GUIDE
ÉTAPE 4 : ENREGISTREMENT DANS UNE BASE DE DONNÉES DISTANTE

Contenu

SQLite est très pratique en local, mais il est parfois nécessaire de mutualiser les informations.

Dans notre exemple, imaginons que l'on veuille mettre en place une base de données centrale qui enregistre les profils pour pouvoir les restaurer si l'application est désinstallée. On utilise aussi classiquement des bases distantes pour mémoriser des informations provenant de plusieurs supports, voire pour les transférer d'un support à l'autre.

Dans notre exemple, la base distante va remplacer la base locale, en ajoutant en plus la possibilité de voir l'historique des mesures enregistrées. On aurait pu le faire aussi avec la base locale, mais vu qu'on va le montrer ici, autant éviter de le faire 2 fois.

En fait, ce n'est pas directement une connexion à une base de données distante qui va se faire, mais une connexion à un serveur HTTP et donc par exemple à une page PHP qui va elle-même interroger une base de données (ou n'importe quoi d'autre) et qui va afficher (donc retourner) un résultat.

Pourquoi ne pas gérer une connexion directement à une base de données distante ? Parce que l'on ne maîtrise pas les temps de réponse via le réseau et il ne faut pas bloquer l'application en attendant la réponse. L'idée est de demander au serveur distant de faire le travail et, s'il y a besoin d'un retour d'information, de le récupérer dans un processus différent.

Attention

À partir de maintenant, vous n'allez plus pouvoir faire de tests directement sur votre mobile car la page PHP et la base de données vont être installées sur un serveur local wamp, non accessible de l'extérieur. Pour que cela fonctionne à partir d'un mobile, il faut installer la page PHP et la base de données sur un serveur distant (par exemple un hébergeur) accessible via Internet ou dans une VM. Il suffit alors de mettre les bonnes configurations dans le code : tout ceci est expliqué plus loin. Il est donc conseillé dans un premier temps de réaliser toute cette partie en utilisant un émulateur. Une fois que vous aurez terminé, si vous voulez, vous ferez les tests avec un serveur distant.

1. La classe AccesHTTP

Comme pour la base locale, vous allez travailler avec une classe existante qui va permettre de gérer la connexion au serveur distant via le protocole HTTP. En réalité, on a besoin de 2 classes : dans les sources, récupérez les classes AccesHTTP.java et AsyncResponse.java et copiez-les via l'explorateur dans votre projet (dans le dossier outils). Ouvrez les 2 nouvelles classes sous Android Studio (car normalement vous devez les voir dans la liste des classes).

Voyons un peu plus en détail le contenu de la classe AccesHTTP car il est important de comprendre son fonctionnement. Elle est assez courte. Lisez attentivement les explications suivantes.

1A. L'entête

Cette classe hérite de AsyncTask qui est une classe de type Thread. Cela signifie qu'elle s'exécute dans un processus isolé, sans bloquer le reste de l'application.

Pour une connexion distante, surtout avec un support mobile où parfois les connexions ne sont pas très rapides, c'est indispensable que l'application ne soit pas bloquée en attendant le résultat d'un appel au serveur.

Cependant, cela pose d'autres problèmes, en particulier continuer à travailler sans avoir encore reçu la réponse. Vous verrez plus loin qu'il faudra trouver une astuce pour n'exécuter certains traitements que si une réponse est bien revenue du serveur.

1B. Le constructeur

Il ne contient rien (juste l'appel du constructeur de la classe mère que l'on met par défaut quand on crée un constructeur vide, justement pour éviter qu'il soit vide).

1C. La méthode addParams

Cette méthode permet de construire la chaîne de paramètres à partir des couples "nom/valeur" reçus en paramètre de la méthode.

Si la chaîne est vide, le couple "nom/valeur" est directement ajouté, sinon il est précédé de "&". Le résultat va donc ressembler à ceci :

```
nom1=valeur1&nom2=valeur2&…&nomN=valeurN
```

Un encodage est aussi réalisé pour ne pas avoir de problème de format lors du transfert des informations.

1D. La méthode doInBackground

C'est la méthode qui va gérer en tâche de fond (donc dans un processus indépendant) la connexion au serveur distant et donc par exemple à une page PHP qui va interroger une base de données et retourner un résultat.

