Rapport de projet d'Android : Confroid

Membres du groupe :

Ailton LOPES MENDES

Jonathan CHU

Fabien LAMBERT--DELAVAQUERIE

Akram MALEK

Gérald LIN

Table des matières

Confroid - Application

Description

Confroid est une application qui permet de centraliser les configurations de toutes vos applications. Il est possible d'ajouter une configuration, de consulter une ou toutes les configurations d'une application. Elles sont stockées dans Confroid sous la forme d'une base de données SQL. Pour communiquer avec Confroid, les applications doivent envoyer des requêtes via une api fournie par Confroid.

La structure Fragment

Un fragment représente une portion réutilisable de l'interface de l'application. Un fragment définit et gère son propre layout, et possède également son propre cycle de vie. Il ne peut exister qu'en étant appelé à partir d'une activité ou d'un autre fragment.

Pour créer un fragment, il faut ajouter les dépendances correspondantes à la librairie et étendre notre classe de la classe Fragment en précisant en paramètre, le nom du layout correspondant au fragment. Nous redéfinissons la méthode "OnViewCreated" dans laquelle nous avons les instructions nécessaires pour l'utilisation du fragment.

Dans notre cas, nos fragments sont appelés depuis un container dans le layout du MainActivity qui permettra par la suite, grâce au Navigation Component, de naviguer entre nos différents fragments.

Navigation Component

La navigation correspond aux interactions de l'utilisateur, lui permettant de naviguer à travers les différentes parties existantes d'une application. L'Android Jetpack Navigation Component permet l'implémentation d'un système de navigation entre les différentes vues d'une application.

La navigation entre les fragments de l'application Confroid est basée sur cette librairie Navigation Component.

1. NavGraph

Le Navigation graph correspond à un diagramme qui permet de définir les chemins destinations et les actions pour chaque layout de celui-ci. Une destination correspond à un layout du Navigation graph.

Ce graphe de navigation est un fichier XML dans lequel nous avons ajouté tous nos fragments et défini les actions entre eux depuis l'éditeur visuel d'Android Studio. Chaque fragment possède un attribut "`android:name=...`" qui est important à définir parce qu'il fait référence à la classe du fragment correspondant.

Ce fichier Navigation graph est ensuite appelé dans le layout hôte qui permettra d'afficher les fragments dans Confroid.

2. Les actions

Les actions sont des connexions logiques entre des destinations d'un graphe de navigation. Elles sont définies par des balises dans le fichier XML du graphe de navigation, et chacune possède un id et un nom d'un layout destination.
Nous avons défini des id explicites permettant de bien comprendre les liaisons entre chaque destination.

Ces id sont des références que nous utiliserons dans notre code et qui permettront de passer d'une destination à une autre.

3. NavHost

Le NavHost correspond au layout qui sert d'hôte pour afficher les éléments du NavGraph.
Dans notre cas, il s'agit du layout de MainActivity dans lequel nous utilisons un décorateur "fragment", auquel nous spécifions les attributs suivants :

- `android:name="androidx.navigation.fragment.NavHostFragment"` → Correspond à la classe et fait office de container. Ce dernier s'occupe d'afficher un à un les layouts du NavGraph lorsqu'ils sont appelés.
- `app:defaultNavHost="true"` → Permet d'intercepter le bouton système Retour.
- `app:navGraph="@navigation/nav_graph"` → Permet de spécifier le graphe de navigation associé à NavHost.

4. NavController et communication entre Fragments

Le Navigation Controller est un objet qui permet de contrôler la navigation entre les destinations d'un NavGraph au sein d'un NavHost.

Dans chaque classe fragment, nous récupérons d'abord le NavController dans une variable qui nous permet par la suite de naviguer entre les fragments, donc les destinations du NavGraph.

Cette navigation s'effectue par le biais des actions que nous avons définies, et nous permet également de passer en paramètre un bundle pour pouvoir transmettre des données exploitables par le Fragment cible.

