

Rapport de projet AMIO

Réalisation d’une application Android exploitant des données IoT

Nom des élèves : Pierre Haller et Florian KROMER

Élèves Ingénieur en 3ème année

Sommaire

[Cahier des charges 3](#_Toc477080567)

[Objectifs 3](#_Toc477080568)

[Technologies 3](#_Toc477080569)

[Réalisation 4](#_Toc477080570)

[Récupération des informations 4](#_Toc477080571)

[Détection d’une lumière allumée 4](#_Toc477080572)

[Tests unitaires 4](#_Toc477080573)

# Cahier des charges

## Objectifs

L'objectif de ce mini-projet est de réaliser une application Android exploitant des données issues d'un réseau de capteurs et exposées à travers un web service (IoTLab de TELECOM Nancy). Le but est de détecter les lumières laissées actives dans les bureaux en soirée.

L'application doit permettre les actions suivantes:

* Lister dans l'activité principale les capteurs actifs et les valeurs de luminosité qu'ils relèvent, en mettant en évidence ceux qui indiquent la présence d'une lumière active
* Emettre une notification si une nouvelle lumière vient d'être allumée en semaine entre 19h et 23h, en spécifiant le capteur impliqué
* Envoyer un email si cet événement survient le week-end entre 19h et 23h ou en semaine entre 23h et 6h.
* Permettre la configuration des plages horaires et de l'adresse email dans un menu dédié

De plus, l’état des capteurs seront récupérées dans un **Service** asynchrone (**AsyncTask)** qui exécutera une tache périodique (**TimerTask**). Celle-ci aura comme but la consultation d'un web service avec **HttpURLConnection** et la lecture des réponses avec **JsonReader**. La persistance des données sera faite avec **SharedPreferences** et la création d'un **BroadcastReceiver** pour gérer les événements système sera necessaire. Enfin il faudra éditer le **Manifest** pour déclarer des permissions et des **intent-filter**, et l'envoi de **Notification**.

## Technologies

Android studio : (v2.2.3) IDE, minSDKversion : 16, targetSDKversion : 21

Graddle : gestionnaire dépendances

Gson : (v2.2) bibliothèque permettant une manipulation du JSON plus souple que l’api de base. Permet également de caster une chaine json en son objet correspondant.

JUnit : (v4.12) biblitotheque permettant l’organisation et les assertions des tests unitaires.

JavaFaker : (v0.12) bibliothèque permettant de générer divers type de données, utile pour les tests unitaires.

L’application a été testé sur un Xiaomi redme note 3 (android 6.0) et un nexus 4 (android 7)

# Réalisation

## Récupération des informations

Notre activité principale va démarrer un service lors du clic sur le bouton « On », par ailleurs si la checkbox est cochée le service démarrera automatiquement. Dans tous les cas, on va déclencher l’appel à notre service (MainService) qui va créer une tache périodique (LightTimerTask) qui sera appelé toute les 10 secondes. Celle-ci lance une tache asynchrone (RestTask) qui va s’occuper d’appeler le web service. La RestTask va se connecter à l’url donnée, et simplement retourner le json obtenu. Ce json va être transformé en une Arraylist de lumière (i.e. les items du json) pour faciliter la manipulation.

Ensuite un callback va être appelé, cela correspond à la fonction de publication (ou mise à jour) des valeurs déclarée dans notre MainService. Celle-ci va transférer les résultats à notre activité principale à l’aide d’un broadcast. Dans le MainActivityBroadcastReceiver on va récupérer la liste de lumière, et demander la mise à jour de la vue.

## Détection d’une lumière allumée

Lors du passage du json à une liste de lumières, on doit détecter l’allumage d’une pièce. Pour cela nous utilisons la moyenne glissante : si une valeur fait augmenter la luminosité d’une pièce de 50%, nous considérons que la lumière a été allumée.

Par exemple nous recevons, pour une mote ces valeurs :

X = 10 (‾x = 10). La lampe est éteinte.

X = 10;18 (‾x = 14). La moyenne n’a pas augmenté de 50% ou plus, la lumière n’est pas allumée.

