

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](#_Toc476992753)

[Objectifs 3](#_Toc476992754)

[Technologies 3](#_Toc476992755)

[Réalisation 4](#_Toc476992756)

[Tests unitaires 4](#_Toc476992757)

# 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

## Détection de l’allumage

Pour détecter l’allumage d’une pièce, nous utilisons la moyenne glissante. Si une nouvelle 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.

## 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 exemple nous avons pu injecter un « faux » contexte propre aux 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’exécution 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).