ENSIAS

جامعة محمد الخامس بالرباط
Université Mohammed V de Rabat

Développement d'une application de téléphonie

ACHBAK NORA

Année Universitaire :2017/2018

Introduction

Afin d'appliquer les méthodologies enseignées dans le cours du développement mobile sous Android, un projet s'impose pour bien maitriser ce langage de programmation.

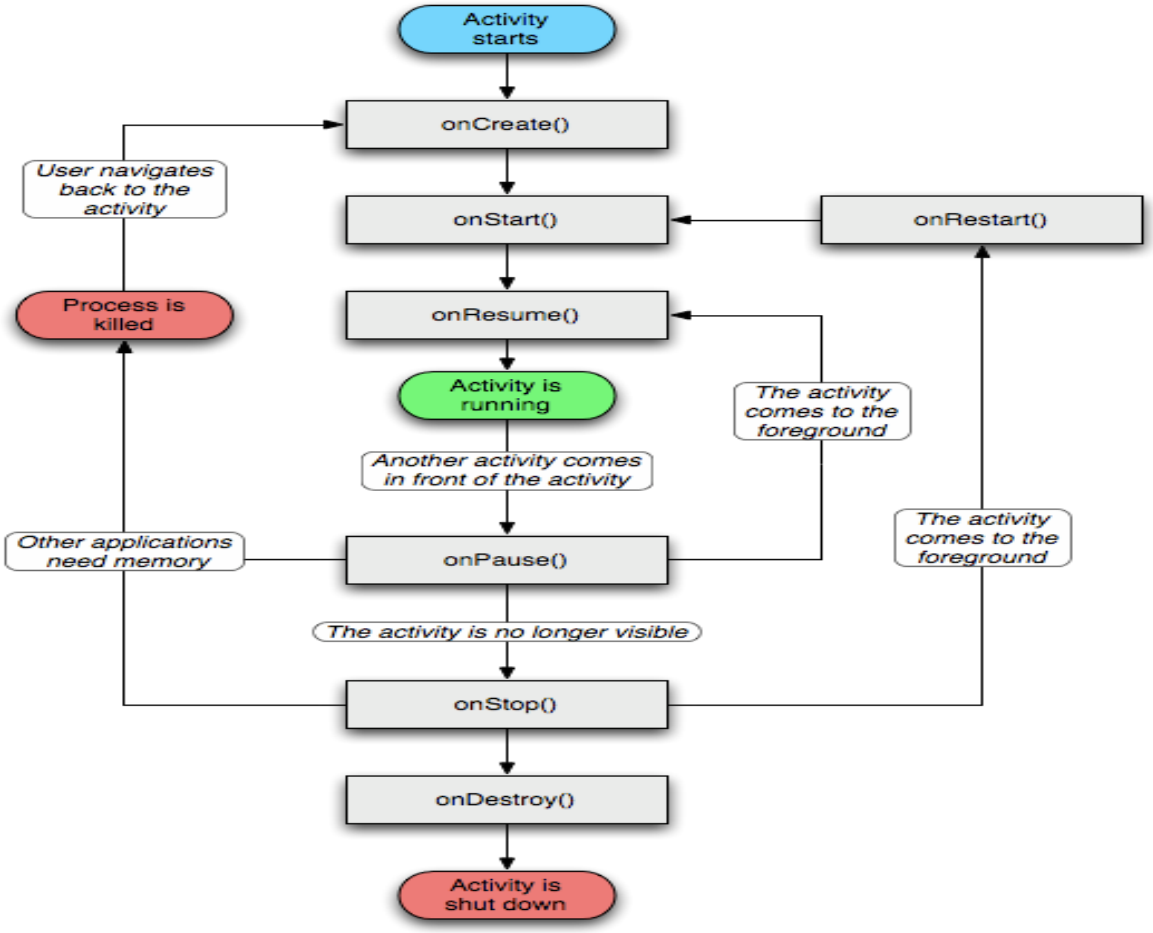
Le sujet de notre projet est la réalisation d'une application mobile dédiée à la gestion des appels téléphoniques. L'application doit être capable de garder l'historique des informations de toutes les appels téléphoniques à savoir le nom de l'appelant, le numéro de téléphone, la date de l'appel, la durée et le type de l'appel (entrant, sortant ou manqué).

