Projet Viticulture TP 3 : bases de données externes

Description du thème

Ce TP permet d'aborder par la pratique les techniques permettant d'accéder à des bases de données stockées sur un serveur distant à l'aide d'Android.

Propriétés	Description
Intitulé long	Projet Viticulture - TP 3 : bases de données distantes
Formation concernée	BTS Services informatiques aux organisations
Matière	SLAM 4, PPE
Présentation	Les TP permettent de découvrir différentes méthodes pour accéder et utiliser des bases de données MySQL stockées sur un serveur distant dans un environnement mobile Android
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
Pré-requis	Développement mobile sous Android, TP1 et 2 du projet Viticulture.
Outils	Eclipse, framework Android API 18, PHP/MySQL
Mots-clés	Application mobile, Android, MySQL, PHP, DAO
Durée	6h
Auteur(es)	Mathieu Capliez
Version	v 1.0
Date de publication	Février 2015

Remarques pour l'enseignant :
Pour utiliser ces TP, il est indispensable de bien configurer les projets fournis et le serveur de base de données.
Dans les projets Android, les variables `serveur` et `chemin` doivent contenir respectivement l'adresse du serveur web utilisé et le chemin vers les scripts PHP.
Sur le serveur web, la base de données MySQL doit être créée, et les scripts PHP configurés correctement. La configuration des variables `$login`, `$mdp`, `$bd`, `$serveur` est accessible dans le fichier `fonctions.php`.

Énoncé

Documentation :
http://www.json.org/fatfree.html
http://developer.android.com/reference/org/json/JSONObject.html
http://developer.android.com/reference/android/os/AsyncTask.html

Ressources :
- projets BD3 et BD4 ;
- archive zip contenant les scripts PHP.

Rappel du contexte de l'activité précédente (TP1) :
Dans le cadre d'un concours, les producteurs de vin de la région Aquitaine sont évalués selon la qualité gustative de leur production.
Lors de la dégustation, chaque membre du jury attribue une note au viticulteur pouvant aller de 0 à 100. Si le jury souhaite modifier cette dernière, ou s'il goûte un autre vin du viticulteur, il pourra changer la note associée.

Le contexte de cet exercice reprend et complète celui du TP1.

Évolution du contexte :
Les membres du jury pourront, de la même manière, accéder et modifier les données sur les viticulteurs qui seront enregistrées sur un serveur distant.

Les applications Android peuvent utiliser des bases de données externes en complément d'une base de données interne SQLite ou de manière autonome.
Habituellement, une application se connecte à un SGBDR directement ou via un *middleware*. Cette connexion permet d'exécuter des requêtes SQL via des connecteurs et de récupérer des curseurs contenant le résultat.
De par le type de connexion réseau des périphériques mobiles, on évite une connexion directe avec le SGBDR. Un serveur web sert le plus souvent d'intermédiaire.
Dans ce TP, différentes méthodes d'accès aux bases de données distantes seront étudiées. Vous pourrez ainsi déterminer les pratiques et/ou les plus adaptées à vos projets.

Après avoir importé les deux projets "bd3" et "bd4", vous testerez et observerez les codes afin de répondre aux questions posées dans les parties suivantes.

Partie 1 : Base de données

Question 1 – Structure de données

À l'aide des sources PHP et de l'annexe 1, répondre aux questions suivantes :

1.1. Identifier le rôle de chaque script PHP mettant en œuvre des requêtes SQL.

À l'aide d'un navigateur et par l'intermédiaire d'un serveur web, observer le résultat d'exécution du script **`getViticulteurs.php`**. (http://serveurWeb/getViticulteurs.php)

1.2. Dans le script **`getViticulteurs.php`**, indiquer le type de données créé (variable **`$resultat`**) lors de la lecture du curseur, et le rôle de la fonction **`json_encode()`**.

Question 2 – Accès aux données

À l'aide des affichages en `logcat` lors de l'exécution du projet **bd3**, des classes correspondantes, et de l'annexe 2, répondre aux questions suivantes.

2.1. Lors de l'exécution du projet, suivre l'ordre d'appel des différentes méthodes à l'aide du `logcat`. En déduire l'ordre chronologique d'exécution des fonctions.

2.2. Dans la classe **AccesHTTP**, indiquer le rôle des variables affichées en `logcat`.

2.3. En étudiant le code de la méthode **doInBackground()** de la classe **AccesHTTP**, identifier le contenu, puis le rôle des variables **cnxHttp**, **paramCnx**.

