L'objectif de ce tutoriel est de vous présenter une des façons d'exploiter en lecture des informations distantes disponibles au format XML. L'application présentera le résultat dans une ListView

1/ Prérequis

Savoir utiliser une ListView : faire le tutoriel suivant

- http://developers.androidcn.com/resources/tutorials/views/hello-listview.html

`Remplacer dans ce tutoriel le new OnItemClickListener() par une implémentation de l'interface OnItemClickListener.`

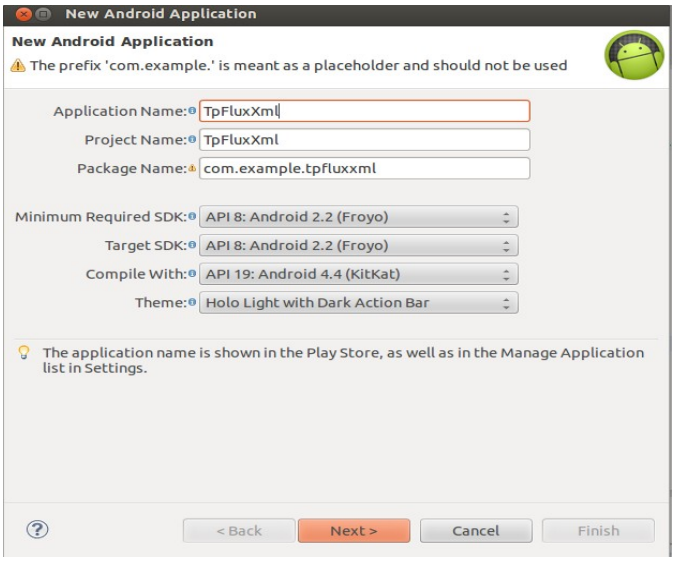
Disposer d'une connexion internet permettant l'accès à cette ressource distante.
Pour cela vous pouvez télécharger cette ressource :
`http://guyonst.free.fr/android/jukebox.tar.gz`
Une fois décompresser dans votre [home directory]/public_html , vous accéder à cette ressource

http://localhost/~sio/jukebox/albums-by-genre.php

L'application utilise SQLITE et non MYSQL, si vous avez un message d'erreur vous devez installer le driver sqlite, pour l'installer , connectez vous en root :
aptitude install php5-sqlite
#/etc/init.d/apache2 restart (relancer apache)
=> vérifier à l'aide d'un navigateur, et observer le source de la réponse.

2/ Projet

1. Lancez le logiciel Eclipse et un émulateur Android
2. Créer un nouveau projet Android : **`TpFluxXml.`**



On utilisera comme cible (Target) : Android 2.2 dans un objectif purement pédagogique. En effet à partir de la version 4 la lecture d'un flux distant est devenu beaucoup plus complexe comme le verrez dans la suite du TP. L'objectif de la première partie de ce TP étant de comprendre comment lire et ensuite “parser” un flux distant Xml.

L'application devant accéder à une ressource distante, nous devons signaler la nécessité d'utiliser une connexion Internet. Ceci se déclare dans le fichier AndroidManifest.xml placé à la racine de l'application :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.ldv.tpxml"
      android:versionCode="1"
      android:versionName="1.0">

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

    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon"
                 android:label="@string/app_name">

        <activity android:name=".Main"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

Il est parfois nécessaire de réaliser un clean (`Project/Clean`) pour permettre la suppression des versions compilées des différentes sources du projet et forcer de nouveau leur compilation.

Lancer l'application. Vous devriez voir dans la console :

- Le nom de l'activité qui est sera créée au démarrage : **`org.ldv.tpxml.Main`**
- Le nom du fichier de déploiement : **`tpxml.apk`**
- Le nom de l'émulateur sur lequel l'application packagée est déployée.

3/ Document XML et Classe métier

Notre application exploite des données structurées.