5. Les animations

Il est possible d'ajouter des animations aux actions. Ces animations rendent la navigation plus fluide à travers les vues de l'application. Elles correspondent à des fichiers XML qui se situent dans le répertoire "anim" des ressources.

Une fois que les fichiers d'animations sont créés, il faut associer les animations aux actions du NavGraph, grâce aux attributs "enterAnim", "exitAnim", "popEnterAnim", "popExitAnim".

6. ViewModel

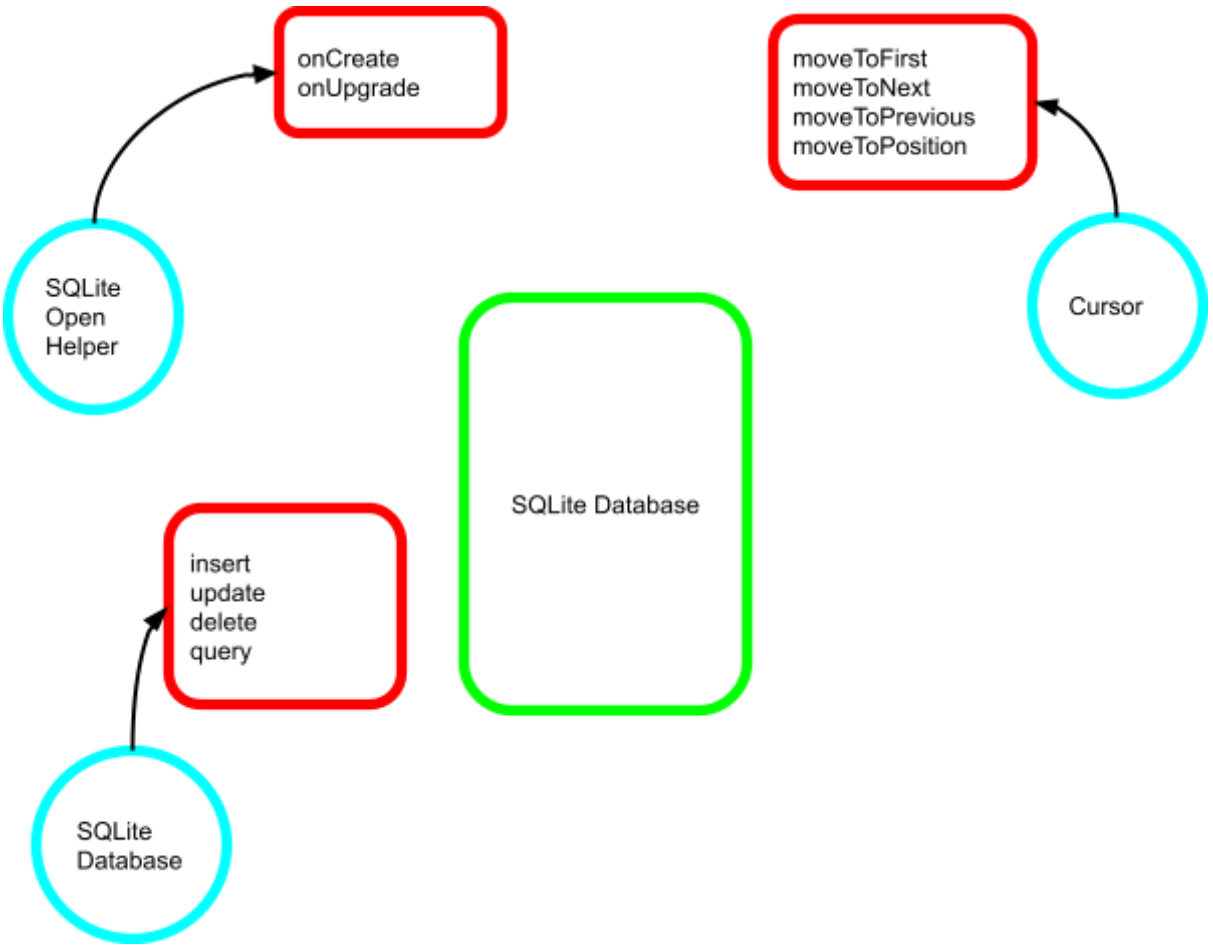
Le ViewModel est la classe qui stocke et gère toutes les données en tenant compte de leur cycle de vie. En plus, le ViewModel permet aux données de survivre aux modifications de configuration telles que les rotations d'écran.