Lorsqu'une application a besoin d'une grande quantité de données, le téléchargement peut prendre beaucoup de temps. C'est souvent le cas lorsque la réception est de mauvaise qualité ou lorsque l'utilisateur se trouve en zone non couverte par la 3G, ou 4G.

2.4. En utilisant des applications connectées sur un smartphone (application de cartographie, navigateur web, etc.), l'application se bloque-t-elle lors du téléchargement des données, ou au contraire, l'interface est-elle toujours active ?

2.5. Expliquer pourquoi le téléchargement des données n'est pas fait par la classe **MainActivity** ?

2.6. Expliquer pourquoi l'affichage des données n'est pas fait dans la classe **AccesHTTP** ?

2.7. En observant les traces d'exécution du projet, indiquer le rôle des méthodes **doInBackground()**, et **onPostExecute()**. Expliquer pourquoi la méthode **onPostExecute()** est redéfinie dans la classe **MainActivity.**

2.8. Bilan : expliquer le fonctionnement de la classe **AccesHTTP**, et son utilisation dans **MainActivity**.

Question 3 – Organisation du code et niveau d'abstraction

À l'aide des affichages en `logcat` lors de l'exécution du projet **bd4**, des classes correspondantes et des annexes 2 et 3, répondre aux questions suivantes.

Après avoir lancé l'application, observer les affichages en `logcat` correspondant au chargement de la liste de viticulteurs ainsi que le code de la méthode **remplirSpinViticulteurs()** dans la classe **MainActivity** .

3.1. Schématiser l'enchaînement des appels de méthodes nécessaires au remplissage de la liste déroulante (classes **MainActivity**, **Viticulteur**, **ViticulteurDAO**, et **AccesHTTP).**

3.2. Pour un traitement similaire à celui étudié dans le projet `bd3`, une classe supplémentaire est nécessaire. Indiquer laquelle. En extrapolant ce qui a été vu au TP1, expliquer le rôle de cette classe.

Dans la classe `AccesHTTP`, le corps de la méthode `onPostExecute` peut être redéfini dans une classe anonyme (question 2.6).

3.3. En observant le code de la méthode **remplirSpinViticulteurs()**, indiquer quelles méthodes de la classe **ViticulteurDAO** sont redéfinies lors de la création de l'objet **viticulteurAcces.**

3.4. Rechercher d'où proviennent ces méthodes, et pourquoi elles doivent obligatoirement être implémentées.

3.5. Indiquer à quel moment cette méthode est appelée, et quelle instruction provoque cet appel.

3.6. Quel avantage a-t-on à ce que ces méthodes soient redéfinies dans la classe MainActivity ?

À l'aide de l'annexe 4 et du projet **bd4**.

3.7. En prenant exemple sur le raisonnement des questions précédentes, expliquer comment se fait le chargement de données après saisie de l'identifiant de juré.

3.8. Expliquer comment savoir dans quelle version de la méthode **onTacheTerminee()** doit-on placer le traitement exécuté après le téléchargement.

Partie 2 : Application

Question 4 – Adaptation de la solution

Lorsqu'un juré lance l'application sur un smartphone, il doit saisir son identifiant. Le nom et le prénom du juré sont alors téléchargés (un objet de la classe **Jure est instancié avec ces données)** et placés dans la propriété **MainActivity.jureSelectionne**. De la même manière, le viticulteur sélectionné dans la liste déroulante est placé dans la propriété **MainActivity.viticulteurSelectionne**. Lorsque le juré a saisi une note pour le viticulteur et appuyé sur le bouton **Enregistrer**, le score doit s'enregistrer dans la table **noter**.

4.1. Identifier les données à transmettre au serveur pour l'insertion et mettre en place le script PHP.

4.2. À l'aide des classes existantes, créer les classes `Noter` et `NoterDAO` permettant d'enregistrer les données saisies.

4.3. Compléter la méthode **onClick()** du bouton **btnEnregistrer** pour finaliser l'enregistrement.

Partie 3 : Réflexion

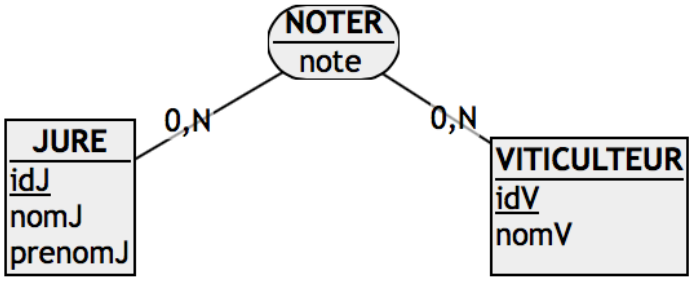
Question 5 – Analyse de la structure applicative

5.1. Quel est le rôle des classes DAO ?

5.2. Pourquoi est-il plus judicieux de faire appel à la classe **AccesHTTP** dans les classes DAO plutôt que directement dans la classe **MainActivity** ?

