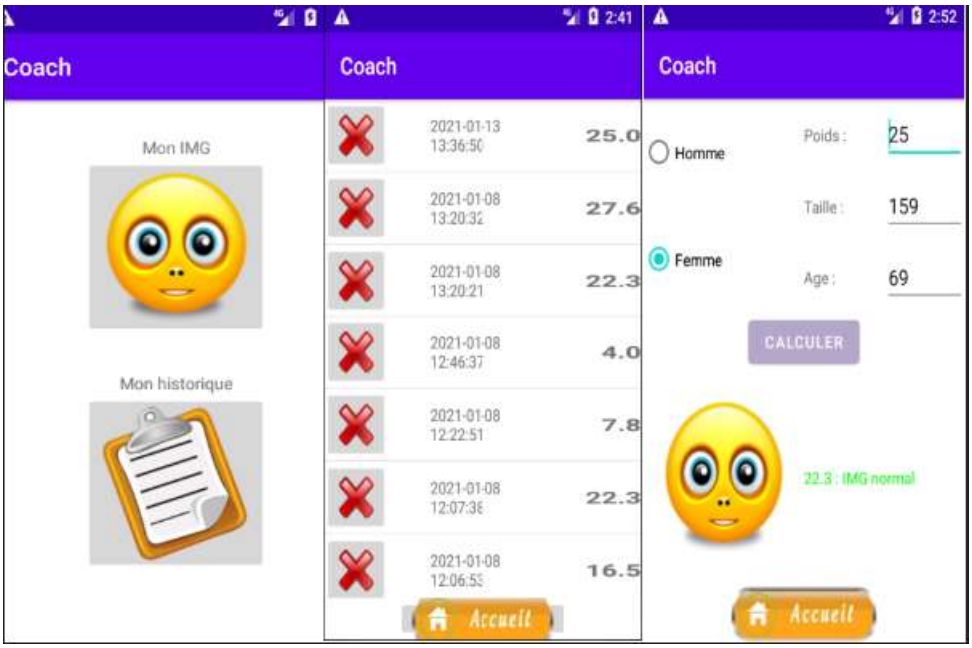
Fiche de Synthèse : Coach *Janvier 2021*

Développement :
Il s'agit d'une application Androïd, afin d'encourager l'utilisateur dans sa perte de poids.
L'application est reliée à une base distante, ce qui facilite l'enregistrement d'un historique.



Cadre :
Il s'agit d'un TP proposé dans le cadre du cours "Réalisation et Maintenance de Composants Logiciels ", par Elisabeth Martins Da Silva.

Support :
Développé en Java avec Androïd Studio 4.1.1 principalement, une bdd gérée par PhpMyAdmin, et l'accès à la bdd via php.

Contraintes :
C'est une 1ère approche de la programmation sur mobile, sous Androïd.

Difficultés rencontrées :
Difficultés pour l'exécution de l'application sur mon téléphone personnel.

