Security Notifications - Rapport de développement

Anis Smail Jefferson Mangue Quentin Campos
Sylvestre Massalaz Yann Bilissor

14 mars 2016

Table des matières

1 Application

1.1 Service de localisation

Le service le localisation est le service le l'application qui permet de manipuler les positions de l'utilisateur. Il est basé sur l'api Google Play Service au lieu de celle native d'android, pour des raisons d'optimisation.

En effet, cette api permet de limiter le nombre de requêtes en ne les faisant que lorsqu'elles sont nécessaires et ainsi réduire la consommation de la batterie.

1.1.1 Algorithme spécifique

Si la carte GoogleMap se charge avant le service, elle ne peut pas appeler celui-ci sous peine de provoquer une NPE. Nous avons donc utilisé le design-pattern du producteur-consommateur afin d'avoir une liste de tâches à effectuer, qui sont ajoutées dans le bon ordre coté producteur, et consommée dans le même ordre strictement coté consommateur.

1.2 HomeActivity

La HomeActivity échange principalement avec le NetworkService. Elle s'occupe de la connexion, l'inscription, la gestion des paramètres ainsi que l'émission d'alerte.

1.2.1 Traitement des échanges avec le NetworkService

La HomeActivity est un multiplexeur de LocalIntent émis par le NetworkService. A tout instant, elle gère un FragmentReceiver, qui est un Fragment pouvant recevoir et traiter un LocalIntent.

Ce LocalIntent provient obligatoirement du NetworkService et est filtré par la HomeActivity. Chaque instance de FragmentReceiver décrit l'Intent qu'elle est capable de traiter, et hérite de la classe abstraite BaseFragmentReceiver, class permettant la gestion des état du Fragment dans le protocole d'échange avec le NetworkService.

1.2.2 Insertion du FragmentReceiver dans la l'activité principale

Afin de traiter les messages reçus par la HomeActivity, un FragmentReceiver doit être attachée à celle-ci. L'activité lui déléguera l'Intent si et seulement si ce Fragment indique qu'il est capable de le traiter. Pour qu'un FragmentReceiver

soit affiché et géré par cette activité, il doit être soumis à la méthode HomeActivity#showFragment. Ce processus est implanté au sein de la classe abstraite BaseFragmentReceiver.

En conclusion, cet agencement de nos classes nous permet d'implanter une sous vue des actions de base, leurs opérations, ainsi que leurs échanges réseaux de façons automatique, maintenable et centralisée. Déplaçant concrètement les opérations de contrôle de l'activité.

2 Back-end

Afin de fonctionner, l'application fait appel à un serveur externe.

2.1 Technologies utilisées

Le serveur est codé en python et utilise le Framework Django, qui permet de concevoir rapidement un serveur pouvant répondre à des requêtes HTML à des url données.

Par exemple, django permet d'associer une fonction python spécifique à chaque type de requête (GET, POST, ...) sur `site.fr/path/of/service`.

> **Note** : Dans notre cas, nous n'utilisons que des requêtes POST.

2.2 Architecture

Le framework Django est basé sur le concept **Modèle - Vue - Contrôleur**. Dans notre cas, l'application Android fait office de vue, et contient une partie du contrôleur ; c'est donc uniquement le modèle et la partie serveur du contrôleur qui est gérée par Django.

2.2.1 Modèle

Le modèle s'articule autour de deux entités principales qui contiennent les données principales de gestion du serveur.

User :

Les utilisateurs de l'application sont identifiés par une adresse mail et un code pin à 4 caractères. En parallèle, d'autres informations personnelles de l'utilisateur sont stockées : Nom et prénom, numéro de téléphone, et adresse email.

De plus, des données internes à l'application sont enregistrées :

— Un id unique

— Le rayon de détection des alertes
— La dernière position connue de l'utilisateur.

Alerte :

Les alertes peuvent être crées par des utilisateurs ; elles correspondent à une position.

Les autres champs sont :

— Un id unique
— L'auteur de l'alerte
— Un nom éventuel
— L'activité de l'alerte (en cours / terminée)

Vote :

Les utilisateurs peuvent donner un avis sur une alerte et voter pour indiquer si elle est véridique ou s'il s'agit d'une fausse alerte. Ceci permet d'utiliser la communauté pour vérifier la validité d'une alerte.

Un vote est associé à un **Utilisateur** et une **Alerte** et correspond simplement à un champ booléen `vrai/faux`.

Session :

Lors de la connexion de l'utilisateur, une session est créée dans la base de donnée. Elle permet de définir une date d'expiration pour la connexion des utilisateurs et est associée à un identifiant de session unique qui permet aux utilisateurs d'effectuer leurs requêtes sans donner à chaque requête leurs informations de connexion.

— Un identifiant unique
— L'utilisateur concerné (clef étrangère)
— Une date d'expiration.

2.2.2 Contrôleur

Les calculs effectués par le serveur permettent de gérer le fonctionnement interne de l'application.

— Lorsque l'utilisateur transmet sa position gps (en faisant une requête pour la liste des alertes actives ou en déclarant une alerte), ses coordonnées actuelles sont mises à jour.
— Lorsque l'utilisateur fait une requête pour connaitre les alertes dans sa zone, il transmet un rayon de detection. Celui-ci est stocké dans la base de donnée pour cet utilisateur.
— Lorsqu'une nouvelle alerte est déclarée, tous les utilisateurs dont le rayon de détection fait que cette alerte est à leur porté sont notifiés de cette création.

2.2.3 Base de données

La base de donnée utilisée est intégrée à Django. Il s'agit de Spatialite, une variante de SQLite qui permet des requêtes efficaces sur les positions géographique.

2.2.4 Notifications

Afin de pouvoir signaler dynamiquement au clients l'apparition de nouvelles alertes près de leur position entre deux appels aux méthodes REST, nous avons utilisé le système de notification proposé par Google (Google Cloud Messaging). Il nous permet plutôt simplement d'envoyer des messages aux diférents clients depuis le serveur.

Répartition des messages :

Il est possible de transmettre les messages descendants de deux façons dans GCM : directement à un appareil et en utilisant un système de sujets. La deuxième version étant plus simple à implémenter (pas de gestion des id clients), nous avons opté pour la proposition suivante :

- les utilisateurs ont un topic "user-UUID"
- les alertes ont un topic nommé "alert-UUID"

Ainsi, l'application Android peut s'abonner aux sujets qui la concerne durant son utilisation, soit le sujet de l'utilisateur courant et le sujet des alertes à proximité.

Les sujets des utilisateurs sont utilisé pour leur transmettre les informations concernant de nouvelles alertes apparues à proximité de leur dernière position connue tandis que les sujets des alertes sont utilisés pour prévenir les utilisateurs concernés des mises à jour concernant lesdites alertes (ici, uniquement leur clôture).

2.3 API REST

Voir annexe spécification.pdf.