

BD SQLITE

Atelier 6



November 22, 2024

Hafsi mahdi

ISET Rades

Dans ce projet, la persistance des données est assurée via une base de données SQLite intégrée. Cette approche permet de stocker et de gérer efficacement les profils des utilisateurs localement, même lorsque l'application est hors ligne. Voici une analyse détaillée des classes impliquées et leur fonctionnement.

public class MySQLiteOpenHelper extends SQLiteOpenHelper {  
  
 //proprietés  
 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);";  
  
 */\*\*  
 \* Constructeur  
 \** ***@param*** *context  
 \** ***@param*** *name  
 \** ***@param*** *factory  
 \** ***@param*** *version  
 \*/* public MySQLiteOpenHelper(@Nullable Context context, @Nullable String name, @Nullable SQLiteDatabase.CursorFactory factory, int version) {  
 super(context, name, factory, version);  
 }

**1. Classe MySQLiteOpenHelper**

**Rôle :**

Cette classe hérite de SQLiteOpenHelper et joue le rôle de gestionnaire de la base de données. Elle est responsable de sa création, mise à jour et gestion.

**Fonctionnalités principales :**

1. **Propriétés :**
   * La propriété creation contient une requête SQL pour créer une table appelée profil.
   * La table profil stocke les données essentielles d'un profil, avec les colonnes :
     + datemesure (clé primaire, de type TEXT) pour enregistrer la date et l'heure de la mesure.
     + poids, taille, age, et sexe (tous obligatoires) pour les données physiques du profil.
2. **Constructeur :**
   * Permet d'initialiser un objet MySQLiteOpenHelper avec les paramètres nécessaires comme le contexte, le nom de la base, le CursorFactory, et la version.

*/\*\*  
 \* Si changement de BD  
 \** ***@param*** *sqLiteDatabase  
 \*/*@Override  
public void onCreate(SQLiteDatabase sqLiteDatabase) {  
 sqLiteDatabase.execSQL(creation);  
  
}

**Méthodes :**

* **onCreate(SQLiteDatabase) :**  
  + Exécutée lors de la création de la base de données pour la première fois.
  + La requête SQL de création de la table est exécutée ici.

*/\*\*  
 \* si changment de version  
 \** ***@param*** *sqLiteDatabase  
 \** ***@param*** *i ancienne version  
 \** ***@param*** *i1 nouvelle version  
 \*/*@Override  
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {  
  
}

* **onUpgrade(SQLiteDatabase, int, int) :**
* Prévue pour gérer les changements de version de la base de données.
* Actuellement vide, mais pourrait inclure des scripts de migration pour mettre à jour la structure des tables.

Passant maintenat au class AccesLocal

**Rôle** : Cette classe est un pont entre l'application et la base de données SQLite. Elle permet d'effectuer des opérations CRUD (Create, Read, Update, Delete) sur la table profil.

public class AccesLocal {  
 // proprietes  
 private String nomBase = "bdCoach.sqlite";  
 private Integer versionBase = 1 ;  
 private MySQLiteOpenHelper accesBD;  
 private SQLiteDatabase bd;  
  
 public AccesLocal(Context context) {  
 accesBD = new MySQLiteOpenHelper(context,nomBase,null,versionBase);  
 }

**Propriétés :**

* nomBase et versionBase définissent respectivement le nom et la version de la base de données.
* accesBD est une instance de MySQLiteOpenHelper utilisée pour accéder à la base.
* bd représente une instance SQLiteDatabase pour effectuer des opérations sur la base.

*/\*\*  
 \* ajout d'un profile dans la BD  
 \** ***@param*** *profil  
 \*/* public void ajout(Profil profil){  
 bd = accesBD.getWritableDatabase();  
  
 String req = "INSERT INTO profil (datemesure , poids , taille , age , sexe ) VALUES ";  
 req += "(\""+profil.getDateMesure()+"\","+profil.getPoids()+","+profil.getTaille()+","+profil.getAge()+","+profil.getSexe()+")";  
 try {  
 bd.execSQL(req);  
 } catch (Exception e) {  
 Log.*e*("AccesLocal", "Erreur lors de l'exécution de la requête SQL : " + req, e);  
 }  
  
 }

**ajout(Profil profil) :**

* Ajoute un nouveau profil dans la base.
* Une requête SQL INSERT est construite dynamiquement pour insérer les valeurs du profil (date, poids, taille, âge, sexe).
* Utilise bd.execSQL pour exécuter la requête, et gère les erreurs en les enregistrant dans les logs.