X = 10 ;18 ;40 (‾x = 22,66). La moyenne a augmenté de plus de 50% (14\*1,5=21 < 22,66), la lumière est allumée, le changeListener est déclenché.

Lorsque le ChangeListener a été déclenché, il faut faire une action en fonction de l’heure courante :

* Si l’heure est comprise entre 18 et 23, on va utiliser le système de notification d’android.
* Sinon on va envoyer un email à l’utilisateur.

Le MailManager est une surcouche au système de mail de java, il va simplement lire le fichier de propriétés pour récupérer un accès à un serveur SMTP (gmail en l’occurance). On va s’assurer de ne pas spammer l’utilisateur en n’envoyant un que toutes les 30 secondes.

Le LightNotification a été généré automatiquement grâce à l’IDE. On va simplement lui fournir le contexte, le titre de la notif et la description. Il va s’occuper de remplacer ces valeurs dans le template correspondant.

## Start on Boot

Une checkbox est présente sur l’interface permettant à l’utilisateur d’activer ou désactiver la fonctionnalité.  
La variable booléen représentant la checkbox va être rendu persistante grâce à l’utilisation de « SharedPreference » :

prefs.edit().putBoolean(***startOnBoot***, b).apply();

Cette ligne est exécutée lorsque l’utilisateur change le contenue de la checkbox.

La donnée peut ensuite être lu en une ligne :

**boolean** isStartChecked = prefs.getBoolean(***startOnBoot***, **false**);

Grâce à un « BroadcastReceiver » on peut lancer le service au démarrage lorsque l’action :

**android.intent.action.BOOT\_COMPLETED"**

est reçu, si le startOnBoot a été activé par l’utilisateur.

## Tests unitaires

Nous avons créé un package différent de celui des sources qui est uniquement destiné aux tests. Nous avons utilisé JUnit

Pour s’assurer de la bonne communication entre notre TimerTask et le web service, nous avons mis en place différents tests unitaires. L’organisation générale du code nous a permis facilement de tester chaque fonctionnalité indépendamment du reste. Par l’exemple nous avons pu injecter un « faux » contexte propre aux les tests, de même nous pouvons injecter l’url à appeler dans notre TimerTask à la volée, enfin le callback appeler à la fin de l’execution de la tache périodique nous a permis de valider l’assertion.

Dans le code ci-dessous nous testons si l’appel à l’API retourne au moins 1 résultat.

@Test  
public void testRESTLightLast(){  
 LightTimerTask timerTask = new LightTimerTask() {  
 @Override  
 public void myTimerTaskCallback(LightRecords lightsRecordsList) {  
 *assertTrue*("/{experiment\_id}/{labels}/(last|first) contains at leats 1 results",lightsRecordsList.size()>=1);  
 }  
  
 @Override  
 public Context myTimerTaskContext() {  
 return getContext();  
 }  
  
 @Override  
 public String myTimerTaskUrl() {  
 return "http://iotlab.telecomnancy.eu/rest/data/1/light1/last";  
 }  
 };  
 timerTask.run();  
  
}

### Test de la détection d’allumage

Nous avons également testé la fonctionnalité de détection d’allumage. Grâce à un Faker, nous envoyons une liste de valeurs généré aléatoirement comprises entre 10 et 20. La lumière doit donc être considéré comme éteinte (1ère assertion). Ensuite nous générons des valeurs comprises entre 40 et 50, la lumière doit être détectée comme allumé (2nd assertion).

# Conclusion

Ce projet nous a permis d’apprendre à développer sur android et de voir le coté graphique (activité, settings, arrayadapter) et aussi le côté pratique (communication avec webservice, vibration, envoi de mail, notification). Nous avons aussi pu coder une suite de test pour valider la pertinence de nos développements. Enfin nous avons porté une attention particulière à la lisibilité et l’organisation du code.

Pour conclure, le développement sur android fut assez agréable car nous connaissions bien l’environnement Java. De plus, la documentation est très bien fourni, l’IDE est correct (bien que nous soyons habitué à eclipse), et le gestionnaire de dépendance graddle fonctionne bien également. Néanmoins il serait intéressant de comparer le temps mis pour développer une application native et la même avec une développement multiplateforme grâce à Cordova / ionic.