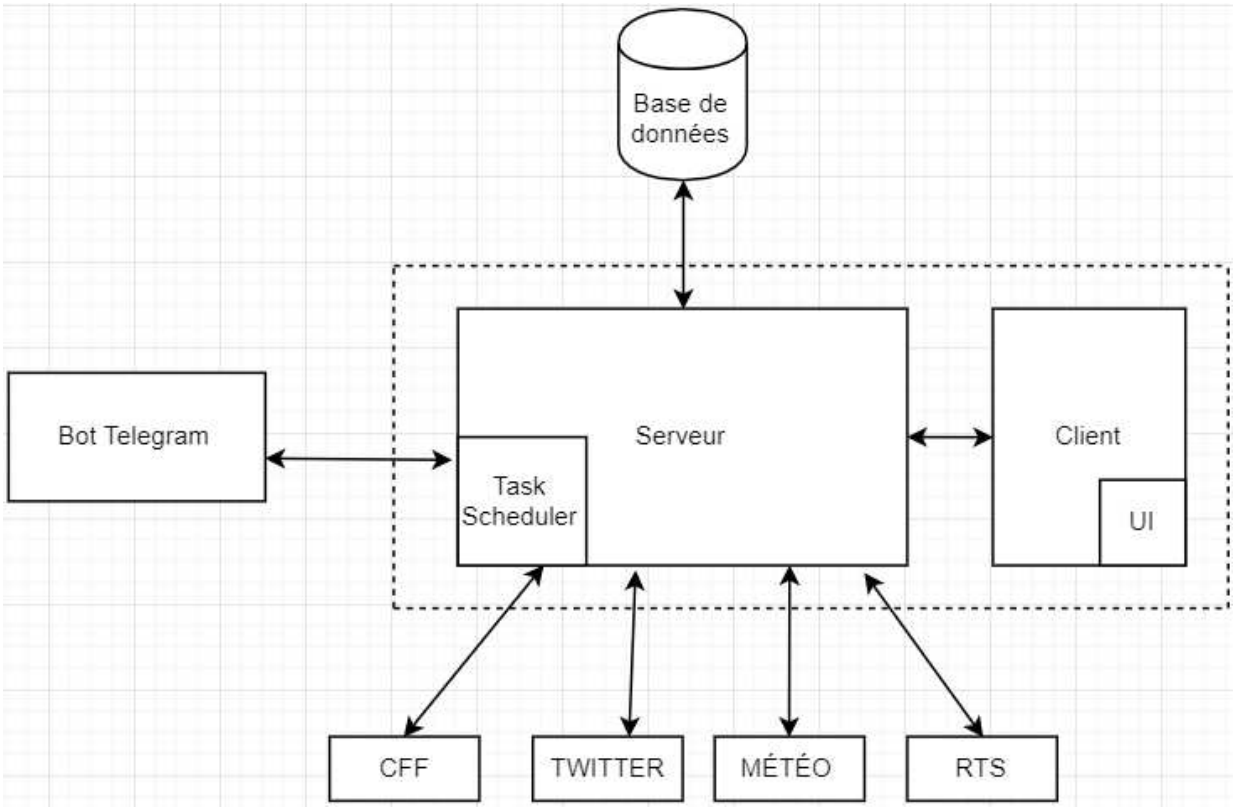
Afin de créer le Bot Telegram permettant cette redirection, nous avons fait appel au BotFather de Telegram.

Dès qu'un nouvel utilisateur s'enregistre sur l'application, un pop-up lui demande s'il souhaite ajouter notre bot Telegram pour recevoir des notifications. En cliquant, il est redirigé sur Telegram et le bot est ajouté, autrement il ne recevra pas de notifications Telegram.

Toute l'implémentation et les requêtes se trouvent dans la classe TelegramNotifications dans le module ASAPP-Server.

2. VISION D'ENSEMBLE

Voici la vue globale de l'application avec les différentes parties permettant le bon fonctionnement de l'application ainsi que quelques informations importantes.