Extrait du document XML reçu :

```
<albums genre='Rock'>

  <album>
    <artist>U2</artist>
    <title>Sunday Bloody Sunday</title>
    <year>1983</year>
  </<album>

  etc.

</albums>
```

La structure d'un album sera déclarée par une Classe métier nommée `Album` :

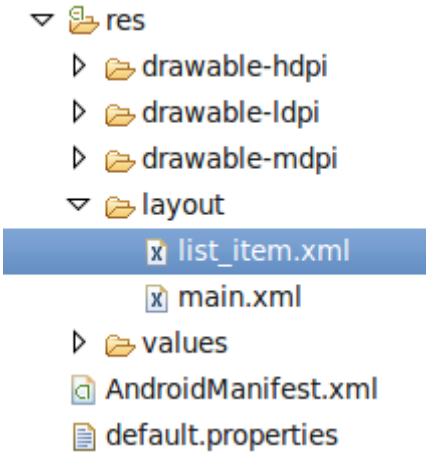
```
package org.ldv.tpxml;

public class Album {
	private String artist;
	private int year;
	private String title;


	@Override
	public String toString() {
		return  title;
	}

	public String getArtist() {
		return artist;
	}

	public void setArtist(String artist) {
		this.artist = artist;
	}

  etc.
```

4/ Classe (ou activité) principale

Pour pouvoir utiliser avec plus de facilité le composant widget d'Andoid `ListView`, nous avons fait hériter la classe Main non pas de `Activity` mais de `ListActivity` :

```
public class Main extends ListActivity
```

Le type (TextView) des éléments de la liste sera renseigné dans un fichier `list_item.xml`. Ce fichier est placé dans le répertoire `layout` lui-même sous-répertoire de `res` comme le montre cette arborescence :



Contenu du fichier list_item.xml :

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:textSize="14sp" >
</TextView>
```

Sont renseignés ici le type (TextView) mais aussi les paramètres utiles à leur affichage.

Ces informations sont référencées dans notre projet par un identifiant public placé dans une classe statique générée par le système (à ne pas modifier !) : `R.layout.list_item`

```
// dans la méthode onCreate
 this.setListAdapter(new ArrayAdapter<Album>(this, R.layout.list_item, albums));
```

Pour lier la liste d'objets `Album` à la `ListView`, on utilise un objet `ArrayAdapter` qui se charge d'adapter la liste d'objets `Album` au besoin du composant utilisé (`ListActivity`). On distingue bien ici le modèle (`albums`) de la vue (`TextView`).

Notre application conservera en mémoire la structure de l'arbre sous la forme d'une collection. Pour cela nous parcourons le document XML afin d'ajouter, un par un, les éléments compatibles avec la déclaration de la collection (ici un objet de type `List` contenant des objets de type `Album`). Nous utilisons une solution clé en main offerte par l'API `XmlPullParser` disponible avec le framework Android.

Voici la partie déclaration des constantes et attributs d'instances de la classe `Main`.

```
public class MainActivity extends ListActivity implements OnItemClickListener
{
      // on renseigne l'adresse de la machine host
      private static final String URL_XMLRESSOURCE =
            "http://xxx/albums-by-genre.php";
      // 10.0.2.2 (pour atteindre le localhost à partir de l'émulateur)
      // "http://10.0.2.2/~xx/xxx.php";

      // Le nom des balises du doc XML reçu
      private static final String ALBUMS = "albums";
      private static final String ALBUM = "album";
      private static final String ARTIST = "artist";
      private static final String TITLE = "title";

      // la liste des albums en mémoire (petite liste)
      private List<Album> albums;
```

Voici la méthode appelée lors de la création de l'instance de l'activité, fortement inspirée du tutoriel `Hello List View` : http://developers.androidcn.com/resources/tutorials/views/hello-listview.html

```
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);

      // on obtient la liste des albums en appelant la méthode parse
      albums = parse(URL_XMLRESSOURCE) ;

      // on initialise la ListView (comme dans le tuto Hello List View)
      setListAdapter(new ArrayAdapter<Album>(this, R.layout.list_item,
albums));

      ListView lv = getListView();
      lv.setTextFilterEnabled(true);
      lv.setOnItemClickListener(this);

