III.1 Introduction :

Android a connu une très grande diffusion ce dernier temps car cette pile logicielle offre plusieurs services grâce à ses APIs gratuites qui sont accessibles pour utiliser ou implémentations personnalisées. Dans ce chapitre, nous allons décrire les APIs Android utilisées au sein de notre projet ainsi que les classes prises en charge et leurs méthodes qui ont servi au développement de notre application. Nous nous sommes particulièrement intéressé à la localisation décrite en premier par la suite aux API manipulant les protocoles de la Air Interface au niveau des stations mobiles.

III.2 La géolocalisation Sous Android :

L'une des caractéristiques déterminantes des téléphones mobiles est la géolocalisation, ce chapitre couvre également les services basés sur la localisation (LBS) qui vous permettent de fixer l'emplacement du périphérique. Elles comprennent les technologies telles que les techniques de détection à base de localisation Wi-Fi, GPS et cellulaire.

La géolocalisation Sous Android a le pouvoir de spécifier la technologie à utiliser explicitement par le développeur, ou il est possible de fournir un ensemble de critères en termes de précision, de coût, et d'autres exigences et laisser Android Sélectionnez le service le mieux approprié. Google Maps et les services basés sur la localisation utilisant la latitude et la longitude pour représenter les positions géographiquement. En effet, les utilisateurs sont plus susceptibles de penser en termes d'une adresse de rue. La bibliothèque des cartes comprend un géo codeur qu'il est possible d'utiliser pour convertir dans les deux sens latitude / longitude et adresses urbaine.

III.2.1 Google Maps API :

L'API Google Maps est l'une des applications de cartographie les plus utilisées au monde. C'est une application de service de géolocalisation gratuite en ligne. Il s'agit d'un géoportail lancé il y a quelques années aux Etats-Unis puis et Europe. Elle offre une vue de carte sur quatre plans à savoir un plan classique, un plan en image satellite, un plan mixte et un plan relief de la région. Pour intégrer ces cartes interactives Google Maps à sa propre application et bénéficie des données associées, l'utilisateur doit disposer d'une clé (Google Maps API Key) propre à son domaine d'utilisation.

Figure III.1 : *Représentation d'une clé API.*

III.2.2 L'utilisation des services de localisation :

Les services basés sur la localisation est un terme générique qui décrit les différentes technologies que vous pouvez utiliser pour trouver l'emplacement actuel d'un périphérique. Les deux principaux éléments de LBS sont :

- **Location Manager** : Fournit des crochets aux services basés sur la localisation.
- **Location Providers** : Chacun de ces fournisseurs représente une technologie de localisation utilisée pour déterminer l'emplacement actuel de l'appareil.

L'utilisation du gestionnaire d'emplacement permet d'obtenir votre emplacement actuel et la suite du mouvement. L'accès aux services basés sur la localisation est assuré par le gestionnaire d'emplacement. Pour accéder à l'emplacement il faut demander une instance de la LOCATION_SERVICE en utilisant la méthode **getSystemService**.

Code d'accès au service :

```
String Service =Context.LOCATION_SERVICE.
LocationManager  locationManager ;
locationManager = (LocationManager) getSystemService(Service) ;
```

Avant de pouvoir utiliser les services basés sur la localisation, nous avons besoin d'ajouter une ou plusieurs balises utilisations-autorisation au fichier Manifest.xml. L'extrait suivant montre comment demander les autorisations grossières dans ce fichier :

<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/><uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION "/>.

La classe LocationManager comprend des constantes de chaîne statiques qui renvoient le nom du fournisseur pour les trois fournisseurs de localisation :

- ✓ ***LocationManager.GPS_PROVIDER.***
- ✓ ***LocationManager.NETWORK_PROVIDER.***
- ✓ ***LocationManager.PASSIVE_PROVIDER.***

Dans la plupart des scénarios, il est peu probable de choisir explicitement un emplacement fournisseur à utiliser. C'est mieux pratique pour spécifier les besoins et laissez Android détermine la meilleure technologie à utiliser. L'utilisation de la classe **Criteria** permet de dicter les exigences d'un fournisseur en termes de précision, la consommation d'énergie (Faible, moyenne, élevée), le coût financier, la capacité de renvoyer des valeurs pour l'altitude et la vitesse.