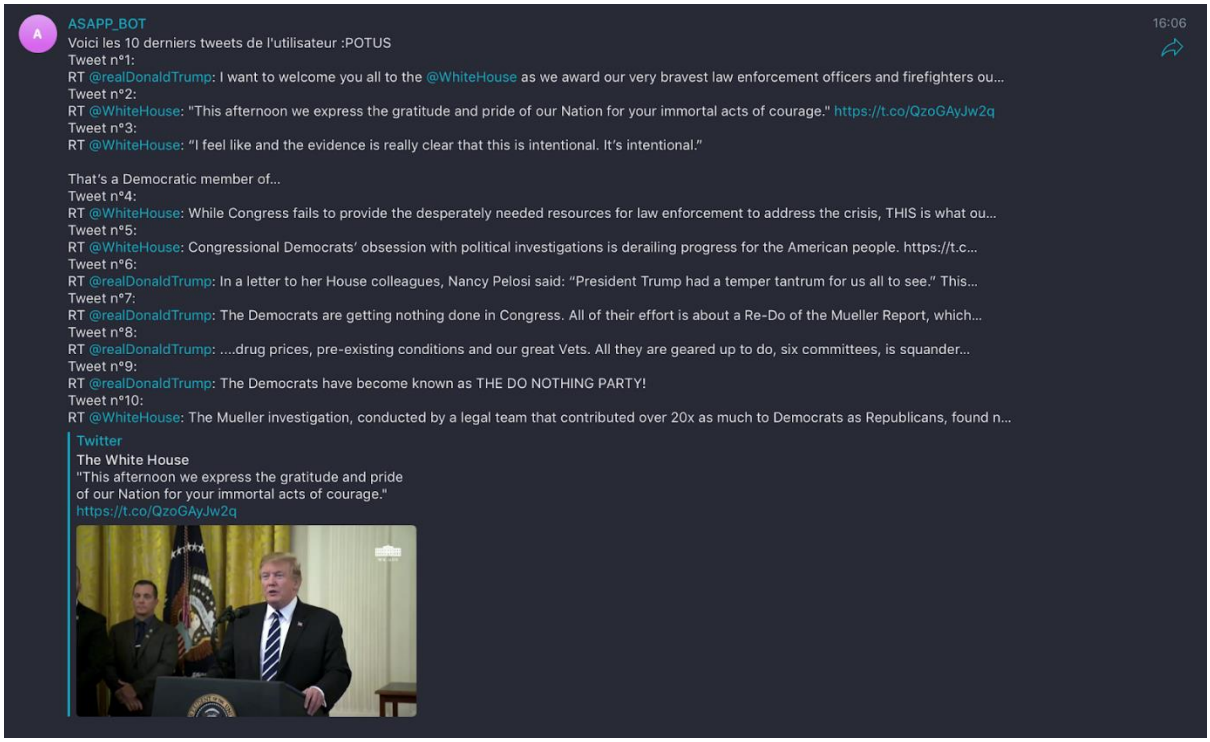
Aspect sécuritaire : Pour garantir un minimum de sécurité, nous avons mis en place une politique de mot de passe grâce à une regex (8 caractères avec une majuscule/chiffre/caractère spécial) et nous hachons le mot de passe avec SHA-512 avant de le saler puis l'ajouter en tant que mot de passe de l'utilisateur.

Filtres de saisie : Nous avons mis en place plusieurs restrictions ainsi que gestion au niveau des saisies utilisateur. Des regex pour différents éléments de nos services nous permettent d'éviter une mauvaise réponse du serveur due.

TaskManager : le gestionnaire d'automatisation des tâches nous permet de récupérer une règle (entré par l'utilisateur) et de renvoyer de façon périodique les résultats de cette règle soit dans la fenêtre principale de l'application, soit directement sur Telegram.

Notification Telegram: Ci-dessous se trouve le résultat d'une règle créé par un utilisateur souhaitant récupérer les 10 derniers tweets de DonaldTrump(@Potus). À travers le ASAPP_BOT l'utilisateur reçoit les nouveaux résultats de sa règle a un intervalle spécifié.