Après une ligne de code pour éliminer certaines erreurs et quelques déclarations, le reste du code est dans un try/catch car la connexion peut éventuellement échouer (serveur non disponible…) et cela ne doit pas pour autant poser de problème à l'application.

La méthode doInBackground est appelée par la méthode execute sur un objet de type AccesHTTP (on le verra plus loin). Elle reçoit en paramètre un tableau d'urls. Quand on appellera la méthode execute (qui appelle doInBackground), on n'enverra qu'une url que l'on récupère ici dans urls[0]. Ce sera l'url de la page PHP sollicitée.

Lisez bien tous les commentaires laissés dans le code pour mieux comprendre le fonctionnement. En résumé, on ouvre une connexion on serveur distant, à partir de l'url reçu en paramètre, on envoie les paramètres en POST (qui seront récupérés côté PHP) et on attend une réponse du serveur distant. D'où l'importance que ce code s'exécute dans un thread indépendant, pour ne pas bloquer le programme.

1E. La méthode onPostExecute

Cette méthode est automatiquement exécutée dès qu'une réponse est reçue par la méthode doInBackground. Pour exploiter le retour obtenu par cette méthode, il suffirait de créer un objet de type AccesHTTP à chaque fois que l'on veut se connecter au serveur distant, et redéfinir à la volée cette méthode, en lui mettant le code à exécuter suite à la réponse du serveur.

Mais il y a mieux : gérer la réponse asynchrone. Regardez le contenu de la méthode :

```
delegate.processFinish(this.ret.toString());
```

Le but est d'appeler la méthode processFinish qui n'est pas dans le thread, alors qu'on est dans un thread. Pour cela, on utilise un "delegate". Si vous regardez plus haut, delegate est un objet de type AsyncResponse, l'autre classe que vous avez récupérée. Jetez un œil dans cette autre classe. En réalité c'est une interface qui force la redéfinition de la méthode processFinish. On placera plus loin cette méthode dans la classe AccesDistant. Cette méthode sera donc automatiquement appelée lors d'un retour du serveur, et recevra en paramètre l'information retournée. En fait, delegate est une propriété publique : elle recevra l'instance de la classe qui contiendra la méthode processFinish.

2. La base de données distante

Le test va se faire avec une base de données MySQL qui sera sollicitée par une page PHP, elle-même appelée à distance par l'application Android. Que ce soit lors de tests locaux (avec wamp) ou sur un serveur distant, la page PHP sera installée au même endroit que la BDD, donc l'accès se fera en localhost entre les 2. En revanche, l'application Android pourra accéder à la page PHP à distance, en donnant la bonne adresse du serveur.

Sur votre ordinateur, lancez wamp, phpMyAdmin et créez sous MySQL la base coach (toujours avec l'encodage utf8_unicode_ci) qui ne contient qu'une seule table : la table profil avec la même structure que vous aviez utilisée dans SQLite (pour datemesure, prenez le type DATETIME, pour le reste, type INT longueur 3 pour poids/taille/age et longueur 1 pour sexe, et gardez datemesure en clé primaire).

3. La page PHP à solliciter

Dans le dossier www de wamp, créez un dossier coach. Dans ce dossier, copiez le fichier fonctions.php récupéré dans les sources.

3A. Connexion à la BDD : fonctions.php

Ouvrez le fichier pour voir son contenu : vous remarquez qu'il ne contient qu'une fonction qui permet de gérer la connexion à la base de données coach.

Si vous mettez en ligne la base de données et les fichiers PHP, il faudra modifier le contenu du fichier fonctions.php en changeant le contenu des variables, par rapport au paramétrage en ligne : $login, $mdp et $bd. A priori, vous ne devriez pas changer $serveur si votre page PHP est installée sur le même serveur que la base de données. Il faudra aussi changer l'adresse IP stockée dans la variable SERVERADDR qui se trouve dans la classe AccesDistant (vous comprendrez en construisant la classe AccesDistant un peu plus loin).

Important

Il existe 2 formats de bases de données : MySQL et MariaDB. La chaîne de connexion est la même (elle commence par mysql) en revanche les 2 types de BDD n'écoutent pas sur le même port. Sans changer le paramétrage par défaut de wamp, normalement MySQL est le SGBDR par défaut à l'écoute sur le port 3306 (d'où le paramétrage dans fonctions.php). Si vous décidez de travailler avec MariaDB ou si, dans la version de wamp que vous utilisez, c'est MariaDB qui est le SGBDR par défaut, alors soit utilisez MariaDB, soit mettez MySQL comme SGBDR par défaut (clic droit sur l'icône de wamp > outils > inverser SGBD), soit donnez changez le numéro de port dans fonctions.php.