Les informations des appels doivent être enregistrées dans une base de données et l'exporter par la suite dans une base de données distante via les web Services.

Ce rapport peut ainsi être subdivisé en trois parties. La première consistera à la présentation de développement mobile sous Android. La seconde partie sera consacrée à l'implémentation proprement dit de l'application dont on va détailler des parties du code. Enfin, la troisième et dernière partie sera réservée à présenter l'application avec les fonctionnalités de base.

Développement mobile

Les technologies mobiles prennent de plus en plus de place sur le marché. Les Smartphones sont considérés comme des petits ordinateurs et dotés d'un système d'exploitation s'appuyant sur un noyau Linux. Cependant, ils diffèrent des ordinateurs classiques par le cycle de vie d'une application. Sous Android, une application est composée d'une ou plusieurs activités. Une activité est la base d'un composant pour la création d'interfaces utilisateur. Afin de facilité la cinématique de l'application, il est préconisé de n'avoir qu'une interface visuelle par activité.



- L'activité démarre : la méthode onCreate est appelée.

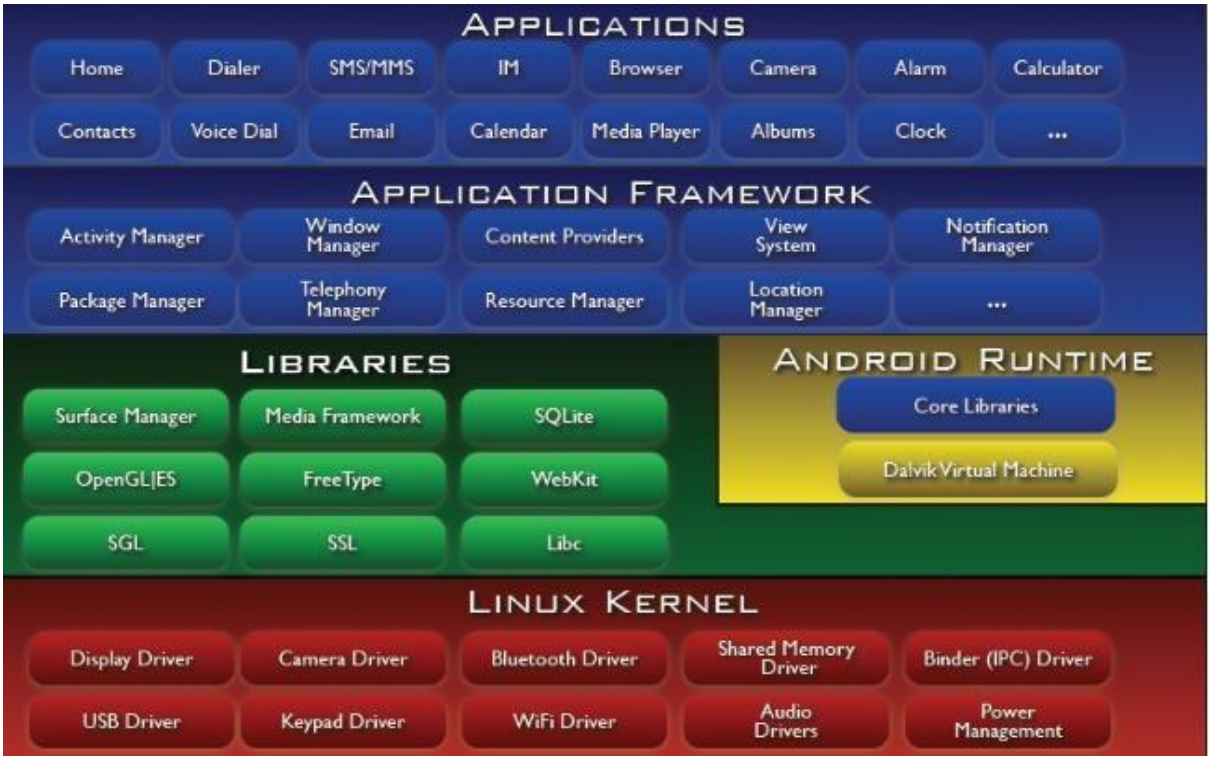
- Pendant l'utilisation d'une activité, l'utilisateur presse la touche Accueil, ou bien l'application téléphone, qualifiée comme prioritaire et qui interrompt son fonctionnement par un appel téléphonique entrant. L'activité est arrêtée (appel de onStop), le développeur détermine l'impact sur l'interface utilisateur, par exemple la mise en pause d'une animation puisque l'activité n'est plus visible.