La base de données SQLite

SQLite est un outil qui permet d'intégrer dans une application mobile, une base de données relationnelle, en utilisant la syntaxe SQL.

La création et la mise à jour de la base de données se fait à l'aide de SQLiteOpenHelper. Cette classe permet d'initialiser notre base de données en y précisant son nom et sa version actuelle.
Ensuite, pour travailler avec une base de données SQLite sous Android, nous avons besoin de la class SQLiteDatabase, qui fournit des méthodes qui permettent d'ouvrir/fermer la base de données, effectuer des requêtes via des instructions SQL.

C'est à partir de ces 2 classes appartenant au paquetage SQLite que l'on va pouvoir définir une table pour chaque application qui est répertoriée par Confroid, ajouter une configuration, fournir une ou toutes les configurations à une application demandeuse.

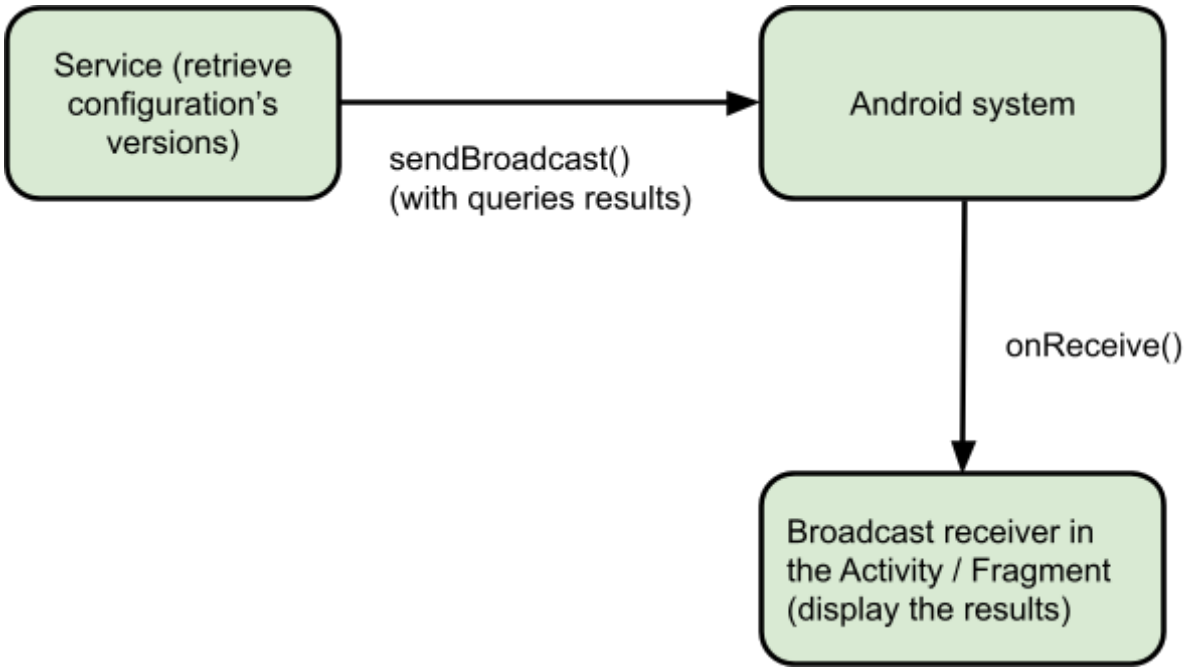


Les Services

Un service est un composant d'une application qui peut effectuer des opérations en arrière-plan sans fournir d'interface graphique. C'est grâce aux services que l'application Confroid va interagir avec la base de données SQL pour fournir les fonctionnalités d'insertion et de récupération de configurations.