3B. Exécution des ordres provenant d'Android : serveurcoach.php

Il faut une seconde page PHP que vous allez créer : appelez-la serveurcoach.php.

Cette page doit permettre de réaliser pour le moment 2 opérations, exactement comme le faisait la base locale : enregistrer un nouveau profil, et retourner le profil le plus récent.

3C. Structure de la page serveurcoach.php

Avant tout, il faut inclure le fichier "fonctions.php" en début de page, afin de pouvoir se connecter à la base de données. Écrivez l'include :

```
include "fonctions.php";
```

L'application Android va envoyer des paramètres en mode POST à la page PHP. Donc la page PHP doit contrôler les paramètres et leurs contenus pour réaliser les opérations demandées.

On va partir du principe que le paramètre aura pour nom "operation" et aura pour contenu, pour le moment, soit "enreg" (pour enregistrer un profil), soit "dernier" (pour récupérer le profil le plus récent).

Donc, faites un premier test pour vérifier l'existence du paramètre "operation" (reçu en POST, mais on peut généraliser en testant en REQUEST) avec la fonction isset :

```
if(isset($_REQUEST["operation"])){
```

Dans ce test, si le paramètre existe, faites un nouveau test pour vérifier son contenu : il contient pour le moment soit "enreg", soit "dernier".

3D. Enregistrement d'un nouveau profil

Dans le test "enreg", récupérez la variable request "lesdonnees". Décodez le contenu de cette variable en utilisant la fonction json_decode et affectez le résultat dans la variable $donnee. Cette variable sera donc un tableau : la case 0 contient datemesure, la case 1 le poids, la case 2 la taille, la case 3 l'âge et la case 4 le sexe. Il est conseillé d'affecter les différentes cases dans des variables avec des noms parlants.

Dans un try/catch (car vous allez accéder à la base de données, donc cela peut générer une erreur), commencez par afficher le mot "enreg" suivi d'un signe reconnaissable, du genre "%" :

```
print ("enreg%") ;
```

Quel est l'intérêt ? En réalité, rien ne va s'afficher "à l'écran" mais tout ce que vous allez afficher dans la page PHP (avec print, echo ou var_dump) va être renvoyé par le réseau, en l'état, vers le support mobile. Au retour du serveur, dans l'application Android, vous pourrez savoir l'origine du message en le découpant (grâce à %).

Puis, connectez-vous à la base de données (utilisez la fonction connexionPDO() qui est dans fonctions.php pour valoriser une variable de connexion, par exemple $cnx). Écrivez la requête d'insertion en la stockant dans une variable :

```
$larequete = "insert into profil (datemesure, poids, taille, age, sexe)";
$larequete .= " values (\"$datemesure\", $poids, $taille, $age, $sexe)";
```

Juste après avoir affecté cette requête dans une variable chaine, affichez-là avec print. Cela permettra de voir dans la console, côté application Android, si la requête est correcte, en cas de problème. Il reste à préparer et exécuter la requête en utilisant $cnx :

```
$req = $cnx->prepare($larequete);
$req->execute();
```

Enfin, dans le catch, ajoutez ces 2 lignes :

```
print "Erreur ! %". $e->getMessage();
die();
```

La 1re ligne permet de renvoyer vers le serveur le message d'erreur généré : ça peut toujours servir.

La 2e ligne permet d'arrêter le script puisqu'il y a une erreur grave. Ceci dit, ça ne changera pas grand-chose côté Android.

3E. Récupération du profil le plus récent

Dans le test "dernier", commencez par faire un :

```
print("dernier%");
```

pour récupérer sous Android l'origine du message. Puis utilisez la même logique de connexion, excepté que vous devez passer par un curseur. Le but est de récupérer toutes les informations du profil le plus récent. Donc attention, on ne récupère qu'une seule ligne du curseur (avec un if et non un while). Mettez ces informations dans un tableau associatif (si vous avez utilisé FETCH_ASSOC, c'est normalement le cas).

```
$req = $cnx->prepare("select * from profil order by datemesure desc");
$req->execute();
// s'il y a un profil, récupération du premier
if($ligne = $req->fetch(PDO::FETCH_ASSOC)){
```

Par contre ensuite, dans le test, il faut l'afficher au format JSON... lisez la suite pour mieux comprendre.