5.3. La classe **ViticulteurDAO** est abstraite et implémente l'interface **EventAsyncViticulteur**. Cette conception impose la redéfinition des méthodes **onTacheTerminee()**. Expliquer l'intérêt de cette obligation.

Annexe 1 : Modèle entité association



Annexe 2 : Règles de conception d'applications Android

Extrait de la documentation Android :
http://developer.android.com/guide/components/processes-and-threads.html

[...] the Android UI toolkit is not thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android's single thread model:

Do not block the UI thread
Do not access the Android UI toolkit from outside the UI thread
[...]
Because of the single thread model described above, it's vital to the responsiveness of your application's UI that you do not block the UI thread. If you have operations to perform that are not instantaneous, you should make sure to do them in separate threads ("background" or "worker" threads).

Annexe 3 : Méthode MainActivity.remplirSpinViticulteurs()

```
105 public void remplirSpinViticulteurs(){
106     System.out.println("methode remplirSinViticulteurs()");
107     ViticulteurDAO viticulteurAcces = new ViticulteurDAO() {
108         @Override
109         public void onTacheTerminee(final ArrayList<Viticulteur> resultat){
110             System.out.println("MainActivity.getViticulteurs : "+resultat.toString());
111 
112             // remplissage du spinner a l'aide de la liste recuperee
113             Spinner spinViticulteurs = (Spinner) findViewById(R.id.spinViticulteurs);
114             ArrayAdapter<String> spinViticulteursAdapter = new ArrayAdapter<String>(
115                     MainActivity.this.getBaseContext(),android.R.layout.simple_spinner_item
116                     );
117             for(int i=0;i<resultat.size();i++){
118                 spinViticulteursAdapter.add(resultat.get(i).getNomV());
119             }
120             spinViticulteurs.setAdapter(spinViticulteursAdapter);
121 
122             // gestion des evenements sur le spinner de selection du viticulteur
123             spinViticulteurs.setOnItemSelectedListener(new OnItemSelectedListener(){
124                 @Override
125                 public void onItemSelected(AdapterView<?> arg0, View arg1,
126                         int arg2, long arg3) {
127                     // TODO Auto-generated method stub
128                     System.out.println(arg2 + " "+ resultat.get(arg2));
129                     viticulteurSelectionne = resultat.get(arg2);
130                 }
131 
132                 @Override
133                 public void onNothingSelected(AdapterView<?> arg0) {
134                     // TODO Auto-generated method stub
135                 }
136             });
137 
138         }
139 
140         @Override
141         public void onTacheTerminee(String resultat) {
142             // TODO Auto-generated method stub
143         }
144 
145         @Override
146         public void onTacheTerminee(Viticulteur resultat) {
147             // TODO Auto-generated method stub
148         }
149     };
150     viticulteurAcces.getViticulteurs();
151 }
```

Annexe 4 : Gestion de l'événement : modification de l'identifiant de juré

```
59              public void afterTextChanged(Editable s) {
60                  jureSelectionne = null;
61                  lblJure.setText("");
62                  if (s.length()>0){
63                      Long saisie = Long.valueOf(s.toString());
64
65                      JureDAO jureAcces = new JureDAO(){
66                          @Override
67                          public void onTacheTerminee(Jure resultat){
68                              jureSelectionne = resultat;
69                              Log.d("log","jure : "+jureSelectionne);
70                              if (jureSelectionne != null){
71                                  lblJure.setText(jureSelectionne.getPrenomJ()+ " "+ jureSelectionne.getNomJ());
72                              }
73                          }
74
75                          @Override
76                          public void onTacheTerminee(String resultat) {
77                              // TODO Auto-generated method stub
78
79                          }
80
81                          @Override
82                          public void onTacheTerminee(ArrayList<Jure> resultat) {
83                              // TODO Auto-generated method stub
84
85                          }
86                      };
87                      jureAcces.getJureByIdJ(saisie);
88                  }
89              }
```