Étant donné qu'un service ne possède pas d'interface graphique, le résultat d'une requête doit pouvoir être récupéré par Confroid pour que l'application puisse afficher une configuration par exemple. Le service doit donc stocker le résultat dans le bundle d'une Intent, qui aura défini une action. Ce résultat sera ensuite récupéré par un objet de type BroadcastReceiver dont l'action liée correspond à l'action définie par le service. Une fois le résultat récupéré, l'objet receiver mettra fin au service appelé.

Dans Confroid, les services seront appelés à la suite d'une requête envoyée par une application demandeuse. La classe de service demandée sera directement fournie dans la requête de l'application. Les requêtes envoyées depuis une application externe sont construites à partir de l'api ConfroidUtils.



L'API ConfroidUtils

L'api ConfroidUtils est un outil qui sert de passerelle entre une application qui veut utiliser les fonctionnalités de Confroid et Confroid. Elle se présente sous la forme d'un singleton ConfroidUtils qui possède des fonctions permettant d'envoyer des données, sous la forme de bundle, à Confroid.
Les principales informations présentes dans le bundle sont: le nom de l'application d'où provient la requête, un token d'identification permettant à Confroid d'accepter ou non une requête, et un identifiant pour la requête afin que lorsque Confroid envoie une réponse à l'application demandeuse, celle-ci reconnaît qu'il s'agit de la requête initialement envoyée.

Lorsqu'une application importe l'api ConfroidUtils, celle-ci possède initialement des champs représentant le token d'identification et l'identifiant de la requête respectivement initialisés à la chaîne vide pour le token et à 0 pour l'identifiant de requête.

- Lors de la première communication entre une application externe et Confroid, L'application devra envoyer une demande de token, qui sera envoyé dans la réponse, afin que Confroid puisse accepter les requêtes de cette même application à l'avenir.
- Pour ce qui est de l'identifiant de requête, il sera généré aléatoirement lors de chaque nouvelle requête, afin que l'application demandeuse n'accepte pas de réponse à une requête qu'elle n'a pas demandée.

Pour la sauvegarde d'une configuration dans la base de données de Confroid, l'application externe doit pouvoir stocker son objet représentant la configuration sous la forme d'un objet accepté par un bundle. Pour cela, l'api ConfroidUtils possède des fonctions permettant de parcourir l'objet de configuration, et le convertit en une Map, qui est facilement stockable dans un bundle.

Les Fragments récupérant les résultats des services

1. AppFragment

Il s'agit du Fragment principal de l'application Confroid. Il contient la liste des applications enregistrées dans la base de données sous la forme d'une RecyclerView. Cliquer sur un item de la liste permet d'accéder au Fragment des versions pour une application.
On peut y faire une recherche pour une application spécifique grâce à la barre de recherche, tant qu'elle est répertoriée par Confroid.

2. AllVersionsFragment

A l'ouverture de ce Fragment, une requête dans la base de données sera effectuée grâce au service ConfigurationVersions, ce qui aura pour but d'afficher la liste des versions de configuration pour une application dans une RecyclerView. De même que pour AppFragment, on peut effectuer une recherche d'une configuration par rapport à sa version ou alors son tag.

Pour voir le contenu d'une configuration, on clique simplement sur un item de la liste, qui ouvrira un nouveau Fragment. Si on accède à ce Fragment depuis une demande d'une application externe, il est possible d'envoyer la liste des versions directement à l'application demandeuse.

3. ConfigFragment

Il permet de visualiser le contenu détaillé d'une version de configuration, en y affichant l'ensemble des champs dans une RecyclerView. Tout comme le Fragment précédent, il est possible d'envoyer la version s'il s'agit d'une demande depuis une application externe. Il est également possible de supprimer la version de la base de données Confroid.

Pour un champ contenant d'autres champs, le Fragment n'affiche pas tout son contenu, il faut cliquer sur ce champ pour voir le contenu de ce champ. Pour un champ à valeur unique, on accède à un Fragment qui permet de modifier ce champ.