3F. Le format JSON

Seules des chaines de caractères peuvent transiter du téléphone vers le serveur distant, et vice-versa. Du coup, on ne peut pas envoyer le "tableau". Tout ce qui est de type complexe (tableau, objet) doit être converti en chaine pour pouvoir transiter par le réseau.

Un des formats les plus utilisés est le format JSON. À quoi cela ressemble ? Toutes les informations ne sont mises que dans une variable de type chaine, est tout est délimité par des accolades. Par exemple :

```
{"datemesure":"…",  "poids":67, "taille":165, "age":35, "sexe":1}
```

Si un tableau à plusieurs dimensions est formaté en JSON, les accolades vont s'imbriquer pour marquer les différentes colonnes, et les lignes dans chaque colonne.

Donc, revenons à notre programme :

Une fois le dernier profil récupéré suite à la lecture avec le curseur, donc dans le test, vous allez l'afficher après l'avoir transformé au format JSON, de cette façon :

```
print(json_encode($ligne));
```

La fonction json_encode va permettre de transformer le tableau $ligne au format JSON. À l'arrivée, côté Android, on pourra le décoder.

4. Accès à internet à partir du support

Dans l'application Android Studio, il faut donner l'autorisation à accéder à internet.

Ouvrez le fichier AndroidManifest.xml qui se trouve dans "manifests". Juste avant la balise fermante </manifest> (et normalement, après la balise fermante </application>), ajoutez la ligne :

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

Il est parfois nécessaire de donner des permissions de ce genre dans une application Android. Par exemple, c'est obligatoire pour pouvoir accéder à l'album photo du téléphone, au répertoire…

5. Classe AccesDistant

Il est temps de créer la classe AccesDistant : créez-là dans le package modele. Cette classe contiendra, entre autres, la méthode processFinish, dont on a parlé plus haut, qui va permettre de gérer la réception des retours du serveur, et que l'on va créer plus loin.

5A. Propriétés

Il faut mémoriser l'adresse de la page PHP sur le serveur. Déclarez la constante de classe SERVERADDR qui va contenir l'adresse http du fichier serveur.php. Attention, l'adresse doit contenir l'adresse IP réelle, et non localhost, donc du genre :

```
"http://192.168.165.101/coach/serveurcoach.php"
```

Vous pouvez trouver votre adresse IP en ouvrant une fenêtre de commande et en tapant ipconfig. Il faut récupérer l'adresse IPv4 qui commence à priori par 192.168. Quand vous mettrez en ligne la base de données et les pages PHP, il faudra bien sûr changer cette adresse.

5B. Constructeur

Créez le constructeur qui ne fait rien, donc insérez juste l'instruction :

```
super() ;
```

5C. Méthode processFinish

Le but est que cette classe gère la fameuse méthode processFinish pour gérer le retour asynchrone du serveur. Commencez par faire en sorte que la classe AccesDistant implémente votre interface AsyncResponse.

Vous allez voir une ampoule rouge apparaître car cette interface force la redéfinition de la méthode processFinish. Cliquez sur la flèche à côté de l'ampoule et choisissez "implement methodes". Vous allez voir apparaître la méthode redéfinie.

Cette méthode va donc permettre de gérer le retour du serveur.

Commencez par faire un Log.d de ce genre :

```
Log.d("serveur", "************" + output);
```

(output étant le paramètre de la méthode et contenant le texte retourné par le serveur)

Quel est l'intérêt ? Si votre programme ne marche pas et que vous vous demandez ce qui s'est passé côté serveur, cet affichage console vous permettra de voir ce qui s'est passé côté serveur, en particulier s'il y a eu une erreur PHP. Vous n'aurez aucun autre moyen de le voir. Vous pourrez alors récupérer le contenu du message dans la console, et de le copier dans un fichier html pour afficher le message d'erreur (ce sera plus lisible que dans la console, car les erreurs PHP sont souvent affichées dans des balises de tableaux HTML).