Code d'utilisation de la classe **Criteria** :

```
Criteria criteria = new Criteria ();
criteria.setAccuracy (Criteria.ACCURACY_COARSE); criteria.setPowerRequirement (Criteria.POWER_LOW);
criteria.setSpeedRequired (false);
criteria.setCostAllowed (true);
```

Après l'introduction des différents besoin il suffit d'appeler le fournisseur qui offre le mieux service depuis ces critères comme le montre la ligne suivante :

```
String Best_Provider = locationManager.getBestProvider (criteria, true);
```

La classe **Location** offre l'accès à récupérer plusieurs paramètres concernant lesinformations où se situe notre périphérique via les méthodes décrites dans la suite :

- **getLatitude() :** Permet d'avoir la latitude.
- **getLongitude() :** Permet d'avoir la longitude.
- **getLocality():**Permet de savoir le lieu de périphérique.
- **getPostalCode():**Permet de récupérer le code postal du lieu.
- **getCountryName() :**Permet de connaître le nom du pays.

III.2.3 L'affichage des paramètres au niveau du périphérique :

Dans les instructions suivantes nous montrons comment afficher ces paramètres sur l'écran du périphérique sous forme d'une activité une fois l'utilisateur demande sa position.

Ces instructions XML créent un texte au niveau de l'activité :

<TextView

android:id=”@+id/myLocationText”

android:layout_width=”match_parent”

android:layout_height=”wrap_content”

android:text= ‘’” layout_height=”wrap_content”

android:text=” ”/>

La ligne suivante relie le code XML avec l’activité :

TextView myLocationText = (TextView) findViewById (R.id.myLocationText);

Une fois ces instruction sont suivies en ayant une clé API qui permet d’afficher la carte Google Map nous aurions une exécution comme le montre la figure suivante sous forme d’un exemple :