     }
```

La classe Main implémente l'interface **`OnItemClickListener`** , lorsqu'on implémente une interface , on doit implémenter toutes les méthodes de cette interface, dans ce cas la méthode :

```
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long arg3)
```

arg2 : représente la position de l'item sélectionné dans la liste

```
@Override

      public void onItemClick(AdapterView<?> arg0, View arg1, int arg2, long
arg3) {
      Toast.makeText(getApplicationContext(),"L'index de cet élément est : " +
      arg2, Toast.LENGTH_SHORT).show();

      }
```

La méthode parse ci-dessous s'inspire fortement sur l'exemple de la documentation de l'API `XmlPullParser` : http://developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html

Son objectif est d'intégrer les données de la ressource distante dans l'application.

```
/**
 * Obtient une liste d'objet Album à partir d'une ressource XML
 * @param strUrl l'adresse de la ressource
 * @return liste d'objets Album (peut être vide)
 */
 private List<Album> parse(String strUrl) {
   List<Album> listeAlbums = new ArrayList<Album>();
   XmlPullParser parser = Xml.newPullParser();
   try {

     // auto-detect the encoding from the stream
     parser.setInput(this.getInputStream(strUrl), null);

     int eventType = parser.getEventType();
     Album currentAlbum = null;
     boolean done = false;
     while (eventType != XmlPullParser.END_DOCUMENT && !done) {
       String name = null;
       switch (eventType) {
       case XmlPullParser.START_TAG:
         name = parser.getName();
         if (name.equalsIgnoreCase(ALBUM)) {
           currentAlbum = new Album();
           Log.i("tag1 : ", "creation album");
         } else if (currentAlbum != null) {
           if (name.equalsIgnoreCase(ARTIST)) {
             currentAlbum.setArtist(parser.nextText());
           } else if (name.equalsIgnoreCase(TITLE)) {
             currentAlbum.setTitle(parser.nextText());
           }
           // etc.
         }
         break;
       case XmlPullParser.END_TAG:
         name = parser.getName();
         // fin de balise Album ?
         if (name.equalsIgnoreCase(ALBUM) && currentAlbum != null) {
           Log.i("ajout album : ", currentAlbum.toString());
           listeAlbums.add(currentAlbum);
         }
         // fin de balise Albums ?
         else if (name.equalsIgnoreCase(ALBUMS)) {
           // oui, on signe l'arrêt de la boucle
           done = true;
         }
         break;
       }
       // avance dans le flux
       eventType = parser.next();
     }
   } catch (Exception e) {
     throw new RuntimeException(e);
   }
   return listeAlbums;
 }
```

Il ne nous reste qu'à présenter la méthode **`getInputStream`** utilisée par notre méthode **`parse.`**