Vous allez ensuite splitter le message reçu en le découpant sur le caractère "%" (le caractère que vous avez précédemment utilisé) pour séparer l'instruction du message :

```
String[] message = output.split("%");
```

Le résultat est alors récupéré dans le tableau message, sachant que la case d'indice 0 contient l'instruction ("enreg", "dernier" ou "Erreur !") et la case d'indice 1 contient le véritable retour du serveur.

Pour éviter une erreur (dans le cas où le serveur n'aurait pas renvoyé une chaine en 2 parties), faites un test sur la longueur du tableau (avec la propriété length) pour contrôler qu'il contient plus d'une case.

Dans ce test, faites une succession de tests sur la case 0 en la comparant avec l'une des 3 instructions : "enreg", "dernier" et "Erreur !" (utilisez la méthode equals pour la comparaison). Pour le moment, pour chacune des instructions, contentez-vous de faire un Log.d pour afficher dans la console l'instruction et le message correspondant qui se trouve dans la case 1 de message. Cela va permettre de contrôler que le PHP fonctionne correctement. On s'occupera ensuite de modifier le code pour prendre en compte correctement les retours d'information.

5D. Méthode envoi

Il faut maintenant écrire la méthode envoi, qui ne retourne rien et qui reçoit en paramètre l'opération (type String) et la donnée à envoyer au format JSON (type JSONArray) : lesDonneesJSON.

Dans cette méthode, déclarez et créez un objet accesDonnees de type AccesHTTP.

Ajoutez l'instruction :

```
accesDonnees.delegate = this;
```

qui permet de faire le lien de délégation entre AccesDistant et AccesHTTP. C'est ainsi que delegate, contenant une instance de AccesDistant, pourra appeler la méthode processFinish dès qu'il y aura un retour du serveur.

Il faut maintenant ajouter les 2 paramètres de envoi, en paramètre de accesDonnees (en utilisant la méthode addParam) :

```
accesDonnees.addParam("operation", operation);
accesDonnees.addParam("lesdonnees", lesDonneesJSON.toString());
```

Vous reconnaissez les noms des variables récupérées en POST dans le code PHP. Il ne reste plus qu'à faire appel au serveur, en appelant la méthode execute sur l'objet accesDonnees, en envoyant en paramètre la constante SERVERADDR. Rappelez-vous que la méthode execute appelle en réalité (dans un second temps) la méthode doInBackground (de la classe AccesHTTP).

6. Modifications du reste du code

Dans la classe Controle, mettez en commentaire ou supprimez la déclaration de l'objet accesLocal et remplacez-le par la déclaration de l'objet accesDistant.

Dans la méthode getInstance, mettez en commentaire ou supprimez la ligne :

```
accesLocal = new AccesLocal(contexte) ;
```

et ajoutez la ligne :

```
accesDistant = new AccesDistant() ;
```

mettez aussi en commentaire ou supprimez la ligne qui permettait de récupérer le dernier profil en local, et ajoutez la ligne :

```
accesDistant.envoi("dernier", new JSONArray());
```

Cette ligne va permettre d'envoyer l'ordre "dernier" au serveur distant, avec un objet JSONArray vide puisque cet ordre n'est pas censé envoyer des informations mais au contraire demander au serveur qu'il nous retourne des informations.

Pour l'envoi du profil, c'est un peu plus compliqué car il faut d'abord convertir l'objet au format JSON. Dans la classe Profil, ajoutez la méthode publique convertToJSONArray qui ne reçoit aucun paramètre et qui retourne un JSONArray. Cette méthode doit :

- déclarer un objet de type List et lui affecter une instance de ArrayList() ;
- ajouter à cet objet (avec la méthode add) successivement les propriétés dateMesure, poids, taille, age et sexe ;
- retourner une instance de la classe JSONArray en mettant en paramètre la liste qui vient d'être créée.

De retour dans la classe Controle, méthode creerProfil, mettez en commentaire ou supprimez la ligne :

```
accesLocal.ajout(profil);
```

et ajoutez la ligne :

```
accesDistant.envoi("enreg", profil.convertToJSONArray());
```

Avec cette ligne, on envoie au serveur le profil au format json, avec l'ordre de l'enregistrer.

Il est temps de faire un premier test. Lancez l'application : normalement il n'y a pas d'ancienne valeur qui s'affiche puisque la base locale n'est plus interrogée.

