Cas EDF : Développement Android - Concepts avancés – Partie 5

Cette publication comporte cinq parties dont l'ordre est dicté par la logique du développement.
Les parties 2 et 3 sont facultatives.

Partie 1 : Gestion des clients
Partie 2 : Géolocalisation de l'agent et géocodage du client sélectionné
Partie 3 : Signature Client
Partie 4 : Communication avec le serveur
- **Partie 5 : Identification, import et export des données.**

Description du thème

Propriétés	Description
Intitulé long	**Cas EDF : Développement Androïd - Concepts avancés - Partie 5 : Identification, import et export des données**
Formation concernée	BTS Services Informatiques aux Organisations
Matière	SLAM 4
Présentation	Développement permettant d'aborder des concepts de la programmation Android d'une application embarquée, communiquant avec un serveur. Il aborde les notions : ➢ d'affichage de liste / d'adapter, ➢ de GEOLOCALISATION / GEOCODER, ➢ de graphisme (canvas) et d'encodage JPG, ➢ d'échange avec un serveur WEB (THREAD / JSON / GSON), ➢ d'utilisation d'un SGBDO DB4o.
Notions	Savoirs • D4.1 - Conception et réalisation d'une solution applicative • D4.2 - Maintenance d'une solution applicative Savoir-faire • Programmer un composant logiciel • Exploiter une bibliothèque de composants • Adapter un composant logiciel • Valider et documenter un composant logiciel • Programmer au sein d'un framework
Transversalité	SLAM5
Pré-requis	Développement d'une application Android sous un environnement Eclipse. (Exemple : Cas AMAP Jean-Philippe PUJOL)
Outils	Eclipse, DB4o, OME, Gson, Google play services, Apache, Mysql
Mots-clés	Application mobile, Android, SGBDO, DB4o, Géolocalisation, Géocodage, Thread, json, Gson, MVC, canvas, encodage JPG
Durée	24 heures (8,4,4,4,4) (Temps divisé par 2 si utilisation du squelette application)
Auteur	Pierre François ROMEUF avec la relecture et les judicieux conseils de l'équipe CERTA
Version	v 1.0
Date de publication	Juin 2014

Contexte

Application embarquée sur une solution technique d'accès (STA) sous Android, permettant à un agent EDF d'effectuer sa tournée journalière de relevés des compteurs EDF.

Les principales fonctionnalités sont :

- Identification de l'agent sur le device avec contrôle sur un serveur web,
- Import des clients depuis un serveur web,
- Affichage des clients,
- Saisie des informations clients,
- Aide au déplacement via géolocalisation de la position de l'agent EDF et géocodage de l'adresse client,
- Enregistrement de la signature client validant les informations saisies,
- Export des données sur le serveur web.

Le SGBD embarqué est un SGBDO DB4o. Le serveur distant est un serveur de type LAMP installé sur la ferme de serveurs ou via un hébergement gratuit (ex : http://www.hostinger.fr/)

Cette application peut être dérivée pour de multiples besoins : ceux des livreurs, des commerciaux, des visiteurs, des contrôleurs ...

Le code fourni en annexe nécessite de votre part une compréhension.
Il représente normalement votre travail de programmeur, de fouille sur internet, avec tests, compréhension et modifications du code.

Il vous est fourni afin que ce développement ne représente qu'un travail raisonnable et pour vous présenter les différentes facettes du développement sur Android.

Conseils : consultez notamment developer.android.com, le tutoriel du zéro (http://uploads.siteduzero.com/pdf/554364-creez-des-applications-pour-android.pdf), etc.

Identification, Import, Export

Vous avez maintenant, avec la partie 4, toutes les connaissances pour terminer ce cas.

Pour les *Activity Actimport* et *Actexport*, nous allons utiliser GSON qui est une bibliothèque permettant de gérer les *parsing* JSON sous Android d'une manière robuste et facile.

- Importez dans le dossier *libs, gson-2.2.4.jar* (https://code.google.com/p/google-gson/downloads/list)
- Ajoutez le .jar à votre projet via *Add JARs* dans le *Java BuildPath* onglet *libraries.*

Règles de gestion :

Au démarrage du projet aucun agent EDF n'est identifié sur le STA, donc seule l'image "Identification" du menu est accessible
Suite à une identification réussie, l'enregistrement des informations de l'agent EDF détenteur du STA est stocké dans la base DB4o.
Ces informations seront utiles par la suite pour les activity Import et Export.
Si un agent EDF est déjà identifié (présence d'un enregistrement AgentEdf dans la base de donnée DB4o), le clic sur l'image "identification" dans le menu provoquera l'affichage d'un message d'avertissement indiquant la présence d'information concernant l'agent EDF détenteur du STA et demandant la confirmation de l'écrasement de ces informations.
En effet le STA peut être donné à un autre agent EDF, où l'identifiant et le mot de passe de l'agent EDF détenteur du STA ont pu être modifiées.