```
private InputStream getInputStream(String strUrl) {
    InputStream res = null;
    try {
      URL url = new URL(strUrl);
      res = url.openConnection().getInputStream();
    } catch (Exception e) {
       Log.d("Erreur.", e.getMessage());
    }
    if (res == null)
      res = new ByteArrayInputStream("<albums><album><title>Vide!
</title></album></albums>".getBytes());
    return res;
  }
```

À partir d'une URL donnée (reçue en paramètre) sous la forme d'une chaîne de caractères, nous instancions la classe URL (standard du JDK) nous permettant de réaliser une connexion réseau (protocole HTTP par défaut). L'astuce consiste ici à retourner toujours une donnée valide (une instance qui implémente l'interface InputStream), que l'URL soit bien formée ou non, contenant une ressource ou non.

TRAVAIL À FAIRE

1. Quel est l'objectif de cette instruction :
```
List<Album> listeAlbums = new ArrayList<Album>();
```

2. Que signifie « parser » un fichier XML?

3. Que représente les « tags » ? Comment est structuré un document XML ?

4. Quel est « l'élement ROOT » et les éléments enfants de votre document XML ?

5. Quel est le type de la valeur retournée par les méthodes parse et getInputStream ?

6. Quelles lignes de codes font appel à ces méthodes ?

7. Quel est le type de la valeur retournée par la méthode `Toast.`makeText(...) ?

8. Pourquoi la méthode *parse* est-elle déclarée **private** ?

9. A quoi correspond cette instruction : name = parser.getName() ;

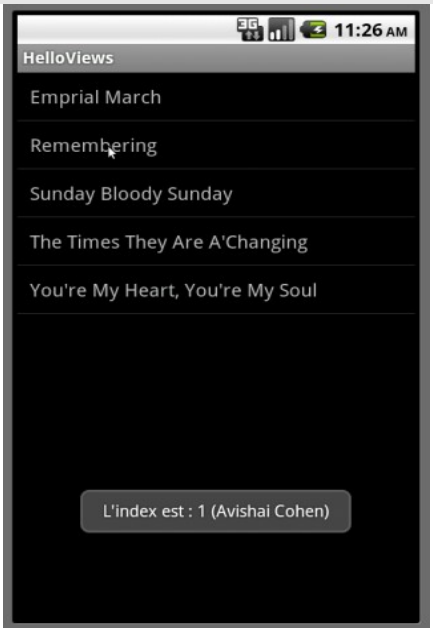
10. Que font les instructions suivantes :
```
if (name.equalsIgnoreCase(ALBUM)) {
                currentAlbum = new Album();
```
    Suite à ces instructions, quel est le titre de l'album ?
    Quelle est l'instruction qui permet de donner un titre à cet album?

11. Quel est l'intérêt de la variable `done` ?

12. Quelles informations sont « loguées » dans la boucle ?

13. Pourquoi la logique de la classe Main utilise des constantes, par exemple `ALBUM` ?

14. (programmation) Après avoir complété le code, vous lancerez l'application (compilation, déploiement et exécution), le résultat devrait être proche de celui-ci (ne pas tenir compte du message Toast):



15. (programmation) Modifier le code afin que le message *Toast* renseigne le nom de l'artiste de l'album sélectionné (voir copie écran ci-dessus).
16. Remplacer le Toast par `AlertDialog`

Plus dur : essayer d'afficher dans l'`AlertDialog`, tous les titres d'un même artiste.

Voilà, vous avez maintenant les connaissances de bases pour vous lancer dans de petits projets Android !

ANDROID 4

Lors du développement d'une application, il faut bien avoir en tête que toutes les tâches consommatrices de ressources (requêtes http, calculs lourds, …) doivent désormais se faire dans un Thread (processus) séparé. En effet, le système affiche un message d'erreur et ferme l'application lorsque le Thread principal (appelé **UI Thread**) est bloqué trop longtemps.

Le langage Java autorise la définition d'une classe à l'intérieur d'une autre classe :

```
class Englobante {
        int a;
        class Englobee{
                . . .
                Englobante.this.a = 12
        }
        . . .
}
```

Un des intérêts de cette notation est qu'une classe interne peut avoir accès aux méthodes et attributs de la classe englobante.

C'est la raison pour laquelle nous allons « englobée » notre classe TraitementTache dans notre classe principale :

```
public class TraitementTache extends AsyncTask<String, Void, ArrayList<Album>
```

Notre classe hérite d'AsyncTask, Les trois paramètres attendus lors de la déclaration sont des types génériques dont voici la signification :

- Le premier est le type des paramètres fournis à la tâche
- Le second est le type de données transmises durant la progression du traitement
- Enfin le troisième est le type du résultat de la tâche

Dans notre exemple, seul le troisième paramètre sera modifié, en effet on souhaite que notre classe retourne un ArrayList d'objet Album.

Une AsyncTask doit obligatoirement implémenter la méthode **doInBackground**. C'est elle qui réalisera le traitement de manière asynchrone dans un Thread séparé. Les méthodes **onPreExecute** (appelée avant le traitement), **onProgressUpdate** (appelée lorsque vous souhaitez afficher sa progression) et **onPostExecute** (appelée après le traitement) sont optionnelles.

Nous allons donc effectuer la lecture et le traitement de notre flux XML distant dans la méthode **doInBackground.** Lorsque le traitement sera terminé la méthode **onPostExecute** sera appelée, cette méthode se chargera de l'affichage de la liste.