Important

Si vous rencontrez une erreur précisant que l'accès au HTTP n'est pas permis, alors allez dans le fichier AndroidManifest.
Il faut préciser qu'on accepte le transfert de requêtes sans cryptage. Pour cela, positionnez-vous à la fin de la ligne de la balise ouvrante "<application", validez pour crée une ligne vide et tapez le code suivante :

```
android:usesCleartextTraffic="true"
```

Pensez ensuite à enregistrer le fichier.

Saisissez des valeurs et faites un calcul. Contrôlez la console : vous devriez voir afficher quelque chose du genre :

```
enreg%insert into profil (datemesure, poids, taille, age, sexe)
values ("Thu Jul 01 14:36:53 GMT+00:00 2021", 42, 160, 53, 0)
```

Cette ligne correspond à l'affichage console (Log.d) qui se trouve tout au début de processFinish. Vous devriez avoir aussi une seconde ligne quasiment identique puisque vous avez aussi pour le moment fait un affichage console dans le test correspondant à un retour "enreg".

Allez vérifier dans phpMyAdmin si le profil s'est ajouté. Normalement non car la date reçue n'avait pas le bon format par rapport à celui attendu par MySQL. C'est encore un problème de conversion. Dans la méthode convertToJSONArray() de la classe Profil, convertissez dateMesure en String (avec la méthode convertDateToString que vous avez créée dans la classe MesOutils) avant de l'insérer dans la liste.

Refaites un test et vérifiez que le profil s'est bien inséré dans MySQL et le format de la date.

Relancez l'application, normalement le dernier profil doit être récupéré dans la console. Vous devriez donc avoir un affichage du genre :

```
dernier%{"datemesure":"2021-07-01
14:40:13","poids":"42","taille":"160","age":"54","sexe":"0"}
```

suivi d'une ligne quasiment identique.

Cela voudra dire que tout fonctionne bien : il y a bien dès le démarrage de l'application, récupération au format JSON du dernier profil enregistré mais bien sûr il ne s'affiche pas encore. Il ne nous reste plus qu'à le traiter.

7. Récupération à partir d'un thread

Quand on dit "il ne reste plus qu'à", en réalité ce n'est pas si simple.

7A. Récupération plus complexe du profil

Les accès à une base locale sont instantanés et surtout se font dans le même thread (le même processus) que le reste du programme. Du coup, le programme attend la réponse de la base locale avant de continuer les traitements, et donc l'affichage des informations récupérées. Mais là, pour éviter de bloquer le programme, l'appel du serveur distant se fait dans un thread séparé. Du coup, la vue ne sait pas à quel moment précis le retour se fait. Ce qui veut dire que ce qu'on avait fait dans MainActivity (l'appel de recupProfil dans la méthode init) ne peut plus fonctionner ainsi.

Dans MainActivity, mettez en commentaire l'appel de recupProfil à la fin de la méthode init. On va appeler cette méthode plus loin et à partir du contrôleur, du coup mettez-là en public.

7B. Le contrôleur doit attendre un profil

De retour dans la classe Controle, il est nécessaire d'utiliser le contexte pour accéder à l'Activity qui gère l'affichage. Du coup, vu que dans getInstance, on reçoit un contexte (qui servait auparavant pour la base locale), faites en sorte de le mémoriser dans une propriété privée statique (context). Attention, mettez la valorisation dans le if (c'est le tout premier contexte qui doit être mémorisé, donc MainActivity).

Remarquez au passage que dans la signature de la méthode creerProfil, vous n'avez plus besoin du contexte car il était nécessaire pour la BDD locale mais il ne l'est plus pour la BDD distante : enlevez le paramètre et faites de même lors de l'appel de creerProfil dans MainActivity. Revenez dans la classe Controle.

Contrairement à la base locale, on ne peut pas directement valoriser profil dans la méthode getInstance (pour le problème de thread expliqué plus haut). Du coup, vous allez créer une nouvelle méthode publique setProfil qui ne retourne rien et qui reçoit en paramètre profil de type Profil. Faites en sorte que cette méthode valorise la propriété profil avec le paramètre. Dans cette méthode, le but est maintenant d'appeler la méthode recupProfil de MainActivity, pour que l'affichage se fasse. Logiquement, vu que MainActivity est un contexte, il suffirait de faire context.recupProfil(). Mais essayez : vous allez voir que ça ne marche pas. Pourquoi ? Parce que contexte est de type Context qui est une classe mère de MainActivity. Donc les méthodes de MainActivity ne sont pas visibles à partir de la classe Context. Il faut transtyper context (sachant que c'est tout de même un objet de type MainActivity) pour accéder à la bonne méthode :

```
((MainActivity)context).recupProfil();
```