On considère alors les l'ensemble des champs d'une configuration comme un arbre avec pour feuilles les champs "simples" avec une valeur et les branches, les champs contenant d'autres champs internes.

On navigue donc entre les fragments avec recyclerview pour les branches et affichage simple pour les feuilles qui servira de modification.

4. Modification d'une configuration par fragments

La modification d'un champ pour une version de configuration se fait en cliquant sur un champ à valeur unique. Cela amène à un Fragment qui permet de définir une nouvelle valeur pour un champ donné et de revenir sur le ConfigFragment avec le champ mis à jour.On navigue donc à travers les fragments pour effectuer toutes les modifications souhaitées pour finalement revenir au ConfigFragment.

Cependant, la base de données ne sera pas immédiatement actualisée car il faudra d'abord cliquer sur un bouton de génération d'une nouvelle version pour la configuration. Ce bouton redirige vers un Fragment où l'on pourra donner un tag et un numéro de version pour la nouvelle version que l'on décide de créer. La validation de ce formulaire effectue un ajout dans la base de données.

Nous avons fait ce choix à la manière d'une nouvelle version d'une application, on crée une version 2 à partir d'une version modifiée de la 1 plutôt que de n'avoir qu'une seule version sans "historique". On peut d'ailleurs toujours supprimer la version initiale si on le désire.

WebService

L'objectif de cette partie est d'envoyer et de télécharger les fichiers cryptés contenant les configurations des applications sur un serveur en ligne. Notre choix s'est porté sur Node.js pour développer le WebService favorisant la simplicité, la maintenabilité et la réutilisabilité.

Les dépendances npm nécessaires :

- npm install -g nodemon
- npm install express --save — Simplifie la création d'API et de serveurs Web.
- npm install sqlite3 — La base de données pour le web service.
- npm install knex multer — Facilite la gestion de la base de données.
- npm install swagger-jsdoc swagger-ui-express — Facilite la création de la documentation.

Voici les routes proposés par l'API :

POST /api/auth/login/ ==> Renvoie un token si l'utilisateur est authentifié.

POST /api/auth/register/ ==> Crée un compte pour l'utilisateur.

GET /api/files/ ==> Récupère la liste des noms de fichiers disponibles
(Le token est requis dans l'entête de la requête).

GET /api/file/{name} ==> Récupérer le fichier de configuration.
(Le token est requis dans l'entête de la requête).

POST /api/upload ==> Permet de sauvegarder le fichier de configuration donnée.
(Le token est requis dans l'entête de la requête).

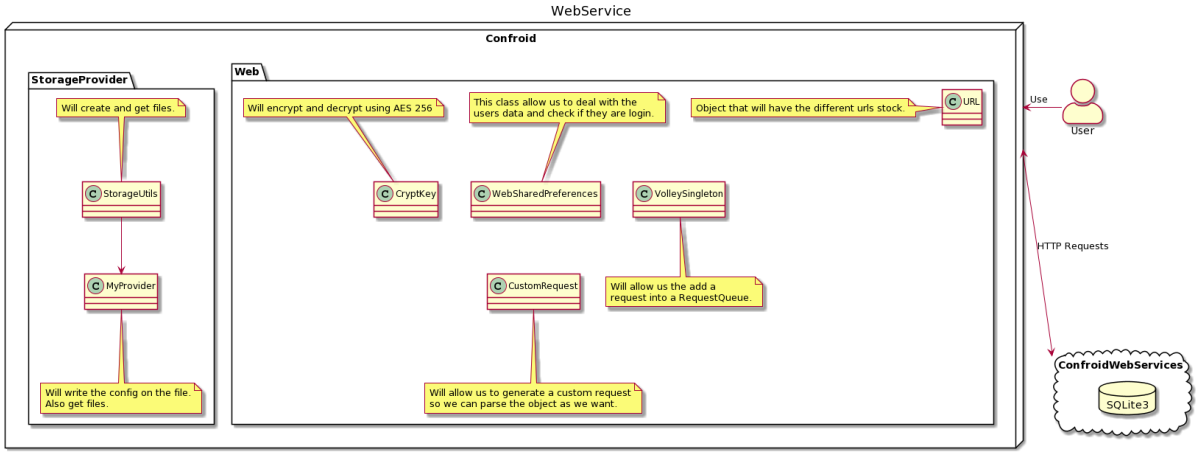
Pour plus d'informations sur le comportement de l'API, veuillez vous référer à la documentation disponible sur le chemin "/api-docs".