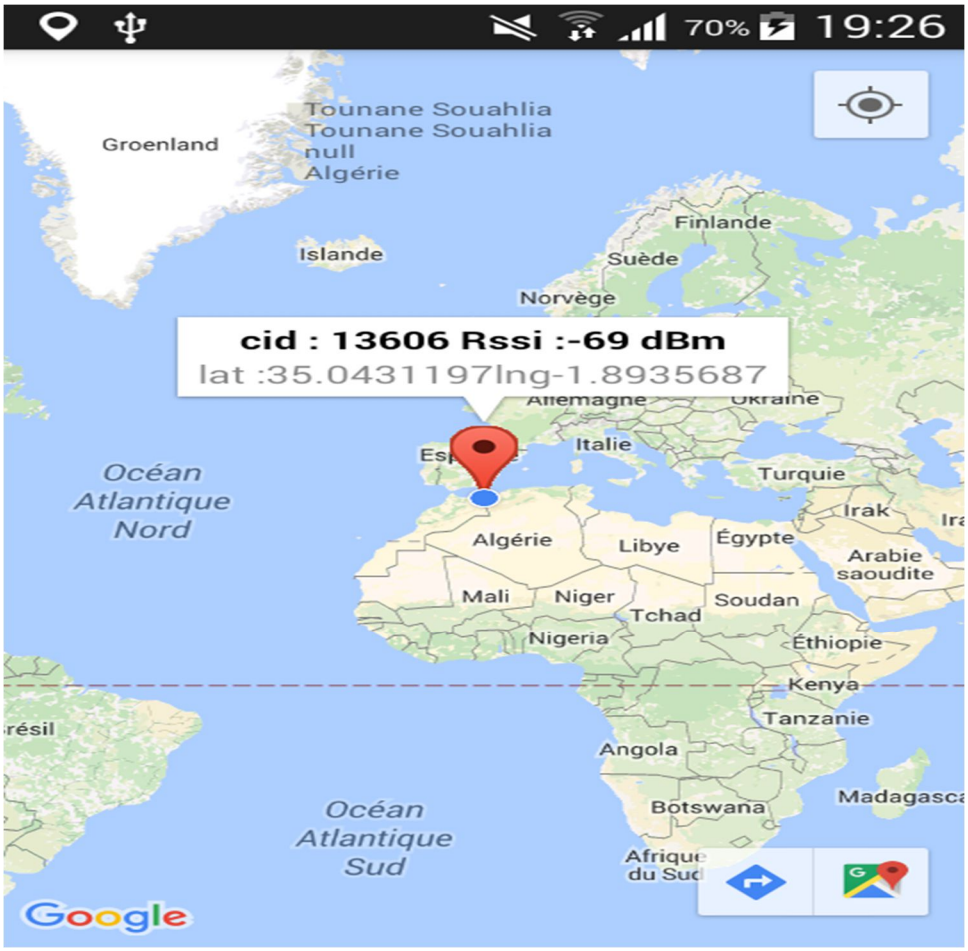

Figure III.2 *: une carte Google Map avec les paramètres de localisation détectée par la station mobile.*

III.3 La téléphonie et les paramètres la signalisation :

Dans cette partie nous expliquons les APIs Android utilisées dans la téléphonie et l'implémentation des classes qui sert à la récupération des informations GSM et les paramètres de la signalisation de l'Air-Interface.

III.3.1 Accès aux propriétés de la téléphonie et l'état du périphérique :

L'accès aux API de la téléphonie est géré par la classe **TelephonyManager**, accessible en utilisant le Méthode **getSystemService**, les lignes suivantes montrent la procédure :

String Service_Name = Context.TELEPHONY_SERVICE;

TelephonyManager telephonyManager = (TelephonyManager) getSystemService (Service_Name);

La TelephonyManager offre un accès direct à la plupart des propriétés de téléphone, le réseau, le module d'identité d'abonné (SIM) et les détails de l'état des données. Vous pouvez également accéder à certaines connectivités. Les informations d'état, même si cela se fait généralement à l'aide du gestionnaire de connectivité.

III.3.2 L'écoute des paramètres du téléphone et du réseau mobile :

En utilisant la TelephonyManager. Nous pouvons obtenir le type de téléphone concernant la technologie qu'il utilise (GSM, CDMA).

Le type de téléphone est accessible via la méthode**telephonyManager.getPhoneType ().**
Ces trois paramètres suivants montrent quelle technologie le périphérique utilise :

- **TelephonyManager.PHONE_TYPE_CDMA) :**s'il s'agit de la CDMA.
- **TelephonyManager.PHONE_TYPE_GSM) :**s'il s'agit que du GSM.
- **TelephonyManager.PHONE_TYPE_NONE) : si** le type est inconnu**.**

L'IMEI du téléphone et sa version sont obtenus via les méthodes :

- **telephonyManager.getDeviceId () :** pour L'IMEI.
- **telephonyManager.getDeviceSoftwareVersion () :** c'est pour la version du téléphone.

La lecture de chacune de ces propriétés exige que la permission READ_PHONE_STATE doive être incluse dans le fichier Manifest :

<uses-permission android: name = "android.permission.READ_PHONE_STATE" />.

Vous pouvez également déterminer le type de réseau auquel vous êtes connecté, ainsi que le nom et le pays de la carte SIM ou le réseau de l'opérateur connecté.

III.3.2.1 La lecture des Détails Réseau

Lorsque votre appareil est connecté à un réseau, vous pouvez utiliser le Gestionnaire de téléphonie pour lire :

-Le code mobile du pays (ISO) à l'aide de **getNetworkCountryIso ().**

-Le code du réseau mobile (MCC + MNC) à l'aide de **getNetworkOperator ().**

-Le nom de l'opérateur via **getNetworkOperatorName ().**

-Le type de réseau auquel vous êtes connecté à l'aide de la **getNetworkType ().**

Pour le type du réseau la TelephonyManager permet de les obtenir, les lignes suivantes montrent les différentes technologies utilisées :

NETWORK_TYPE_CDMA.
NETWORK_TYPE_EDGE.
NETWORK_TYPE_GPRS.
NETWORK_TYPE_HSDPA.
NETWORK_TYPE_HSPA.
NETWORK_TYPE_HSUPA.
NETWORK_TYPE_LTE.
NETWORK_TYPE_UMTS.
NETWORK_TYPE_UNKNOWN : c'est quand la technologie n'est pas reconnue par le périphérique.