7C. AccesDistant reçoit le profil

Il ne reste plus qu'à faire en sorte que cette nouvelle méthode setProfil soit appelée au bon moment : quand l'information revient du serveur. Donc, dans la classe AccesDistant, il faut pouvoir accéder au contrôleur : dans les propriétés, déclarez l'objet controle de type Controle. Dans le constructeur, enlevez le super() qui ne sert plus à rien et affectez à la propriété controle l'appel de la méthode getInstance sur la classe Controle, en mettant null en paramètre (car on sait que l'objet controle existe forcément déjà, donc inutile de lui envoyer un contexte). Ceci dit, dans la méthode getInstance de la classe Controle, pour éviter que null soit mis dans la propriété context, ne valorisez cette propriété que si context, reçu en paramètre, n'est pas null.

Dans la méthode processFinish, dans le test correspondant à "dernier", enlevez le Log.d. Le but est d'appeler la méthode setProfil de Controle, mais on ne peut pas encore car il faut au préalable construire l'objet profil à partir des informations reçues du serveur. Commencez par mettre dans une variable "info" de type JSONObject, une nouvelle instance de JSONObject en mettant en paramètre message[1]. Vous allez voir que la ligne est soulignée en rouge. Effectivement, la transformation en objet JSON n'est possible que si le paramètre est au format JSON. Il faut donc capturer l'erreur. Cliquez sur l'ampoule rouge et choisissez "suround try/catch". Si l'ampoule n'est pas visible, faites un clic droit sur le soulignement rouge et demandez "show context action".

Dans le try, après la création info, vous allez récupérer dans des variables locales séparées, les différentes parties du profil reçu : pour récupérer par exemple le poids qui se trouve dans la case 1 du JSON info que vous venez de créer, vous aller écrire :

```
Integer poids = info.getInt("poids");
```

Observez la ligne : info étant un JSONObject, on peut récupérer les différents éléments de l'objet avec les méthodes get (getInt, getString...) en précisant le nom de la propriété à récupérer dans l'objet. Faites de même pour taille, age et sexe. Pour dateMesure (de type String), c'est un peu plus complexe car on récupère un String et on doit le transformer en Date. On pourrait tout simplement utiliser la méthode convertStringToDate mais elle n'utilise pas un bon format car MySQL possède les dates au format "`yyyy-MM-dd hh:mm:ss`".

Du coup il faudrait une autre méthode convertStringToDate dans MesOutils en changeant le contenu de expectedPattern. Le plus malin est de modifier la méthode actuelle en lui ajoutant cette information en paramètre plutôt qu'en mettant le pattern en dur, et d'appeler cette nouvelle méthode dans AccesDistant pour formater la date avec le format "`yyyy-MM-dd hh:mm:ss`" en second paramètre. Mais attention, on a déjà utilisé cette méthode de conversion à un autre endroit du programme, avec un seul paramètre. Pour éviter de toucher au code existant, le plus propre est de surcharger la méthode convertStringToDate dans MesOutils, cette fois avec un seul paramètre (comme elle l'était avant) et en lui faisant juste retourner l'appel de l'autre méthode avec 2 paramètres, en mettant en second paramètre "`EEE MMM dd hh:mm:ss 'GMT+00:00' yyyy`".

Une fois tous les éléments récupérés (dans AccesDistant), vous allez pouvoir créer un objet profil avec toutes ces informations. Il ne reste plus qu'à appeler la méthode setProfil de la classe Controle en envoyant en paramètre le profil qui vient d'être créé.

Faites un test : vous devriez cette fois voir s'afficher le dernier profil enregistré dans la base MySQL.

Si vous voulez faire un test sur un support mobile, cette fois il faut installer les fichiers PHP et la base de données sur un serveur distant (par exemple un hébergeur), accessible via Internet. Il faut aussi changer la configuration du fichiers fonctions.php ainsi que l'adresse du serveur dans AccesDistant. Tout est expliqué plus haut, dans cette partie.

Pensez à faire un "commit and push" pour l'enregistrement sur Bitbucket.

CNED, BP 60200, 86980 Futuroscope Chasseneuil Cedex, France

87D22TDWB1C21