Description détaillée du développement

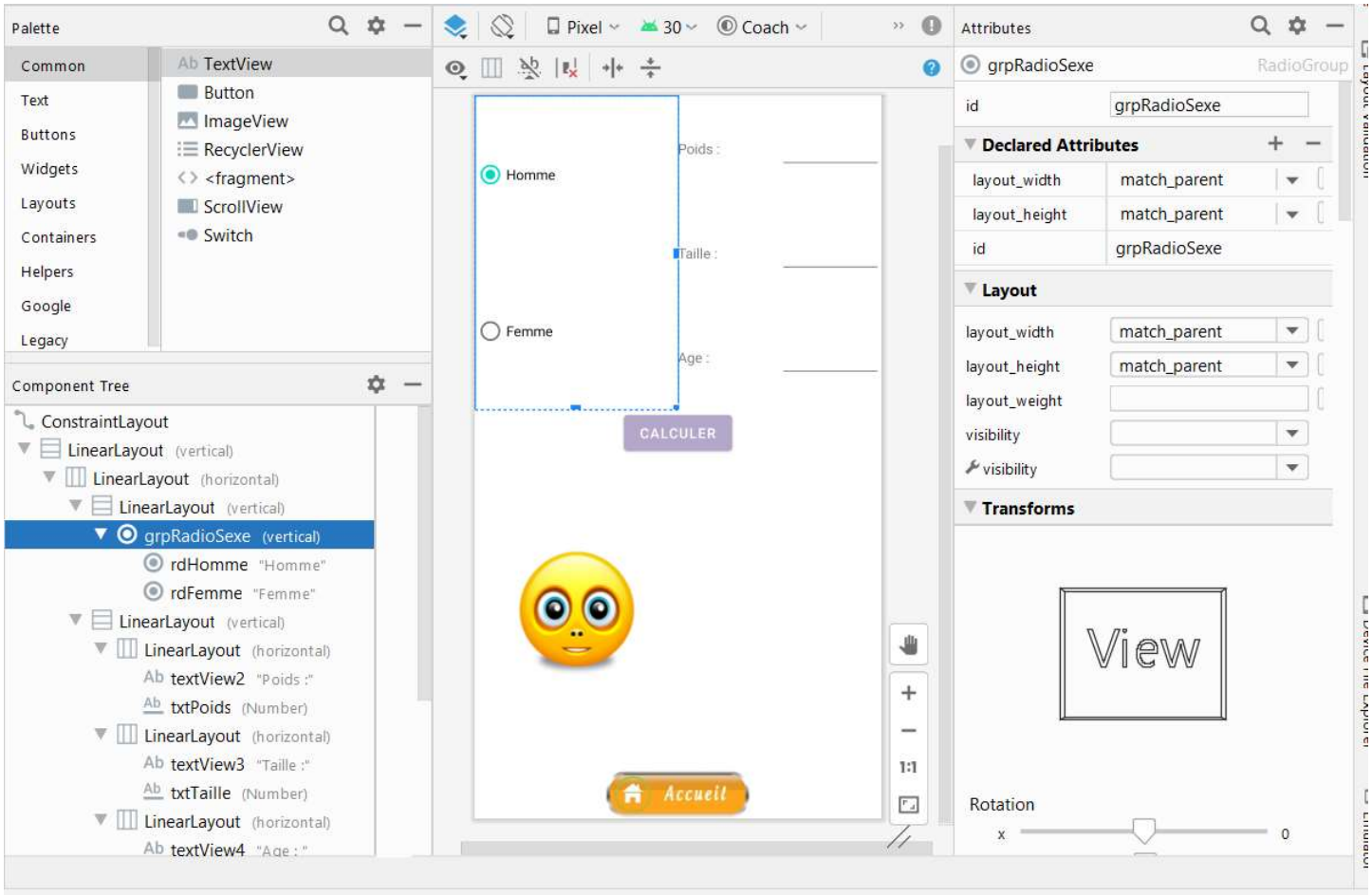
J'ai suivi les étapes suivantes pour le développement:
- organisation du code
- création d'une 1ère interface
- développement des premières fonctionnalités
- enregistrement grâce à la sérialisation
- enregistrement en local avec Sqlite
- enregistrement dans une base distante
- multiplication des interfaces
- des listes interactives

1. Organisation du code

Cette application a été développée suivant le modèle MVC – Modèle Vue Contrôleur. Cela permet une conception claire et efficace grâce à la séparation des données de la vue et du contrôleur, un gain de temps de maintenance et d'évolution de l'application, et une plus grande souplesse pour une organisation entre différents développeurs.

2. Création d'une 1ère interface

Constuire une interface avec Androïd Studio est finalement assez simple, sous réserve qu'on ait la patience de comprendre comment fonctionnent les layouts



On peut ajouter les éléments que l'on souhaite via un glisser-déposer, et à droite on peut modifier les propriétés (nom, centré ou non, taille, etc …)

Au fur et à mesure de la construction avec la souris, Androïd Studio génère automatiquement le code correspondant :

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/androi
    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=".vue.CalculActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center_horizontal"
            android:layout_weight="1"
            android:orientation="horizontal">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_gravity="center_horizontal"
                android:layout_weight="1"
                android:gravity="center_horizontal|center_vertical"
                android:orientation="vertical">

                <RadioGroup
```

Surtout, le plus intéressant, c'est qu'une classe MainActivity se créée automatiquement. Il suffit de la déplacer dans le dossier 'Vues' et on pourra commencer à travailler.