```
public class TraitementTache extends AsyncTask<String, Void, ArrayList<Album>>  {
        private static final String URL_XMLRESSOURCE = "http://10.0.2.2/~stephane/xxx/xxx" ;
        private static final String ALBUMS = "albums";
        private static final String ALBUM = "album";
        private static final String ARTIST = "artist" ;
        private static final String TITLE = "title";
        // la liste des albums en mémoire (petite liste)
        private ArrayList<Album> albums;

        @Override
        protected ArrayList<Album> doInBackground(String... params) {
                // on obtient la liste des albums en appelant la méthode parse
                albums = parse(URL_XMLRESSOURCE) ;
                return albums;
          }

        private ArrayList<Album> parse(String strUrl) {
        idem android 2}

        private InputStream getInputStream(String strUrl) {
        idem android2 }
```

```
@Override
protected void onPostExecute(final ArrayList<Album> albums)          {
        setListAdapter(new ArrayAdapter<Album>(MainActivity.this, R.layout.list_item, albums));

        ListView lv = getListView();
        lv.setTextFilterEnabled(true) ;
        lv.setOnItemClickListener(new OnItemClickListener() {

                public void onItemClick(AdapterView<?> parent, View view, int position,long id) {
                        Toast.makeText(getApplicationContext(),"L'index de cet élément est : " +
                        albums.get(position), Toast.LENGTH_SHORT).show();
                }
          });
}
```

Notre classe TraitementTache sera instanciée dans notre classe principale :

```
public class MainActivity extends ListActivity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                new TraitementTache().execute();
        }
```

Le résultat final :

```
public class MainActivity extends ListActivity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                new TraitementTache().execute();
        }

        public class TraitementTache extends AsyncTask<String, Void, ArrayList<Album>> {

                @Override
                protected ArrayList<Album> doInBackground(String... params) {
                        // TODO
                }

                private ArrayList<Album> parse(String strUrl) {
                        //TODO
                }

                private InputStream getInputStream(String strUrl) {
                        //TODO
                        }
                @Override
                protected void onPostExecute(final ArrayList<Album> albums){
                        //TODO
                }