*/\*\*  
 \* recuperation de dernier Profile de la BD  
 \** ***@return*** *\*/* public Profil recupDernier(){  
 bd = accesBD.getReadableDatabase();  
 Profil profil = null;  
 String req = "select \* from profil";  
 Cursor cursor = bd.rawQuery(req, null);  
 cursor.moveToLast();  
 if(!cursor.isAfterLast()){  
 Date date = new Date();  
 Integer poids = cursor.getInt(1);  
 Integer taille = cursor.getInt(2);  
 Integer age = cursor.getInt(3);  
 Integer sexe = cursor.getInt(4);  
  
 profil = new Profil(date,poids,taille,age,sexe);  
  
 }  
 cursor.close();  
 return profil;  
 }  
}

**recupDernier() :**

* Récupère le dernier profil enregistré.
* Exécute une requête SQL SELECT pour récupérer tous les enregistrements dans la table profil.
* Le curseur (Cursor) est déplacé sur le dernier enregistrement pour obtenir les données.
* Retourne un objet Profil basé sur ces données.
* Gère proprement la fermeture du curseur pour éviter les fuites de ressources.

Passant au class Control :

**Rôle dans Control**

* La classe Control utilise AccesLocal pour interagir avec la base de données SQLite, notamment pour :
  + Ajouter un nouveau profil.
  + Récupérer le dernier profil enregistré.

public final class Control {  
 private static Control *instance* = null;  
 private static Profil *profil*;  
 private static String *nomFic* = "saveprofil.bin";  
 private static AccesLocal *accesLocal*;  
 private static AccesDistant *accesDistant*;

**Déclaration de l’attribut AccesLocal :**

* Un objet AccesLocal est utilisé pour gérer l'accès aux données.
* Déclaré comme une propriété privée statique de Control.

*/\*\*  
 \* Creation de l'instance  
 \** ***@return*** *instance  
 \*/*public static final Control getInstance(Context context){  
 if (Control.*instance* == null){  
 Control.*instance* = new Control();  
 *accesLocal* = new AccesLocal(context);  
 *profil* = *accesLocal*.recupDernier();  
 }  
 return Control.*instance*;  
}

**Initialisation de AccesLocal :**

* Lors de la création de l’instance Control via la méthode getInstance(Context), l’objet AccesLocal est initialisé avec le contexte de l’application.

**Récupération du dernier profil :**

* Lors de l'initialisation, la méthode recupDernier d'AccesLocal est appelée pour charger le dernier profil depuis la base :

profil = accesLocal.recupDernier();

*/\*\*  
 \* Creation de profile  
 \** ***@param*** *poids  
 \** ***@param*** *taille en cm  
 \** ***@param*** *age  
 \** ***@param*** *sexe 1 pour homme et 0 pour femme  
 \*/*public void creerProfil(Integer poids , Integer taille , Integer age , Integer sexe , Context context){  
 *profil* = new Profil(new Date(), poids , taille , age , sexe );  
 *accesLocal*.ajout(*profil*);  
  
}

**Ajout de profil dans la base :**

* Dans la méthode creerProfil, en plus de la sérialisation, le profil est ajouté à la base de données via :

accesLocal.ajout(profil);

Passant au classe mainActivity

**Rôle dans MainActivity :**

* **Ajout de profil :** Les données saisies dans l'interface utilisateur (poids, taille, âge, sexe) sont utilisées pour créer un nouveau profil.
* **Affichage du dernier profil :** Si un profil existe déjà dans la base de données, il est affiché lors de l’ouverture de l’application.

**Implémentation :**

1. **Création d’un profil :**
   * Lorsque l’utilisateur appuie sur le bouton "Calcul", la méthode creerProfil du contrôleur est appelée pour sauvegarder les données dans SQLite via AccesLocal.
2. **Récupération et affichage du dernier profil :**
   * Lors de l’initialisation de l’interface, la méthode recupProfil récupère le dernier profil à partir de Control, qui utilise AccesLocal.
3. **Affichage conditionnel :**
   * Les champs d’entrée de l’interface (EditText, RadioButton, etc.) sont remplis avec les données du dernier profil récupéré.