```java
package com.example.coach.vue;

import …

public class MainActivity extends AppCompatActivity {

    private Controle controle;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
```

3. Développement des premières fonctionnalités

Il a fallu d'abord créer, dans le dossier 'Modèle', une classe Profil afin de pouvoir calculer l'IMG (Indice de Masse Graisseuse). Cette classe contient les propriétés suivantes :

```
public class Profil implements Serializable, Comparable {

    // constantes
    private static final Integer minFemme = 15; // maigre si en dessous
    private static final Integer maxFemme = 30; // gros si au dessus
    private static final Integer minHomme = 10; // maigre si en dessous
    private static final Integer maxHomme = 25; // gros si au dessus

    // propriétés
    private Integer poids;
    private Integer taille;
    private Integer age;
    private Integer sexe;
    private float img;
    private String message;
    private Date dateMesure;
```

Outre le Constructeur qui valorise ces propriétés, et Getter et les Setter habituels, c'est ici qu'on effectue les calculs et que l'on attribue le message approprié en fonction du résultat :

```
/**
 * valorise img avec le bon calcul
 */
private void calculIMG(){
    float tailleEnCm = ((float)taille)/100;
    img = (float)((1.2*poids/(tailleEnCm*tailleEnCm))+(0.23*age)-(10.83*sexe)-5.4);
}

/**
 * valorise message avec le bon message en fonction de l'img
 */
private void resultIMG(){
    message = "normal";
    Integer min = minFemme, max = maxFemme;
    if(sexe == 1){
        min = minHomme;
        max = maxHomme;
    }
    if(img < min){
        message = "trop faible";
    }else{
        if(img > max){
            message = "trop élevé";
        }
    }
}
```

A ce moment, il faut créer une classe Contrôle, mais qui devra être une instance unique (Singleton)

```
/**
     * Création de l'instance unique de Controle
     * @return instance
     */
    public static final Controle getInstance(Context contexte){
        if(contexte != null) {
            Controle.contexte = contexte;
        }
        if(Controle.instance==null){
            Controle.instance = new Controle();
        }
        return Controle.instance;

    }
```

Ainsi, il y a une seule instance de contrôle : soit celle qu'on est en train d'utiliser, et s'il n'y en a pas alors seulement il y en a une qui se créée.

Ensuite, c'est bien au contrôleur de créer un profil :

```
/**
 * Création du profil
 * @param poids
 * @param taille en cm
 * @param age
 * @param sexe 1 pour homme, 0 pour femme
 */
public void creerProfil(Integer poids, Integer taille, Integer age, Integer sexe, Context
context){
    Profil unProfil = new Profil(new Date(), poids, taille, age, sexe);
    lesProfils.add(unProfil);
    Log.d("date", "*************************" + new Date());
}
```

Dans la vue, nous pouvons donc associer un évènement 'clic sur le bouton' en créant une méthode ecouteCalcul() : cette méthode récupère ce qui est saisi dans les cases txtPoids, txtTaille et txtAge, les transtype en Integer, et ensuite affiche le résultat :

```
/**
 * Evénement clic sur le bouton calculer
 */
private void ecouteCalcul(){
    btnCalc.setOnClickListener(new Button.OnClickListener() {
        public void onClick(View v) {
            Integer poids = 0;
            Integer taille = 0;
            Integer age = 0;
            Integer sexe = 0; // femme par défaut
            try {
                poids = Integer.parseInt(txtPoids.getText().toString());
                taille = Integer.parseInt(txtTaille.getText().toString());
                age = Integer.parseInt(txtAge.getText().toString());
            }catch(Exception e){}
            if(rdHomme.isChecked()){
                sexe=1;
            }
            // vérifie que tout est saisi
            if(poids==0 || taille==0 || age==0){
                Toast.makeText(CalculActivity.this, "Veuillez saisir tous les champs",
Toast.LENGTH_SHORT).show();
            }else{
                affichResult(poids, taille, age, sexe);
            }