- Une fois l'appel téléphonique terminé, le système réveille l'activité précédemment mise en pause (appel de onRestart, onStart).

- L'activité reste trop longtemps en pause, le système a besoin de mémoire, il détruit l'activité (appel de onDestroy).

- onPause et onResume rajoutent un état à l'activité, puisqu'ils interviennent dans le cas d'activités partiellement visibles, mais qui n'ont pas le focus. La méthode onPause implique également que la vie de cette application n'est plus une priorité pour le système. Donc si celui-ci a besoin de mémoire, l'Activity peut être fermée. Ainsi, il est préférable, lorsque l'on utilise cette méthode, de sauvegarder l'état de l'activité dans le cas où l'utilisateur souhaiterait y revenir avec la touche Accueil.

Android se base sur un noyau Linux 2.6. Le SDK Android possède une bibliothèque de librairie de plusieurs classes java de base pour plusieurs types d'application (exemple : OpenGL|ES pour la 3D, SSL pour les protocoles de sécurité, etc.). Une application Android se repose sur un Framework qui facilite l'utilisation des classes de base et sert d'interface entre les "Librairies" et les applications.



Android est un système d'exploitation pour téléphone portable de nouvelle génération développé par Google. Celui-ci met à disposition un kit de développement (SDK) basé sur le langage Java.

Nous avons utilisé pour le développement de notre application l'environnement Android studio.


Android Studio Setup Wizard
Welcome to Android Studio
Recent Projects
No Project Open Yet
Quick Start
Start a new Android Studio project
Open an existing Android Studio project
Import an Android code sample
VCS
Check out project from Version Control
Import project (Eclipse ADT, Gradle, etc.)
Configure
Docs and How-Tos
Android Studio 1.1.0 Build 135.1740770. Check for updates now.

L'implémentation

L'interface graphique

Android utilise des fichiers ressources xml : "main.xml", pour générer une interface graphique (ou view). Ce fichier est appelé par le fichier principal "main.java" grâce à la commande "setContentView(R.layout.main)". "R" est un fichier ressource de l'application contenant tous les identifiants des widgets (références vers les widgets). Il est généré automatiquement par le SDK.

Voila un exemple de fichier xml de l'activité main :

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:textStyle="bold|italic" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Journal d'appels"
            android:textSize="24sp"
            android:layout_gravity="center_horizontal"/>
        <ListView
            android:id="@+id/maListe1"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="#009ff5d9"></ListView>
    </LinearLayout>
</android.support.constraint.ConstraintLayout>
```

Les permissions

Dans un premier temps on a pu récupérer les informations nécessaires pour notre application à partir de l'application mère du téléphone dédiée à la gestion de l'historique des appels. Pour cela on a utilisé deux permissions READ_CALL_LOG et WRITE_CALL_LOG.