Après avoir implémenté l'API, nous avons décidé d'utiliser la bibliothèque Volley pour communiquer avec l'API à partir d'Android. Cette bibliothèque facilite et permet une mise en réseau des applications Android plus rapide.

Nous utilisons un local tunnel pour établir la connexion entre le web service et l'application Android. En effet, il est nécessaire d'avoir un lien HTTP pour transmettre les requêtes. Ainsi nous lançons premièrement le web service en le reliant à l'url : https://valkyroid.loca.lt grâce à la commande `lt -p 8080 --subdomain valkyroid`.

Nous nous sommes référés à la documentation Android de Volley pour ajouter les dépendances dans le build.gradle, pour comprendre son fonctionnement, et également pour pouvoir l'utiliser.

Ensuite nous avons pensé à la structure du code côté Android pour la partie Web Service.



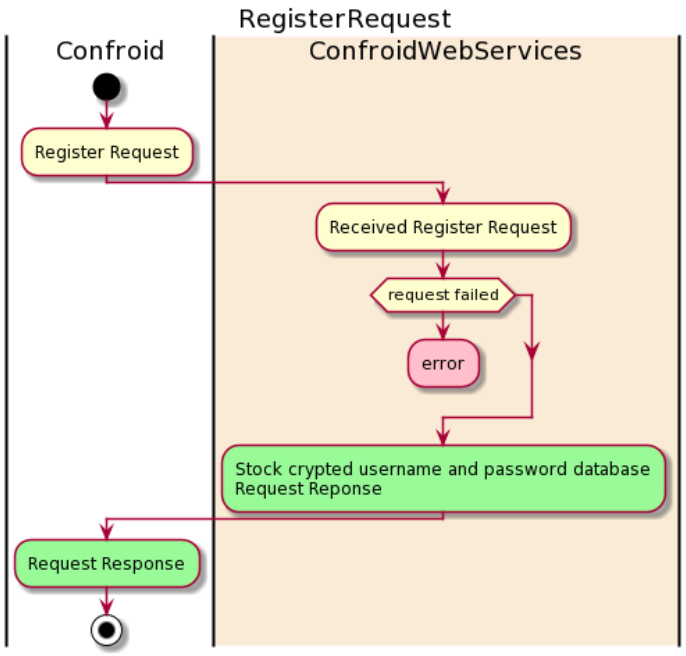
Nous avons séparé la partie WebService en deux packages pour le côté android du projet :

- Le package storageprovider qui s'occupe de la gestion des fichiers, c'est-à-dire la création, l'écriture et la récupération d'un fichier.
- Le package web qui s'occupe de la partie communication vers le Web Service par des requêtes avec les différentes classes de l'image ci-dessus. Elle s'occupe également de la partie sécurité avec la classe CryptKey.

La partie android sert à faire la connexion entre l'utilisateur et le web service. Il existe différentes étapes pour qu'un utilisateur puisse interagir avec ses données stockées dans le web service.

Tout d'abord un utilisateur crée son compte lors de sa première utilisation de l'application. En commençant par la page d'inscription.