          }
}
```

Projet fin d'année :

Reprendre votre PPE, vous devez avoir une Table Contact (ou autre) et une Table Loisirs (ou autre) et une Table de liason (LoisirsContacts par exemple).

1 - A partir de la table Contact générer un fichier XML, dont vous définirez la structure.

2 – Afficher vos contacts dans une ListView Android (la version 4 d'Android serait appréciable)

3 – La ListView n'affichera que les informations sommaires du contact (nom, prénom etc...) . Lorsque l'on cliquera sur un contact des informations plus complètes devront s'afficher dans une AlertDialog.

4 – Prévoir également d'afficher les loisirs du contact.

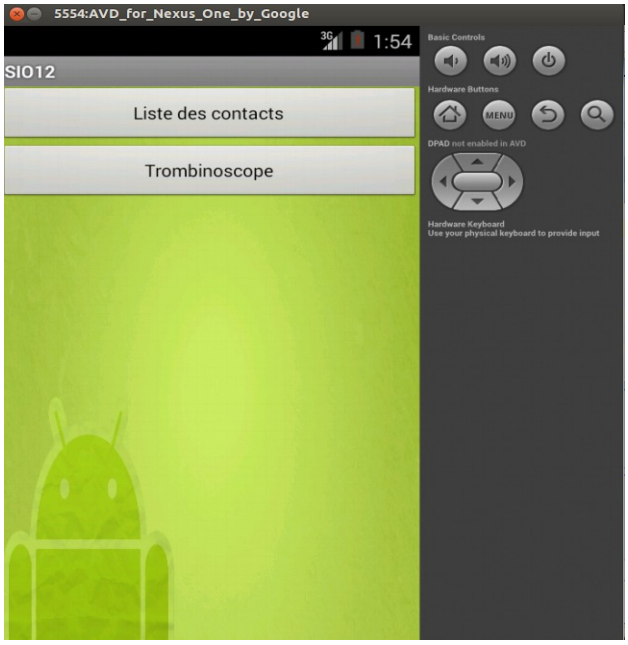
4 – Envisager une évolution de votre activité, comme par exemple une authentification, la possibilité de mettre à jour, d'ajouter un contact, présenter dans la ListView une photo du contact etc.....