        }
    });

}
```

4. Enregistrement grâce à la sérialisation

Pour qu'au lancement de l'application le dernier profil s'affiche, une classe Serializer nous a été fournie directement.

Elle a été utilisée : dans la Classe Contrôle, dans la méthode créerProfil()

```
// sérialisation du profil
Serializer.serialize(nomFic, profil, context);
```

Ensuite, une fois le profil créée et donc enregistré, toujours dans la classe Contrôle on créée une méthode pour le récupérer :

```
/**
 * Récupération d'un profil sérialisé
 * @param context
 */
private static void recupSerialize(Context context){
    profil = (Profil)Serializer.deSerialize(nomFic, context);
}
```

Et puis, dans l'Activity, une méthode pour remplir les cases avec le profil ainsi récupéré :

```
/**
 * Récupération du profil sérialisé
 */
public void recupProfil(){
    // si un profil a été mémorisé
    if(controle.getTaille()!=null){
        // affichage des informations du profil
        txtPoids.setText(controle.getPoids().toString());
        txtTaille.setText(controle.getTaille().toString());
        txtAge.setText(controle.getAge().toString());
        if(controle.getSexe()==1){
            rdHomme.setChecked(true);
        } else{
            rdFemme.setChecked(true);
        }
        // remettre à vide le profil :
        controle.setProfil(null);
        // provoque le clic sur le bouton du calcul
        // btnCalc.performClick();
    }
}
```

5. Enregistrement en local avec Sqlite

Une autre possibilité pour conserver un historique, c'est l'enregistrement dans une base de données. Il est possible pour cela d'utiliser une base de données locale (sur le téléphone), telle SqLite.

La classe MySQLiteOpenHelper nous a donc été fournie, déjà toute prête, dont voici un extrait :

```
public class MySQLiteOpenHelper extends SQLiteOpenHelper {

      // propriété de création d'une table dans la base de données
      private String creation="create table profil ("
                  + "datemesure TEXT PRIMARY KEY,"
                  + "poids INTEGER NOT NULL,"
                  + "taille INTEGER NOT NULL,"
                  + "age INTEGER NOT NULL,"
                  + "sexe INTEGER NOT NULL);";

    /**
     * Construction de l'accès à une base de données locale
     * @param context
     * @param name
     * @param version
     */
      public MySQLiteOpenHelper(Context context, String name, int version) {
            super(context, name, null, version);
            // TODO Auto-generated constructor stub
      }

...
```

Grâce à cet outil, on peut développer une méthode ajout() dans la classe AccesLocal :

```
/**
 * Pour ajouter un profil (identifié par la date)
 * @param profil
 */
public void ajout(Profil profil) {
    this.bd = this.accesBD.getWritableDatabase();

    String req = "INSERT INTO profil (datemesure, poids, taille, age, sexe) values ";
    req += "(\"" + profil.getDateMesure() + "\","
            + profil.getPoids() + ","
            + profil.getTaille() + ","
            + profil.getAge() + ","
            + profil.getSexe() + ")";
    bd.execSQL(req);
}
```

6. Enregistrement dans une base distante

Nous arrivons à la dernière possibilité d'enregistrement (et la plus intéressante), c'est à dire la base distante, telle qu'on peut la manipuler à partir d'une application 'classique', enregistrée sur un serveur (distant donc ...).

Le principe est simple : l'application va se connecter à une page http, qui elle-même va interroger la base de données.

Pour gérer tout ceci, une classe AccesHTTP.java et une classe AsyncResponse.java nous ont été fournies. Voici la classe AccesHTTP :