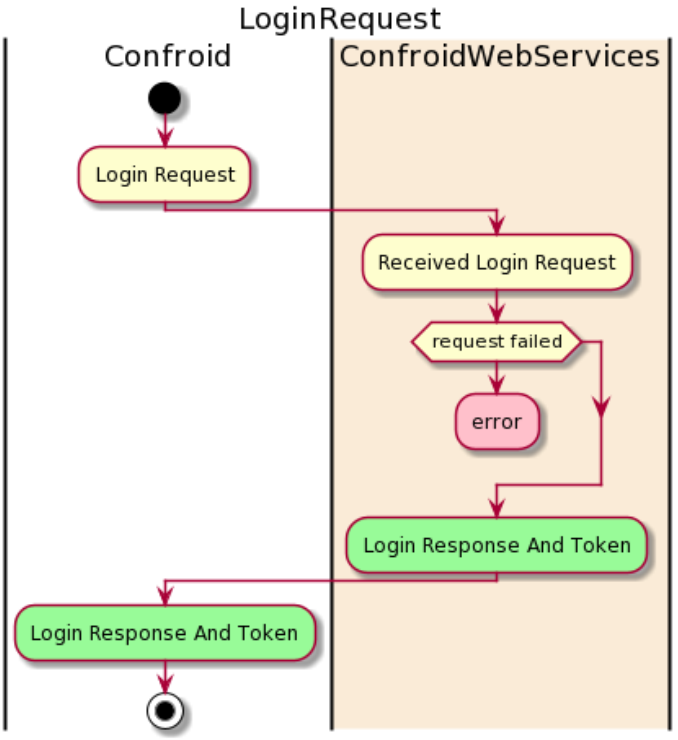
Suite à l'inscription d'un compte grâce à un nom d'utilisateur et un mot de passe, le programme Confroid enverra par la suite une requête.



En effet, Confroid envoie une requête à l'API avec le nom d'utilisateur et le mot de passe cryptés. Celle-ci vérifie ensuite le format de la requête. Si elle n'est pas correcte ou s'il y a eu un problème, elle renvoie une réponse d'erreur à Confroid. Sinon elle stocke le nom d'utilisateur et le mot de passe cryptés dans la base de données SQLite3 du web service.

Pour le cryptage, nous cryptons deux fois dans Confroid avant d'envoyer les informations, puis nous cryptons encore une fois depuis le web service.

Finalement le web service répond par un statut de réponse 200 pour indiquer la réussite de la requête.

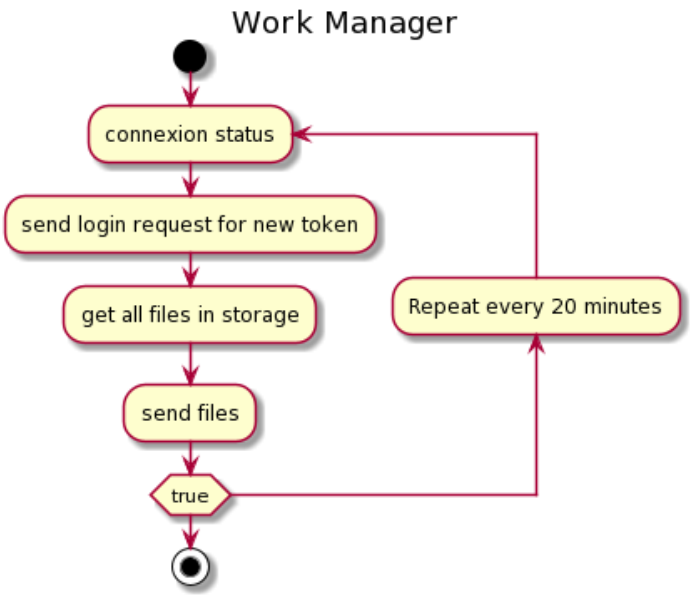


Lors de la connexion d'un utilisateur, Confroid envoie une requête à l'API avec le nom d'utilisateur et le mot de passe pour se connecter et demander un token.

Tout d'abord, Confroid envoie une requête contenant le nom d'utilisateur et le mot de passe cryptés. Ensuite, le web service vérifie le format de la requête, mais également la présence de l'utilisateur dans la base de données. S'il n'existe pas, le web service renvoie une réponse d'erreur à Confroid. Sinon il génère un token qui devra être réinitialisé toutes les 30 minutes.