```
<uses-permission android:name="android.permission.READ_CALL_LOG" />
<uses-permission android:name="android.permission.WRITE_CALL_LOG" />
```

Ces deux permissions sont ajoutées au niveau du fichier « AndroidManifest.xml »pour permettre à l'application d'accéder à l'application mère de l'historique des appels sur le téléphone et récupérer les informations nécessaires ou bien agir sur l'application mère en demandant la suppression d'un appel par exemple.

Le tri

Il est demandé de faire le tri sur deux critères :la date des appels et la durée mais on a fait juste le tri sur la durée, car le tri selon la date est effectué automatiquement lorsqu'on récupère les données à partir de l'application d'origine.

Pour trier les appels par leurs durées on a utilisé une méthode de tri pour les listes offertes par la classe « Collections », il s'agit de la méthode « Collections.sort() » en implémentant l'interface « comparator ».

```
Collections.sort(i, (Comparator) (p1, p2) → {
        if(p1.duration.compareTo(p2.duration) == 0) {
            return p1.name.compareTo(p2.name);
        } else {
            return Integer.valueOf(p1.duration.compareTo(p2.duration));

        }
```

Si on a deux appels qui ont la même durée on compare leurs noms d'appelant, sinon on compare juste leurs durées.

Le filtrage

Pour les filtre on a opté comme méthode filtrer(), dans cette méthode on charge un ArrayList de type InfoCall. Le chargement se fait de la même façon que dans la méthode getCalDetails dans l'activité principale. Puis, à travers une condition « if » on verifie si le type de l'appel est celui du filtre. Prenons l'exemple des appels sortants :

```
for(int j=0;j<i.size();j++){
    if(i.get(j).type=="OUTGOING") i1.add(i.get(j));

}
```

"i" est une liste temporaire où on trouve tous les appels sous forme des instances de la classe InfoCall. Et i1 est la liste qui contient les appels après filtrage et celle qui va être affiche par l'adaptateur dans la ListView.

De la même façon le filtre des appels entrants et manqués.

Pour le filtre selon les appels du mois encours, on a une classe dédiée à ceci « MoisEncours ».

```
Calendar cal = Calendar.getInstance();
int m1=cal.get(Calendar.MONTH);
final Date callDayTime1 = new
Date(Long.valueOf(managedCursor.getString(date)));
if (callDayTime1.getMonth()==m1)    infoArrayList1.add(infos);
```

On obtient le mois courant à travers la classe "Calendar" qui offre le mois sous forme d'entier à l'aide de cal.get(Calendar.***MONTH***) et on l'a comparé avec celui de l'appel obtenu avec le curseur.

De même le filtre selon le numéro, on cherche dans la liste des appels tous qui ont le même numéro obtenu lors du clic (le clicListener). Ici la méthode filtrer a comme paramètre le numéro qu'elle va chercher.

Le menu

Pour ajouter un menu, on a ajouté un nouveau dossier parmi les layouts nommé « menu » ou on trouve un fichier xml où on trouve les options offertes « item », ceux qui seront affichés ont comme « showActions » always et les autres never.

```
<item android:title="Missed"
    android:id="@+id/Missed_id"
    app:showAsAction="never"/>
```

On instancie le menu dans l'activité à l'aide de la classe MenuInflater

```
MenuInflater inflater=getMenuInflater();
inflater.inflate(R.menu.main_menu,menu);
```

L'acces aux interfaces de ce menu se fait par **public boolean** onOptionsItemSelected (MenuItem item) et chaque nouvelle activité est obtenue avec un intent.

L'adaptateur

Afin d'afficher plusieurs informations sur l'appels dans la même case de la listview on a eu recours aux adaptateur personnalisés. En effet, on un fichier xml et une classe :Costom_row_view et MyCostumBaseAdapter.

Costom_row_view : un fichier xml où on a défini les champs de la case qui sont des textView.

MyCostumBaseAdapter : une classe qui hérite de la classe BaseAdapter. Elle a comme parametre un ArrayList de type infoCall est sur celui-ci qu'on va agir, et mInflater de type LayoutInflater. La classe LayoutInflater est utilisée pour instancier le fichier XML de mise en page dans ses objets View View correspondants.En d'autres termes, il prend comme entrée un fichier XML (Costom_row_view)et crée les objets View. Et la méthode **public** View getView(**int** position, View convertView, ViewGroup parent) lie entre les éléments de du fichier xml et ceux de la liste.

La méthode getCallDetails()

Pour pouvoir récupérer les détails des appels à partir de l'application mère du téléphone on a définit la méthode getCallDetails() qui implémente des méthodes offertes par la permission CallLog.

Dans un premier temps on a déclaré un curseur « managedCursor » auquel on a associé la méthode managedQuery.