Ces commandes ne fonctionnent que lorsque la MS est connectée à un réseau mobile et elles peuvent être fiable s'il s'agit d'un réseau CDMA. L'utilité de la méthode getPhoneType (), comme le montre l'extrait de code précédent sert à déterminer quel type de téléphone est utilisé.

III.3.2.2 Les détails de la carte SIM :

Avant de pouvoir lire ces détails, il faut s'assurer que la carte SIM est dans un état prêt. Il faut déterminer cela en utilisant la méthode getSimState () via la **TelephonyManager.**

Différents détails de la SIM :

SIM_STATE_ABSENT: SIM absente.

SIM_STATE_NETWORK_LOCKED: SIM bloquée.

SIM_STATE_PIN_REQUIRED: Code pin de la SIM.

SIM_STATE_PUK_REQUIRED: Code puck de la SIM.

SIM_STATE_UNKNOWN: Etat inconnu.

SIM_STATE_READY: Etat actif.

La méthode **getSimSerialNumber ()** permet d'obtenir le numéro de série de la carte SIM.

III.3.2.3 Surveillance des appels téléphoniques entrants :

Notre application peut intercepter les appels téléphoniques entrants seulement pendant qu'il est en cours d'exécution, vous pouvez remplacer. La méthode **onCallStateChanged ()** met en œuvre cet état d'écoute, et l'enregistre pour recevoir des notifications lors les changements d'état d'appel en implémentant ces paramètres :

- **CALL_STATE_IDLE :** Etat de repos (Pas d'appel).
- **CALL_STATE_OFFHOOK:**Etat d'appel.
- **CALL_STATE_RINGING :** Etat de sonnerie (Réception d'un appel), pour ce dernier nous pouvons y ajouter le variable **incomingNumber** pour obtenir le numéro appelant.

Les modifications apportées à l'état du téléphone sont surveillés en utilisant la classe PhoneStateListener, avec un certain état de changements diffusés comme Intentions.

Pour ce cas nous devons ajouter la méthode**telephonyManager.listen(callStateListener, PhoneStateListener.LISTEN_CALL_STATE)** qui se met en écoute au court du temps.

III.3.2.4 L'écoute de connexion de données et les détails de l'état de transfert :

En utilisant les méthodes **getDataState ()** et **getDataActivity ()**, vous pouvez trouver la connexion de données en courset l'activité de transfert de données, respectivement :

La méthode **getDataActivity()** permet d'obtenir les informations suivantes:

DATA_ACTIVITY_IN : Données entrants (voie descendante).

DATA_ACTIVITY_OUT : Données sortantes (voie montante).

DATA_ACTIVITY_INOUT : Les deux sens.

DATA_ACTIVITY_NONE : Pas d'activité.

Ainsi que **getDataState ()** permet d'avoir les états suivants :

DATA_CONNECTED : Connexion active.

DATA_CONNECTING : Transfert de données.

DATA_DISCONNECTED: Déconnexion.

DATA_SUSPENDED : Données suspendues

III.3.2.5 L'écoute des données de signalisation cellulaire :

En se servire de la classe **ConnectivityManager** et son service **ConnectivityManager** et son service **CONNECTIVITY_SERVICE** nous pouvons connaître le type de connexion lors l'utilisateur est connecté via le Wi-Fi ou les services des opérateurs comme la 3G par exemple, juste en appelant ces méthodes suivantes :

Pour le Wi-Fi :

- **getActiveNetworkInfo().getTypeName().equals("WIFI").**

Pour les services de l'opérateur:

- **getActiveNetworkInfo().getTypeName().equals("MOBILE").**

La classe **CellLocation** comprend des méthodes pour extraire les différentes informations d'emplacement de l'utilisateur en fonction du type réseau cellulaire. Dans le cas du GSM la classe **GsmCellLlocation**qui est une instance de la classe principale **CellLocation**comporte plusieurs méthodes comme :

getCid() : l'ID de la cellule où se situe l'utilisateur.

getLac() :l'ID de la zone de localisation de l'utilisateur.

getPSC : c'est la norme de débit GSM qui se situe entre (-1) et (-6) allant de 22.8 Kbits/s jusqu'à ? Respectivement.