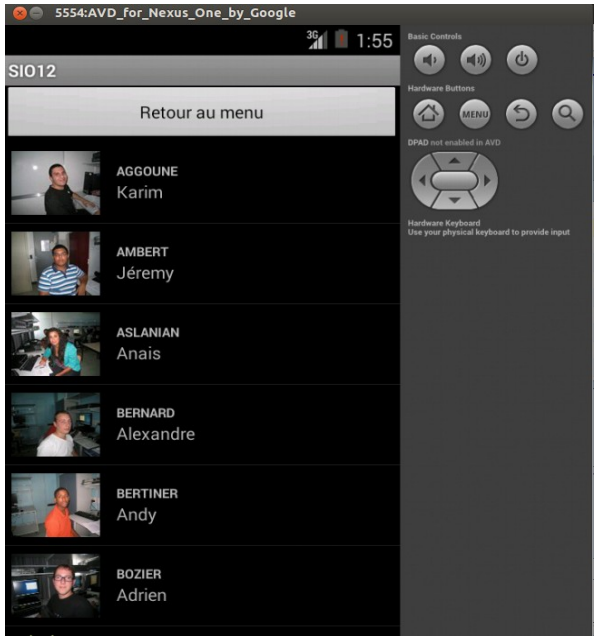
A déposer sur moodle le jour de la rentrée (28 avril) :

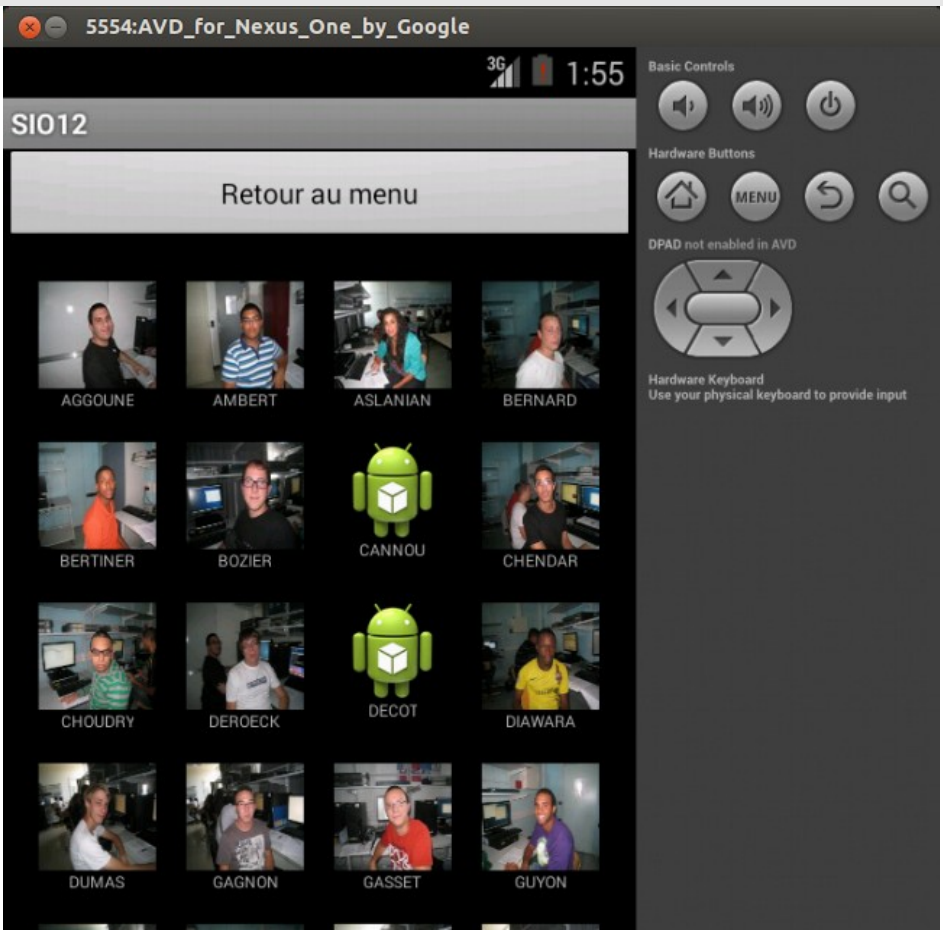
Une archive contenant :

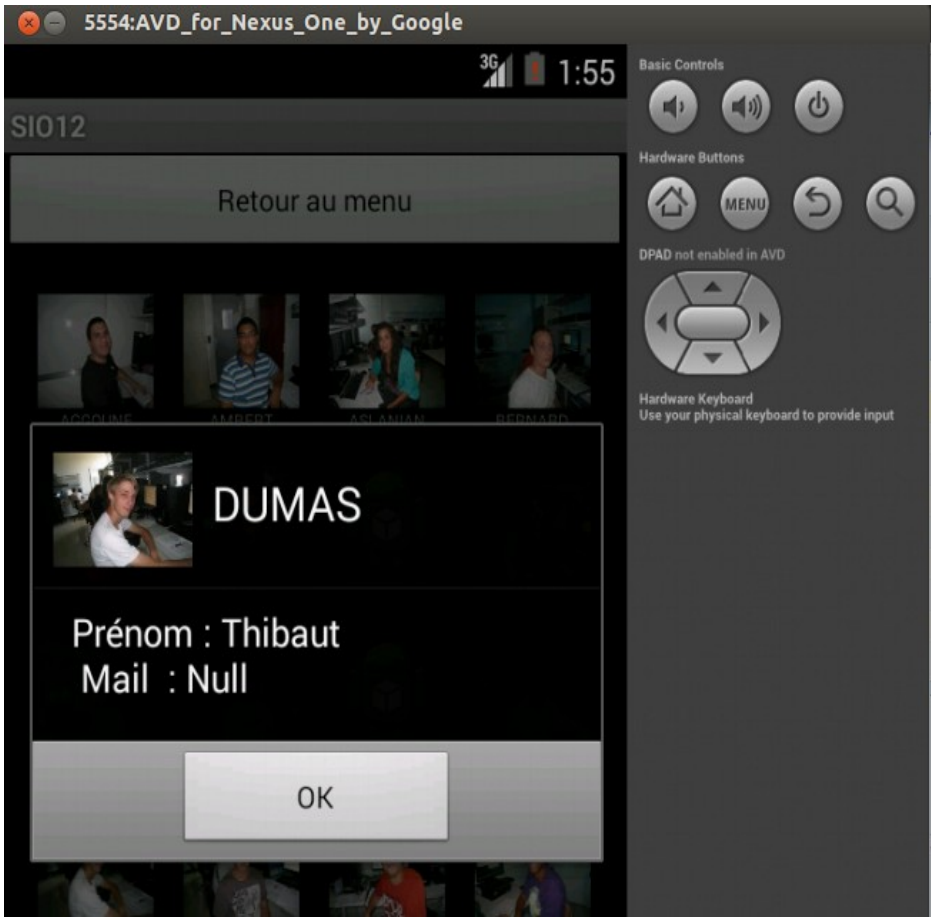
- les réponses aux questions 1 à 13 du TP
- la structure de votre fichier XML
- le code PHP permettant de générer ce fichier
- le code Android de votre application

Exemples de réalisations possibles :






5554:AVD_for_Nexus_One_by_Google
1:55
SIO12
Retour au menu
AGGOUNE
AMBERT
ASLANIAN
BERNARD
BERTINER
BOZIER
CANNOU
CHENDAR
CHOUDRY
DEROECK
DECOT
DIAWARA
DUMAS
GAGNON
GASSET
GUYON
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input


5554:AVD_for_Nexus_One_by_Google
1:55
SIO12
Retour au menu
DUMAS
Prénom : Thibaut
Mail : Null
OK
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input