```
final Cursor managedCursor = managedQuery(CallLog.Calls.CONTENT_URI, projection: null, selection: null, selectionArgs: null, sortOrder: null);
```

La méthode managedQuery permet de fournir un répertoire des appels à travers la méthode « Callog.Calls.CONTENT_URI » passée parmi ses arguments .

Par la suite on a déclaré des variables de type entier, et on les affecte comme valeur l'index de la colonne qui contient la donnée désirée à travers la méthode « getColumnIndex ».

Voilà un exemple qui renvoie l'index du numéro de téléphone :

```
int number = managedCursor.getColumnIndex(CallLog.Calls.NUMBER);
```

De la meme manière on récupere l'index du nom de l'appelant,l'identifient de l'appel,la date ,la durée et le type de l'appel.

Puis on a fait une boucle while sur toutes les lignes du résultat du curseur dans laquelle on a récuperer le resultat de chaque colonne sous forme d'une chaine de caractères à travers la méthode « getString ».

On a aussi intégré un switch dans cette boucle :

```
String dir = null;
int dirCode = Integer.parseInt(callType);
switch (dirCode) {
    case CallLog.Calls.OUTGOING_TYPE:
        dir = "OUTGOING";
        break;
    case CallLog.Calls.INCOMING_TYPE:
        dir = "INCOMING";
        break;
    case CallLog.Calls.MISSED_TYPE:
        dir="MISSED CALL" ;
        break;
}
```

On a converti callType qui est de type string en un entier par la
méthode « Integer.parseInt(callType) », puis on a fait le switch sur la valeur de cet entier pour pouvoir récupérer le type de l'appel.

Base données

-Il est demandé d'enregistrer les informations sur les appels (numérotel,durée d'appel,la date d'appel) dans une base de données, donc on a utilisé base de donnée offert par Android SQLite, après on a affiché les données dans un tableau SQLite est une base de données open source, qui supporte les fonctionnalités standards des bases de données relationnelles comme la syntaxe SQL, les transactions et les prepared statement. La base de données nécessite peu de mémoire lors de l'exécution (env. 250 ko), ce qui en fait un bon candidat pour être intégré dans d'autres environnements d'exécution SQLite prend en charge les types de données TEXT (similaire à String en Java), INTEGER (similaire à long en Java) et REAL (similaire à double en Java). Tous les autres types doivent être convertis en l'un de ces types avant d'être enregistrés dans la base de données. SQLite ne vérifie pas si les types des données insérées dans les colonnes correspondent au type défini, par exemple, vous pouvez écrire un nombre entier dans une colonne de type chaîne de caractères et vice versa.

Donc On a créé premièrement la classe sqliteOpenHelper qui contient deux méthodes oncreate() qui est utilisé pour accéder à une base de donnée qui n'a pas encore créé onupgrade() est appelée si la

version de la base de données est augmentée dans le code de votre application. Cette méthode vous permet de mettre à jour un schéma de base de données existant ou de supprimer la base de données existante et la recréer par la méthode onCreate()

```
import ...

public class MySqlOpenHelper extends SQLiteOpenHelper {
    public MySqlOpenHelper(Context context) { super(context, "listInfo.db", null, 1); }

    @Override
    public void onCreate(SQLiteDatabase db) {

        String maRequete = "create table infosAppel (id text not null,numTel text not null, date text not null, duree text not null)
        db.execSQL(maRequete);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP table infosAppel ;");
        onCreate(db);