IsNetworkRoaming () : vérifie l'état de Roaming s'il est activé ou non.

getNetworkType () : Elle s'agit du type de réseau si c'est 1 le réseau est de type GSM, si c'est 2 le type est la CDMA.

Pour la puissance de signal nous faisons l'appel à la classe **NeghiboringCellInfo()** qui doitêtre implémentée sous forme d'une liste en utilisant la méthode**getNeghboringCellInfo()** qui donne l'accès à récupérer la puissance de signal que l'utilisateur reçoit de la part de sa cellule via la méthode **getRssi ()**.

Avec la même classe nous pouvons obtenir les mêmes informations précédentes mais des cinq autres cellules voisines que le périphérique peut lire en introduisant une boucle lit toute la liste précédente et en ajoutant la méthode **get(i)** avant chacune des méthodes précédentes.
Par exemple : **NeighboringList.get(i).getCid()**permet d'obtenir les cinq IDs de cellules voisines.

Ça reste le cas de la CDMA s'il marche ok sinon je le pose comme problème ou remarque.

III.3.3 Gestion du réseau et la connectivité internet :

La ConnectivityManager représente le service de connectivité réseau. Elle est utilisée pour surveiller l'état des connexions réseau, configurer les paramètres de basculement, et de contrôler les radios du réseau.
Pour utiliser le gestionnaire de connectivité, nous avons besoin d'ajouter ces deux permissions dans le Manifest pour que l'application lise et écrive l'accès de l'état du réseau :

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>.

La classe WifiManager, qui représente le service de la connectivitéWi-Fi, peut être utilisée pour configurer les connexions Wi-Fi, gérer la connexion actuelle, rechercher les points d'accès,et surveiller les changements dans la connectivité Wi-Fi.
Pour utiliser le Gestionnaire Wi-Fi, la demande doit avoir des permissions d'accès et du changement de l'état du Wi-Fi incluses dans le Manifest toujours :

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>.

Il existe plusieurs méthodes pour écouter les paramètres Wi-Fi :
-getSSID () : obtenir le nom SSID de notre connexion Wi-Fi.
-getBssid () : récupérer l'adresse physique du point d'accès.
-getFrequency () : donne la fréquence utilisée lors la connexion.
-getLinkSpeed () : envoie le débit.
-getRSSI() : c'est pour la puissance Wi-Fi.

-getCapabilities () : elle sert à détecter le type de sécurité et la cryptographie utilisés.

Les méthodes **getIpAddress()** et **getMacAddress ()** donnent les adresses IP et l'adresse physique du périphérique respectivement.

Nous avons aussi fait un programme de scan Wi-Fi, il permet de scanner tous les points d'accès détectés et donne les mêmes informations précédentes pour tous ces points. Nous avons utilisés la classe **WifiReceiver** et la méthode :

SCAN_RESULTS8AVAILABLE_ACTIONpassant par le service **WifiManager**, les implémentés dans une liste et récupérer les informations grâce à la méthode **getScannResults(),** la ligne suivante décrit la liste utilisée :

WifiList= **mainWifi**.**getScanResults ()** ;

III.4 Conclusion :

Comme nous avons dit à la fin du deuxième chapitre, nous avons visé les librairies Android qui servent à exploiter les paramètres et les informations dans différents services comme la géolocalisation, la téléphonie, les paramètres radio et de signalisation ainsi que le Wi-Fi. Dans le chapitre suivant nous allons décrire l'application finale. En effet, tous au long plusieurs versions ont été développées en se focalisant sur les notions abordées dans ce chapitre.

Contenu