Exemple d'un message du bot Telegram :



3. INSTALLATION

Pour démarrer l'application, il suffit d'exécuter les jars correspondant au serveur et au client (le serveur en premier).

Si vous souhaitez reprendre ce projet et développer/modifier certains aspects, vous devez importer le projet sur votre IDE en important pour chaque module (Client, Server, Common) les fichiers pom.xml correspondant.

Dans les deux cas au-dessus, le serveur ainsi que la base de données utilisées sont celles déployées sur le cloud

Si vous souhaitez effectuer des essais de notre application avec votre propre base de données, nous vous fournissons un fichier docker-compose.yaml dans le serveur afin de pouvoir déployer rapidement une base de données locale. Afin de la créer vous devez avoir préalablement avoir installé docker avant d'exécuter dans le dossier courant la commande : docker-compose up

Celle-ci permet de créer une base de données SQL locale où les credentials sont à spécifier dans le docker-compose.yaml .

4. LIBRAIRIES & OUTILS

Pour l'interface graphique nous avons utilisé le framework JavaFX.

Pour la gestion de dépendance nous avons utilisé l'outil maven.
La librairie OkHTTP a été utilisé pour effectuer des requêtes sur l'API de RTS.

Nous avons déployé le serveur ainsi que la base de données sur le Cloud en utilisant les services d'Amazon Service (RDS & EC2).

5. TESTS EFFECTUÉS

Voici une liste des tests ayant été effectués sur notre application.

Code couleur : Réussite | Bugs restants | Ne fonctionne pas

Tests	Résultats attendus	Résultats obtenus	
Création d'un utilisateur	Un utilisateur est ajouté dans la base de données	L'utilisateur est ajouté dans la base de données mais si l'utilisateur a déjà ajouté le bot Telegram, il peut y avoir un bug	
Connexion d'un utilisateur	L'utilisateur arrive sur la page d'accueil	L'utilisateur arrive bien sur la page d'accueil	
Choix d'une langue par l'utilisateur	La langue de l'application est modifiée	La langue se modifie mais il faut changer de page pour que cela prenne effet	
Modification du mot de passe de l'utilisateur	Le hash du mot de passe est modifié	La modification du mot de passe ne fonctionne pas encore	
Se déplacer sur toutes les pages	Le déplacement sur toutes les pages ne génère pas d'erreur	Se déplacer sur les différentes pages ne génère pas d'erreur	
Création d'une règle pour le service météo	Une règle est ajoutée dans la BD et visible en dessous du service	L'ajout de règle météo fonctionne correctement. Améliorable car il faut changer de fenêtre pour afficher la nouvelle règle dans la liste	
Création d'une règle pour le service CFF	Une règle est ajoutée dans la BD et visible en dessous du service	L'ajout de règle CFF fonctionne correctement. Améliorable car il faut changer de fenêtre pour afficher la nouvelle règle dans la liste	
Création d'une règle pour le service RTS	Une règle est ajoutée dans la BD et visible en dessous du service	L'ajout de règle RTS fonctionne correctement. Améliorable car il faut changer de fenêtre pour afficher la nouvelle règle dans la liste	

Création d'une règle pour le service Twitter	Une règle est ajoutée dans la BD et visible en dessous du service	L'ajout de règle Twitter fonctionne correctement. Améliorable car il faut changer de fenêtre pour afficher la nouvelle règle dans la liste	
Vérifier que chacune des règles s'affiche dans la page d'accueil	Les règles s'affichent dans la page d'accueil	Les règles s'affichent bien dans la page d'accueil	
Vérifier que l'utilisateur reçoit un message sur Telegram (s'il a été programmé)	Un message est envoyé sur le Telegram de l'utilisateur	L'utilisateur reçoit bien les messages Telegram	
Vérifier qu'une règle peut être supprimée	La règle est supprimée de la base de données	Ne fonctionne pas encore	
Supprimer toutes les règles d'une personne	Les règles associées à la personne ont toutes été supprimées de la base de données	Ne fonctionne pas encore	