République Tunisienne
Ministère de
l'Enseignement
Supérieur
et de la Recherche
Scientifique

الجمهورية التونسية
وزارة التعليم العالي
والبحث العلمي

Compte rendu TP SQLite

Application de calcul de l'IMG

Auteur
Karim Dhafer

Matière
Développement Mobile sur Android

Classe
L3-DSI3

Année universitaire
2024-2025

Sommaire

Liste des figures

Introduction

Cet atelier se concentre sur l'implémentation de la persistance des données sous Android dans l'application de calcul de l'IMG afin de mémoriser les entrées de l'utilisateur ce qui facilite l'utilitation de l'application. Pour persister les données des profils insérées par l'utilisateur, nous allons utiliser SQLite comme une base de données pour gérer ces informations.

Objectifs

- Comprendre la base de données SQLite sur la platforme Android.
- Implémenter la logique de la base de données SQLite dans l'application.
- Implémenter la base dans les différentes couches de l'application.

1. SQLite sous Android

SQLite est un composant essentiel du système Android, offrant un moyen simple et efficace de gérer les données internes des applications tout en garantissant la persistance des informations sur l'appareil.

1.1 SQLite

SQLite est le moteur de base de données le plus utilisé au monde. Il est open source, rapide, de petite taille et couvre toutes les fonctionnalités standard des bases de données relationnelles. Il nécessite peu de mémoire lors de l'exécution, environ 250 ko, ce qui en fait le meilleur candidat pour les environnements d'exécution disposant de ressources limitées, comme les smartphones. Pour cette raison, il est intégré au système Android, et on le trouve donc sur chaque appareil Android. De plus, il est utilisé par Apple dans la plupart des applications natives sur les serveurs et les ordinateurs macOS, ainsi que sur les appareils iOS tels que les iPads et les iPhones.

1.2 Utilisation de SQLite sous Android

Comme il a été mentionné plus haut, SQLite est présent sur chaque appareil Android. Son atout réside dans la facilité de son utilisation, qui ne nécessite aucune configuration ni gestion manuelle de la base de données. Il suffit de définir les instructions SQL pour créer et mettre à jour la base de données, après quoi celle-ci sera automatiquement gérée par la plateforme Android. Lors de la création d'une application qui utilise la base de données SQLite, celle-ci sera enregistrée par défaut dans le répertoire suivant : */data/data/<nom_du_paquet_de_l'application>/databases*.

1.3 Les avantages d'utilisation de SQLite sous Android

Voici quelques avantages de l'utilisation de SQLite dans une application Android :

- **Légèreté :** SQLite est de petite taille et nécessite peu de mémoire.

- **Facilité d'intégration :** SQLite est intégré par défaut dans le système Android, ce qui permet aux développeurs de l'utiliser sans avoir à installer des composants supplémentaires.
- **Support des fonctionnalités SQL standards :** SQLite prend en charge la plupart des fonctionnalités SQL standards, telles que les requêtes SELECT, INSERT, UPDATE, et DELETE.
- **Gestion automatique des fichiers :** La base de données est gérée automatiquement par la plateforme Android.

2. Implémentation de SQLite dans l'application

Pour implémenter la base de données SQLite dans l'application de calcul de l'IMG, il faut d'abord créer les classes qui vont intéragir avec la base de données, permettant ainsi de créer la base, d'y ajouter des données et de les lire.

2.1 La classe `MySQLiteOpenHelper`

La classe `MySQLiteOpenHelper` est essentielle pour la gestion de la base de données SQLite dans l'application. Elle hérite de la classe `SQLiteOpenHelper`, qui est une classe abstraite prédéfinie dans Android et fournit des méthodes permettant de créer et de gérer la structure de la base de données.

```
1    public class MySQLiteOpenHelper extends SQLiteOpenHelper {...}
```

2.1.1 Intérêt et Rôle

Cette classe permet d'initialiser et de maintenir la base de données de manière structurée, simplifiant le processus de création et de mise à jour des tables. Cela offre une gestion centralisée des changements de structure de la base.
`MySQLiteOpenHelper` redéfinit deux méthodes abstraites très importantes afin de gérer la structure globale de la base de données : `onCreate()` pour la création de la base de données, et `onUpgrade()` pour la mise à jour de la version de la base.