Le web service renvoie le statut de réponse 200 pour indiquer la réussite de la requête en plus du token. A la réception de la réponse, nous décryptons une fois le mot de passe et le stockons dans nos SharedPreferences. Cela permet de rester connecté à l'application même après l'avoir relancée.

Une fois que l'utilisateur est connecté, il peut voir les différents fichiers qui sont stockés dans sa base de données. Les fichiers de l'utilisateur seront stockés dans un dossier interne au téléphone. Nous avons dû ajouter les permissions de lecture et d'écriture dans un dossier externe à l'application. De plus, nous avons attribué un provider pour pouvoir récupérer et modifier les fichiers.



Pour afficher les fichiers, Confroid envoie des requêtes d'envoi des fichiers cryptés automatiquement au WebService en utilisant le principe du WorkManager. Le WorkManager permet de planifier des tâches régulières qui s'appliqueront même si l'application Confroid est fermée.

Le WorkManager vérifie si l'utilisateur est connecté puis envoie une requête de login pour demander à nouveau le token. Ensuite, il récupère tous les fichiers stockés dans la mémoire interne du téléphone et les envoie au web service. Si la réponse suite à l'envoi d'un fichier n'est pas bonne, Confroid déconnecte par défaut l'utilisateur.

L'API du WebService stock les configurations reçues sous forme de fichiers. En effet, ces dernières sont sauvegardées dans un dossier "ressources/config-uploads/{id}" où pour chaque utilisateur un nouveau dossier lui est attribué en fonction de son identifiant qui contient ses configurations personnelles.

Répartition des tâches

Le projet Confroid se divise en 2 parties. Il y a dans un premier temps la partie application mobile qui consiste à centraliser les configurations des applications qui communiquent avec Confroid dans une base de données. Il y a également la partie web services qui permet à l'application Confroid d'importer et d'exporter les configurations pour chaque utilisateur depuis le site valkyroid.

Ailton et Akram se sont occupés de la partie web service, en développant la gestion d'authentification d'un utilisateur, le stockage des configurations dans une base de données et l'appel à ce web service par l'application Confroid.

Fabien, Gérald et Jonathan ont travaillé sur l'application mobile. Leur travail consistait à développer la base de données pour stocker les configurations, développer des applications de test, créer un moyen de communication entre ces applications et Confroid, créer l'api ConfroidUtils afin qu'elle soit importée par les applications externes.

Une fois ces 2 parties fonctionnelles, elles ont été convergées puis l'ensemble de l'équipe était chargée de s'assurer que tout fonctionnait, de détecter d'éventuels bugs, de les corriger, d'ajouter une couche de design à l'application, et de développer de nouvelles fonctionnalités.

A la fin de chaque tâche, le membre de l'équipe qui traitait cette tâche créait une pull request sur Github en notifiant le reste de l'équipe que le code avait besoin d'une relecture avant d'être déployé sur la branche principale.
Nous organisions également une réunion chaque semaine afin de revenir sur la semaine passée, sur les fonctionnalités qui ont été développées. Elle permettait également de préparer les semaines à venir, en définissant de nouvelles tâches et de les attribuer à chacun.

Difficultés rencontrées

Avant de commencer à développer le projet, une première difficulté que nous avons rencontrée, était de bien comprendre chaque partie du projet Confroid. Nos premières réunions d'équipe consistaient à mettre en commun nos connaissances du projet et de bâtir l'architecture afin de pouvoir facilement répartir les tâches entre les membres.

Mais il arrivait que pendant la période où chacun travaille individuellement sur sa tâche, un membre avait du mal à se lancer dans le développement car il n'aurait pas saisi un point de sa tâche. Dans ce cas, soit les personnes qui travaillaient sur un module en commun, organisaient une réunion pour essayer de résoudre ce problème, soit toute l'équipe se réunissait, soit nous contactions directement le professeur afin d'être sûr de ne pas partir dans une mauvaise direction.