```java
public class AccesHTTP extends AsyncTask<String, Integer, Long> {

	// propriétés
	private ArrayList<NameValuePair> parametres;
	private String ret = null;
	public AsyncResponse delegate = null;

	/**
	 * Constructeur : ne fait rien
	 */
	public AccesHTTP() {
		parametres = new ArrayList<NameValuePair>();
	}

	/**
	 * Construction de la chaîne de paramètres à envoyer en POST au serveur
	 * @param nom
	 * @param valeur
	 */
	public void addParam(String nom, String valeur) {
		parametres.add(new BasicNameValuePair(nom, valeur));

	}

	/**
	 * Méthode appelée par la méthode execute
	 * permet d'envoyer au serveur une liste de paramètres en GET
	 * @param strings contient l'adresse du serveur dans la case 0 de urls
	 * @return null
	 */
	@Override
	protected Long doInBackground(String... strings) {
		HttpClient cnxHttp = new DefaultHttpClient();
		HttpPost paramCnx = new HttpPost(strings[0]);

		try {
			// encodage des paramètres
			paramCnx.setEntity(new UrlEncodedFormEntity(parametres));
			// connexion et envoi des paramètres, attente de réponse :
			HttpResponse reponse = cnxHttp.execute(paramCnx);
			// transformation de la réponse :
			ret = EntityUtils.toString(reponse.getEntity());
		} catch (UnsupportedEncodingException e) {
			Log.d("erreur encodage ", "****************** " + e.toString());
		} catch (ClientProtocolException e) {
			Log.d("erreur protocole ", "****************** " + e.toString());;
		} catch (IOException e) {
			Log.d("erreur InputOutput ", "****************** " + e.toString());;
		}
		return null;
	}

	/**
	 * Sur le retour du serveur, envoi l'information retournée à processFinish
	 * @param result
	 */
	@Override
	protected void onPostExecute(Long result) {
		try {
			delegate.processFinish(ret.toString());
		} catch (JSONException e) {
			Log.d("erreur JSON ", "***********" + e.toString());
		}
	}

}
```

Cette classe est de type Thread, elle s'exécute donc dans un processus isolé sans bloquer le reste de l'application.

Concernant la page PHP qui se sert d'intermédiaire entre l'application Androïd et la base de données, outre une fonction de création d'un PDO avec les bons paramètres pour se connecter, cette page va donc se charger des requêtes, et surtout elle va convertir ce qu'elle récupère (de l'appli androïd en cas d'un enregistrement dans la base, ou bien à partir de la base pour l'envoyer à l'application Androïd) en format JSON, puisque seules les chaînes de caractères peuvent transiter du téléphone vers le serveur distant.

```
elseif($_REQUEST["operation"] == "enreg") {
      try {
         // récupération des données en POST
         $lesdonnees = $_REQUEST["lesdonnees"];
         $donnee = json_decode($lesdonnees);
         $datemesure = $donnee[0];
         $poids = $donnee[1];
         $taille = $donnee[2];
         $age = $donnee[3];
         $sexe = $donnee[4];
         // insertion dans la bdd
         print("enreg%");
         $cnx = connexionPDO();
         $larequete = "INSERT INTO coach_profil (datemesure, poids, taille, age, sexe)";
         $larequete .= "VALUES(\"$datemesure\", $poids, $taille, $age, $sexe)";
         print($larequete);
         $req = $cnx->prepare($larequete);
         $req->execute();
      } catch(PDOException $e) {
         print "erreur !%" . $e->getMessage();
         die();
      }
```

Il faut donc maintenant, naturellement, créer une classe AccesDistant.java pour gérer tout ceci.

Ci-dessous, un exemple des instructions pour enregister un nouveau profil dans la base :