2.1.2 Fonctionnement et implémentation

Tout d'abord, pour gérer plusieurs profils dans la base de données, il est nécessaire de spécifier une clé primaire permettant de différencier ces profils. Pour résoudre ce problème, nous ajouterons la date exacte de création du profil, qui sera implémentée dans la classe `Profil` sous la forme d'un attribut de type `Date` appelé par le constructeur.

```
1    public class Profil implements Serializable {
2    // propriétés
3    private Date dateMesure;
4    ...
5
6    public Profil(Date dateMesure, Integer poids, Integer taille,
        Integer age, Integer sexe) {
7        this.dateMesure = dateMesure;
```

```
        ....
    }
```

Comme le profil inclura désormais la date exacte de son instanciation, le constructeur de la classe `Profil` devra être appelé avec cette date précise lors de la création d'un profil. Cet appel est effectué par la classe `Controle` via la méthode `creerProfil()`, qui invoque le constructeur avec `new Date()`.

```
    public void creerProfil(Integer poids, Integer taille, Integer
       age, Integer sexe, Context contexte) {
        profil = new Profil(new Date(), poids, taille, age, sexe);
        ....
    }
```

Maintenant, nous créons la classe `MySQLiteOpenHelper` sous le package outils puisque c'est une classe utilitaire. Tout d'abord, la classe utilise une requête SQL pour créer la table profil en spécifiant sa structure.

```
    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);";
```

Puis, `MySQLiteOpenHelper` doit déclarer le constructeur par appel au constructeur de sa super classe `SQLiteOpenHelper`.

```
    public MySQLiteOpenHelper(@Nullable Context context, @Nullable
       String name, @Nullable SQLiteDatabase.CursorFactory factory,
       int version) {
        super(context, name, factory, version);
    }
```

Enfin, la classe doit redéfinir les méthodes abstraites suivantes de la classe mère :

- **onCreate() :** exécute la requête SQL pour créer la table lorsque la base de données est initialement créée.

```
    @Override
    public void onCreate(SQLiteDatabase db) { db.execSQL(
       creation); }
```

- **onUpgrade() :** laissée vide, mais elle peut être utilisée pour gérer les changements de version de la base de données.

```
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
        newVersion) {}
```

2.2 La classe `AccesLocal`

La classe `AccesLocal` est conçue pour fournir des méthodes facilitant l'interaction avec la base de données SQLite, notamment l'ajout et la récupération de données. Elle encapsule la logique d'accès aux données tout en utilisant `MySQLiteOpenHelper` pour la gestion de la connexion à la base de données.

2.2.1 Intérêt et Rôle

`AccesLocal` permet de centraliser l'ensemble des opérations liées aux données, favorisant une meilleure organisation et une maintenance simplifiée du code. Elle joue un rôle de passerelle entre la logique métier de l'application et la base de données.

2.2.2 Fonctionnement et implémentation

Pour implémenter ce rôle de passerelle dans cette classe, il faut utiliser un attribut de type `SQLiteDatabase`, une classe prédéfinie dans Android, qui a des méthodes permettant de faire une gestion complète d'une base de données SQLite sur les appareils Android. Parmis ces méthodes, nous avons intérêt à utiliser les deux au-dessous :

- **execSQL(String sql) :** Une méthode pour éxecuter les instructions SQL qui ne sont pas de type SELECT.
- **rawQuery(String sql) :** Une méthode qui éxecute les requêtes SQL de type SELECT et retourne un ensemble de résultat dans un curseur : CURSOR.

Maintenant, après avoir connu les méthodes à utiliser, nous commençerons à coder la classe `AccesLocal` en suivant ces étapes :

1. Déclaration des propriétés comme le nom et la version de la base qui seront utilisés pour gérer la base.

```
1        private String nomBase = "bdTP1.sqlite";
2        private Integer versionBase = 1;
3        private MySQLiteOpenHelper accesBD;
4        private SQLiteDatabase bd;
```

2. Appel au constructeur de **'MySQLiteOpenHelper'** pour initialiser un accès à la base en utilisant l'attribut **'accesDB'**.

```
1        public AccesLocal(Context contexte) {
2            accesBD = new MySQLiteOpenHelper(contexte, nomBase,
                 null, versionBase);
3        }
```

3. Déclaration de la méthode d'ajout qui accepte un profil comme paramètre, ouvre la base en mode écriture pour insérer ce profil dans la table profil de la base de données SQLite et éxecute une requête SQL d'insertion d'une ligne dans la base.

```
1        public void ajout(Profil profil) {
2            bd = accesBD.getWritableDatabase();
3            String req = "INSERT INTO profil (datamesure, poids,
                 taille, age, sexe) VALUES ";
4            req += "(\"" + profil.getDateMesure() + "\"," +
                 profil.getPoids() + "," + profil.getTaille() + ",
                 " + profil.getAge() + "," + profil.getSexe() + ")
                 ";
5            bd.execSQL(req);
6        }
```