Suite à une identification correcte, s'il n'y a pas la présence d'informations concernant les clients dans la base DB4o, seule l'image "import" du menu devient accessible.
Après un import réussi des clients à visiter, les images "Relevé des compteurs" et "Sauvegarde" deviennent accessibles.
Après un export (sauvegarde) réussi des données clients sur le serveur, les informations de la base DB4o concernant les clients sont effacées. Seules restent donc accessibles les images "Identification" et "Import"

- Créez une méthode dans la MainActivity controleAffichage() à

A partir de Modele, appel des méthodes qui permettent de récupérer la List<Client> et la List<AgentEdf>
Les informations de ces 2 listes vous permettront de rendre accessibles ou non les images du menu.
.setEnabled(**false**); ou .setEnabled(**true**);

Identification

Pour simplifier, nous considérerons que seul un agent EDF unique à un instant t peut utiliser le STA.

- Créez une classe AgentEdf dans votre projet avec comme données identifiant, motDePasse.
- Générer les getter et setter
- Créez dans Modele les méthodes utiles permettant de stocker et modifier l'enregistrement unique.
- Modifiez sur le serveur, le fichier connect.php, afin que sur réception de l'identifiant et du mot de passe de l'agent EDF, il renvoie 1 si l'agent est identifié, 0 sinon.

Layout de l'activity Identification

- Modifiez le *Layout* de l'activity Identification en ajoutant 2 *EditText* permettant de saisir l'identifiant et le mot de passe.
- Modifiez le label du bouton "testcon" du layout de l'activity Identification
- Ajoutez un bouton "cancel"

On rajoutera exceptionnellement (si Wamp ou Lamp) un *EditText* pour l'adresse du serveur 192.168.xxx.xxx qui n'est pas utile pour l'application normale.

Code :

- Ajoutez une méthode testPresenceAgent() qui vérifiera la présence d'un enregistrement AgentEdf dans la base DB4o

Si un agent est déjà identifié, un message d'avertissement "Un agent est déjà identifié, voulez vous écraser ces informations ?" sera affiché via un AlertDialog.Builder (bouton ok et cancel).

La gestion du clic sur cancel de la boite de dialogue provoquera le retour à la MainActivity avec l'information 'cancel'

- Modifiez le code de la méthode `retourIdentification` pour gérer le retour de connect.php

Si l'identification est réussie, écriture ou modification des informations de l'agent EDF et retour à la MainActivity en lui indiquant que l'identification est réussie.
Si l'identification échoue affichage d'un message indiquant que le mot de passe ou l'identifiant sont erronés.
La gestion du clic sur cancel provoquera le retour à la MainActivity avec l'information 'cancel'.

Communication avec la MainActivity

On associera, à la fin de l'activity Identification (il en sera de même pour les activities Export et Import), un retour vers la MainActivity avec une communication de l'information de réussite ou d'échec de l'opération.

Comment communiquer cette information à l'activity appelante (ici la MainActivity) ?

La communication avec la *MainActivity* se fera via un `startActivityForResult.` Idem pour *import* et *export* dans la *MainActivity* qui permet de démarrer une activity et de pouvoir récupérer un résultat.

- Déclarez dans la MainActivity 3 variable entières permettant de reconnaitre l'activity qui provoque le retour

```
static final int codeRetourIdentification = 1;
static final int codeRetourImport = 2;
static final int codeRetourExport = 3;
```

- Modifiez la méthode imageClick

Exemple pour l'activity Identification

```
// On créé un objet Bundle, c'est ce qui va nous permetre d'envoyer
// des données à l'autre Activity si nécessaire
Bundle objetbunble = new Bundle();
objetbunble.putString("xxxx", "xxxxx");
// On créé l'Intent qui va nous permettre de démarrer l'autre
// Activity
Intent intent = new Intent(this, Identification.class);
// On affecte à l'Intent le Bundle que l'on a créé
intent.putExtras(objetbunble);
// On démarre l'autre Activity
startActivityForResult(intent, codeRetourIdentification);
```

- Créez la méthode qui permet de récupérer le résultat de l'activity appelée

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            // on regarde quelle Activity a répondu

            switch (requestCode) {
            case codeRetourIdentification:
                  // On regarde qu'elle est la réponse envoyée et en fonction de la
                  // réponse on ????.

                  switch (resultCode) {
                  case ???:
                        return;

                  case ????:
                        return;
                  }
            }
      }
```

Dans *l'Activity* appelée