    }
}
```

Main activity

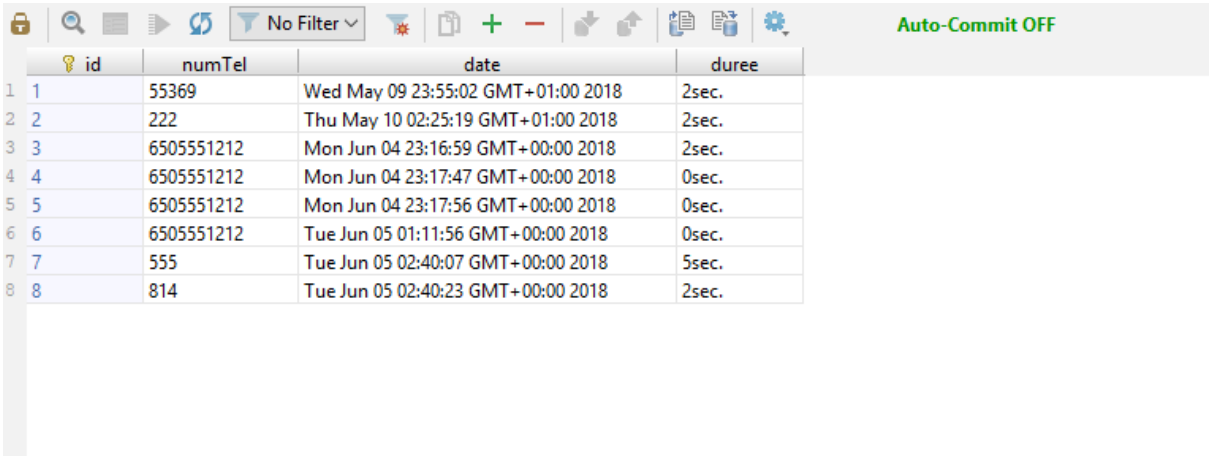
```
////////// base donnée

        MySqlOpenHelper dbHelper = new MySqlOpenHelper(getApplicationContext());
        SQLiteDatabase db =dbHelper.getWritableDatabase();
        ContentValues values =new ContentValues();
        values.put("numTel",managedCursor.getString(number));
        values.put("date",callDayTime.toString());
        values.put("duree",calculateTime(managedCursor.getString(duration)));

        long b = db.insert( table: "infosAppel", nullColumnHack: null,values);
        if(b==-1){
            Toast.makeText( context: this,  text: "nono", Toast.LENGTH_SHORT).show();
        }
        else {
            Toast.makeText( context: this,  text: "yes", Toast.LENGTH_SHORT).show();