```
/**
 * Retour du serveur distant
 * @param output
 */
@Override
public void processFinish(String output) throws JSONException {
    Log.d("Serveur", "***************" + output);
    // découpage du message reçu :
    String[] message = output.split("%");
    // dans message[0] : soit "enreg", soit "dernier", soit "erreur"
    // dans message [1] : le reste du message
    // s'il y a bien 2 cases :
    if(message.length > 1) {
        if(message[0].equals("enreg")) {
            Log.d("enreg", "*************" + message[1]);
        } else if(message[0].equals("dernier")) {
```

Pour faire l'inverse, c'est à dire charger une page de l'application sur le téléphone à partir des informations de la base de données, c'est un peu plus long forcément puisqu'il faut récupérer les paramètres nécessaires et les transtyper dans le bon type pour que l'appli puisse les exploiter.

Dans la classe contrôle, il suffit de remplacer les lignes pour un accès local par des lignes pour un accès distant.

7. Multiplication des interfaces

L'objectif est de créer une page d'accueil qui proposera d'accéder soit à l'historique, soit au calcul de l'IMG :

Pour ce faire, il a fallu changer la page d'accueil de l'application. C'est assez simple grâce à Androïd Studio. Ensuite, une nouvelle Activity pour afficher les images-boutons, et qui sera la nouvelle MainActivity.

Concrètement, cette classe se contente de créer l'activité demandé par l'utilisateur selon le bouton sur lequel il clique. Selon le cas, l'activité concernée sera alors créée :

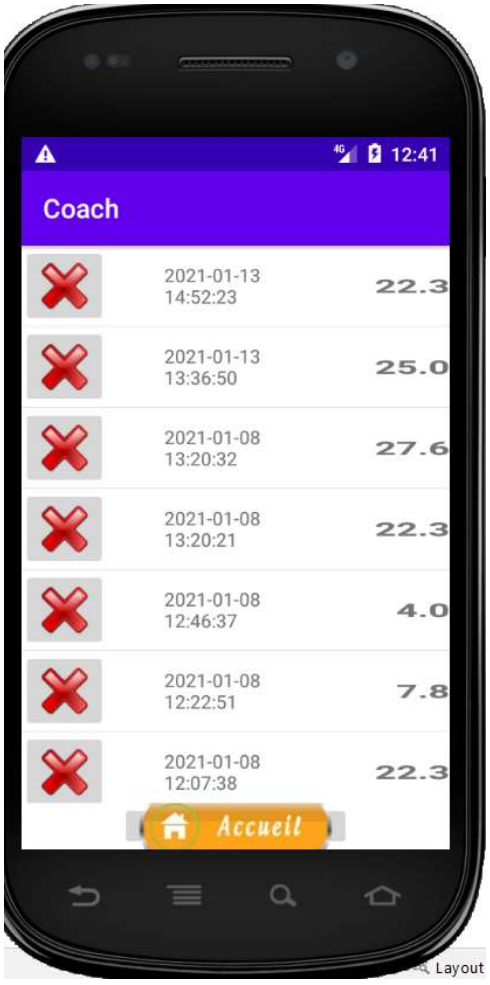
```
public class MainActivity extends AppCompatActivity {

    private Controle controle;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.controle = Controle.getInstance(this);
        ecouteMenu((ImageButton) findViewById(R.id.btnMenuIMG), CalculActivity.class);
        ecouteMenu((ImageButton) findViewById(R.id.btnMenuHistorique), HistoActivity.class);
    }

    /**
     * Ouvrir l'activity correspondante au bouton-menu
     * @param btn
     * @param classe
     */
    private void ecouteMenu(ImageButton btn, final Class classe) {
        btn.setOnClickListener(new ImageButton.OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, classe);
                intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);
                startActivity(intent);
            }
        });
    }
}
```

8. Des listes interactives



Il a bien sûr fallu créer les layouts nécessaires pour la conception de la vue.

Cette étape est un peu difficile à expliquer ici, je vais donc tenter de simplifier : une classe Adapter a été créée, pour nous permettre de formater chaque ligne multi-objets de la liste. Ensuite, dans la page PHP, une requête de sélection récupère tous les profils, les ajoutent dans un array(), et transtype en format JSON.
L'application n'a plus à récupérer les informations pour les afficher.