```
setResult(???);
finish();
```

Exemple de réponse : Activity.*RESULT_OK, Activity.RESULT_CANCELED,...*

Import des données

- Créez l'activity ActImport

Règles de gestion :

Si l'import des données clients est réussi, effacer toutes les données clients de la base DB4o, puis ajouter les données de l'import.

Layout

Un bouton "ok" et un bouton "cancel"

Code :

- Créez dans Modele 2 méthodes :
  - *void deleteClient()* qui permet de supprimer toutes les instances de la classe client (cf. *listeclient()* avec *db.delete(…)*)
  - *void addClient(ArrayList<Client> vclient)* qui, à partir d'une collection de Client, va les ajouter à DB4o. Nous ne testerons pas l'existence de ces objets puisque c'est une création (appel de la méthode après l'appel de *deleteClient()*).

- Modifiez la classe Connexion pour intégrer l'activity ActImport.

- Modifiez les paramètres d'appel de la classe Connexion pour une exécution de import.php

- Ajoutez à *l'Activity ActImport* la méthode *retourImport(StringBuilder sb)*

Cette méthode sera appelée à partir de la méthode *onPostExecute* de la classe *Connexion*.

Ci joint le code d'une partie de cette méthode qui permet, à partir de la réception d'un tableau de Json et à partir de GSON, de reconstituer les objets de la classe Client et de les stocker dans une Arraylist

```
public void retourImport(StringBuilder sb)
{
        JsonElement json = new JsonParser().parse(sb.toString());
        JsonArray varray = json.getAsJsonArray();
        Gson gson = new GsonBuilder().setDateFormat("yyyy-mm-dd").create();
        ArrayList<Client> listeClient = new ArrayList<Client>();
        for (JsonElement obj : varray) {
              Client client = gson.fromJson(obj.getAsJsonObject(), Client.class);
              client.setSituation(0);
              client.setDernierReleve(0.0);
              client.setDatedernierreleve(new Date());
              client.setSignatureBase64("");
              listeClient.add(client);
              }
      //....
}
```

- Terminez le code de la méthode retourImport
  - Suppression et ajout des clients si ArrayList n'est pas vide
  - Retour à la main activity avec le code de retour

- Créez *import.php* à l'image de *connect.php.*

Export des données

- Créez l'activity ActExport

Règles de gestion :

Si l'export des données clients est réussi, effacer toutes les données clients de la base DB4o.
Ici, nous ne nous soucierons pas de la problématique des accès concurrents puisque les enregistrements sont uniquement dédiés aux relevés des compteurs et non à l'aspect commercial ou financier de l'activité.
Nous considérerons que c'est un export global et non partiel : l'agent EDF les envoie quand ils sont tous traités.

Layout

Un bouton "ok" et un bouton "cancel"

Code :

- Modifiez la classe Connexion pour intégrer l'activity ActExport.
- Ajouter à la classe *Actexport* la variable *private String sClient="";*
- Modifiez les paramètres d'appel de la classe Connexion
  - pour une exécution de export.php
  - pour ajouter le 4° paramètre : sClient.
- Modifiez la méthode on create afin d'initialise sClient

sClient est une chaine de caractères des informations clients, encodées en JSON et séparées par un séparateur (choix arbitraire @@@) permettant de les traiter dans *export.php*

```
Modele modele = new Modele();
ArrayList<Client> listClient = modele.listeClient();
Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd hh:mm:ss.S")
            .create();
int i = 0;
for (Client vcli : listClient) {
      sClient = sClient + gson.toJson(vcli);
      i++;
      if (i < listClient.size()) {
            sClient = sClient + "@@@";
      }
```

- Modifiez la classe *Connexion* afin d'ajouter la prise en compte de ce 4ème paramètre dans la méthode *doInBackground*

```
String vlistcli="";
if (vclassactivity.contains("Actexport")) {
            vlistcli = params[3];
      }
```

Puis ajout de ce paramètre aux paramètres passés en *post* de la même manière que pour *l'Activity Identification*

```
if (vclassactivity.contains("Actexport") ) {
                        JSONObject jsonParam = new JSONObject();
                        jsonParam.put("ID", vid);
                        jsonParam.put("pass", vpass);
                        jsonParam.put("listcli", vlistcli);
                        out.write(jsonParam.toString());
                        out.flush();
                    }
```

- Créez *export.php* à l'image de *connect.php.*

Traitement et modification des clients via :

```
$data = json_decode(file_get_contents("php://input"),true);

$id=$data['ID'];
$pass=$data['pass'];
$listcli=$data['listcli'];

$tabclient = explode("@@@", $listcli);
foreach ($tabclient as $client)
{
$clientdecode = json_decode($client);
$sql = "UPDATE client SET dateDernierReleve='".$clientdecode-
>{'dateDernierReleve'}."',situation=".$clientdecode->{'situation'}.",dernierReleve=".$clientdecode-
>{'dernierReleve'}.",signatureBase64='".$clientdecode->{'signatureBase64'}."' where identifiant='".$clientdecode-
>{'identifiant'}."'";
$result = mysql_query($sql);
}
```

- Modifiez *export.php* pour renvoyer 1 si l'update des clients est réussi, 0 sinon

- Ajoutez à *l'Activity ActExport* la méthode *retourExport(StringBuilder sb)*

Cette méthode sera appelée à partir de la méthode *onPostExecute* de la classe *Connexion*.
Cette méthode permet un retour à la main activity avec le code de retour en analysant sb.