        }
        dbHelper.close();
```

Résultat

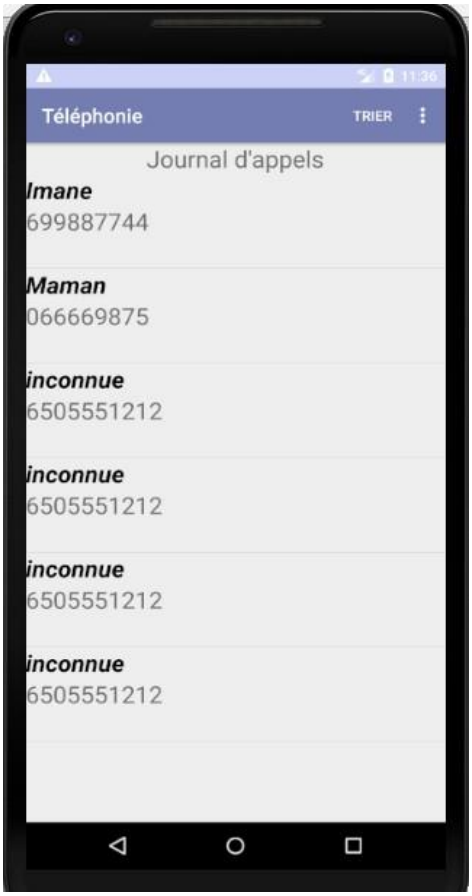


	id	numTel	date	duree
1	1	55369	Wed May 09 23:55:02 GMT+01:00 2018	2sec.
2	2	222	Thu May 10 02:25:19 GMT+01:00 2018	2sec.
3	3	6505551212	Mon Jun 04 23:16:59 GMT+00:00 2018	2sec.
4	4	6505551212	Mon Jun 04 23:17:47 GMT+00:00 2018	0sec.
5	5	6505551212	Mon Jun 04 23:17:56 GMT+00:00 2018	0sec.
6	6	6505551212	Tue Jun 05 01:11:56 GMT+00:00 2018	0sec.
7	7	555	Tue Jun 05 02:40:07 GMT+00:00 2018	5sec.
8	8	814	Tue Jun 05 02:40:23 GMT+00:00 2018	2sec.

Présentation de l'application

Le but de cette application est de mettre en place une application sous Android dédiée à la traçabilité des appels téléphoniques.

Le point principal est l'affichage des appels réalisés sous forme d'une liste, où chaque case de la liste on a mis le nom et le numéro de l'appel.



l'activité principale

Comme on peut remarquer, l'application est nommée « Téléphonie »et le menu est constitué du deux champs « Trier » et un ensemble groupé d'options. Ainsi que le journal d'appels. Si l'appel est celui d'un contact déjà enregistré dans le carnet des contacts le nom est affiché sinon on affiche « inconnue ».

menu de l'activité principale

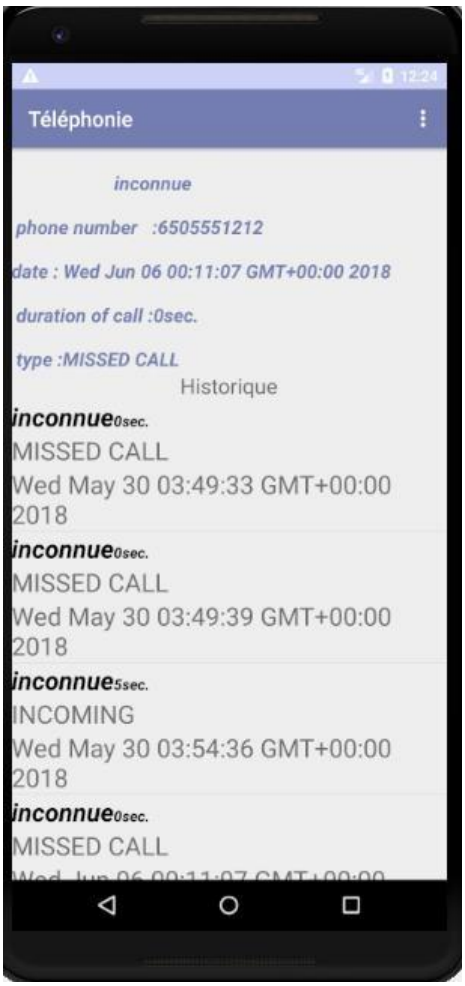
On offre aussi la possibilité d'avoir plus d'informations sur chaque appel. Avec un simple clic sur un appel une nouvelle activité s'ouvre en offrant plus de détails.

Dans cette activité on les informations de l'appel à travers le nom, le numéro, la durée de l'appel et la date et le type d'appel : entrant, sortant ou manqué.

En bas, on a un historique d'appels émis avec cette contact.

Pour ce qui concerne la suppression d'élément de la liste, il est faite par un clic enfoncé.

Le tri est fait selon la durée des appels.

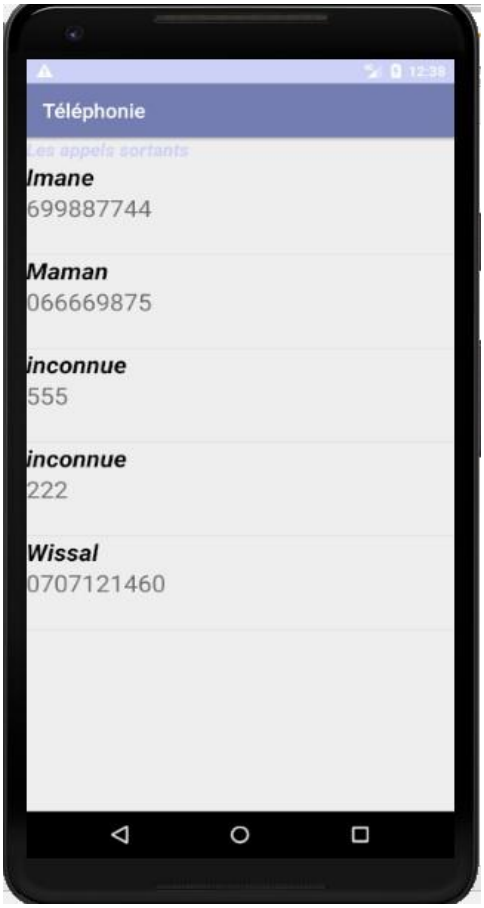


Résultat du tri

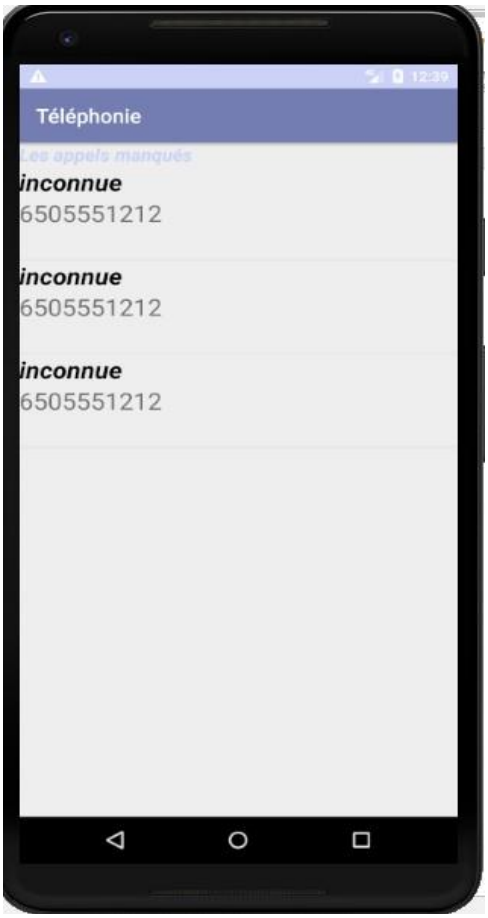
Pour les options du filtrage on a un filtrage selon le type, les appels du mois encours ainsi que selon le numéro.

Pour les appels selon le numéro ils sont affiché avec les détails supplémentaire lors du clic simple.

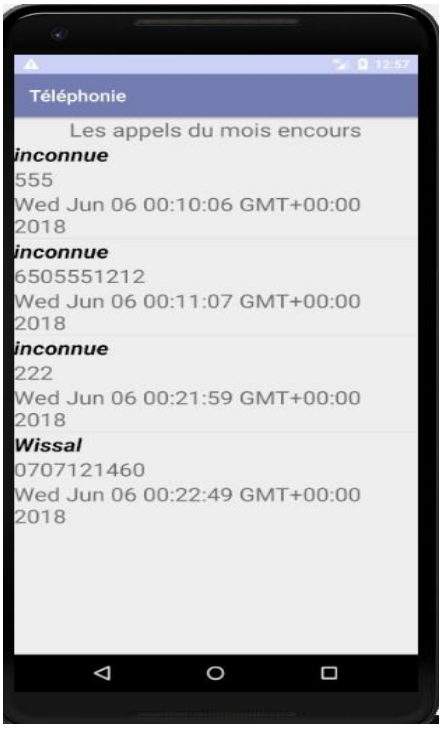
Pour le filtrage par type, pour chacun s'affiche une activité indépendante où il y a uniquement les appels ce même type.



filtre selon le type « sortants"



filtre selon le type « manqué »



filtre selon les appels du mois encours

Conclusion

L’objectif de notre projet était la réalisation d’une application mobile dédiée à la traçabilité des appels téléphoniques. Le travail était divisé en trois parties : une première partie pour la récupération des données, une deuxième l’enregistrement des données et la troisième pour les opérations sur les appels.