4. Déclaration de La méthode **'recupDernier()'** qui ouvre la base en mode lecture afin de récupérer l'ensemble des enregistrements de la table profil, puis le curseur est déplacé à la dernière ligne pour obtenir le dernier profil sauvegardé qui sera à son tour utilisé pour instancier un objet Profil.

```
1         public Profil recupDernier() {
2             bd = accesBD.getReadableDatabase();
3             Profil profil = null;
4             String req = "SELECT * FROM profil";
5             Cursor curseur = bd.rawQuery(req, null);
6             curseur.moveToLast();
7             if(curseur.isAfterLast()) {
8                 Date date = new Date();
9                 Integer poids = curseur.getInt(1);
10                Integer taille = curseur.getInt(2);
11                Integer age = curseur.getInt(3);
12                Integer sexe = curseur.getInt(4);
13                profil = new Profil(date, poids, taille, age,
                      sexe);
14            }
15            curseur.close();
16            return profil;
17        }
```

3. Implémentation de la base dans les autres couches de l'application

3.1 Implémentation dans le controleur

Dans cette partie, nous intégrons les méthodes de la classe `AccesLocal` dans la classe `Controle`. Pour ce faire, nous allons déclarer une instance de la classe `AccesLocal`, puis appeler ses méthodes à partir des méthodes appropriées dans la classe `Controle`.

1. Déclarer la variable de type AccesLocal

```
1     private static AccesLocal accesLocal;
```

2. Appeler les méthodes de création de la base et de récupération du dernier produit, Ces méthodes doivent être invoquées dans la méthode getInstance après la création de l'instance.

```
1     accesLocal = new AccesLocal(contexte);
2     profil = accesLocal.recupDernier();
```

3. Appeler la méthode d'ajout d'un produit immédiatement après son instanciation sous la méthode `creerProfil()`.

```
1     public void creerProfil(Integer poids, Integer taille,
          Integer age, Integer sexe, Context contexte) {
2         profil = new Profil(new Date(), poids, taille, age, sexe
              );
```

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

3.2 Implémentation dans la vue

Dans cette partie, nous ajouterons une méthode `recupProfil()` dans la classe `MainActivity`, permettant d'afficher le dérnier profil inséré lors du lancement de l'application.

1. Ajouter une variable pour le bouton radio du femme pour l'exploiter dans le code ultérieurement.

```
    private RadioButton rdFemme;
    ...
    private void init() {
    ...
    rdFemme = (RadioButton) findViewById(R.id.rdFemme);
    ...
    }
```

2. Créer une méthode **recupProfil()** dans **MainActivity** qui vérifiera si un profil a été sérilaisé à partir de la fonction **recupSerialize()** qui se déclenche dès le lancement de l'applciation. Puis cette méthode remplira les champs de l'interface utilisateur avec les données récupérées.

```
    private void recupProfil() {
        if (controle.getPoids() != null) {
            txtPoids.setText(controle.getPoids().toString());
            txtTaille.setText(controle.getTaille().toString());
            txtAge.setText(controle.getAge().toString());
            rdFemme.setChecked(true);
            if (controle.getSexe() == 1) {
                rdHomme.setChecked(true);
            }
            findViewById(R.id.btnCalc).performClick();
        }
    }
```

3. Appeler la méthode précédente dans la fonction **init()** de la classe **MainActivity** aprés la création de l'instance, pour qu'elle se déclenche lors du lancement de l'application tout de suite de la récupération des données sérialisées et elle affiche alors les données dans l'interface.

```
    private void init() {
        ...
        this.controle = Controle.getInstance(this);
        recupProfil();
        ecouteCalcul();
    }
```

4. Résultat et vérification de l'implémentation

Maintenant il suffit de lancer l'application, d'entrer des données, puis de calculer le résultat. Ensuite, en relançant l'application, les mêmes valeurs saisies avant sa fermeture devraient apparaître. D'après cela, on peut conclure que l'application récupère ces informations avec succès.

4.1 Gestion de base de données d'une application sur Android

Android sauvegarde les données de la base interne d'une application pour une expérience utilisateur plus fluide. Il enregistre ces données et fichiers internes dans un répertoire privé et propriétaire à l'application nommée `databases`. Ce répertoire se trouve généralement à cet emplacement : **/data/data/<nom_du_paquet_de_l'application>/databases**.

4.2 Vérification de l'existance du fichier SQLite

Pour mieux vérifier la persistance des données, on peut consulter le fichier SQLite, nommé **bdTP1.sqlite**, qui est sauvegardé sous le répertoire `databases` mentioné plus haut. Ce répertoire peut être accessible à travers un outil nommé `Device Explorer` sous Android Studio. Pour naviguer à ce fichier et visualiser son contenu, il faut suivre ces étapes :

1. Tout d'abord, ouvrir Android Studio et lancer l'émulateur.
2. Accéder ensuite à l'onglet Device Explorer dans la barre d'outils à partir de *View > Tool Windows > Device Explorer*, comme cette image l'indique.

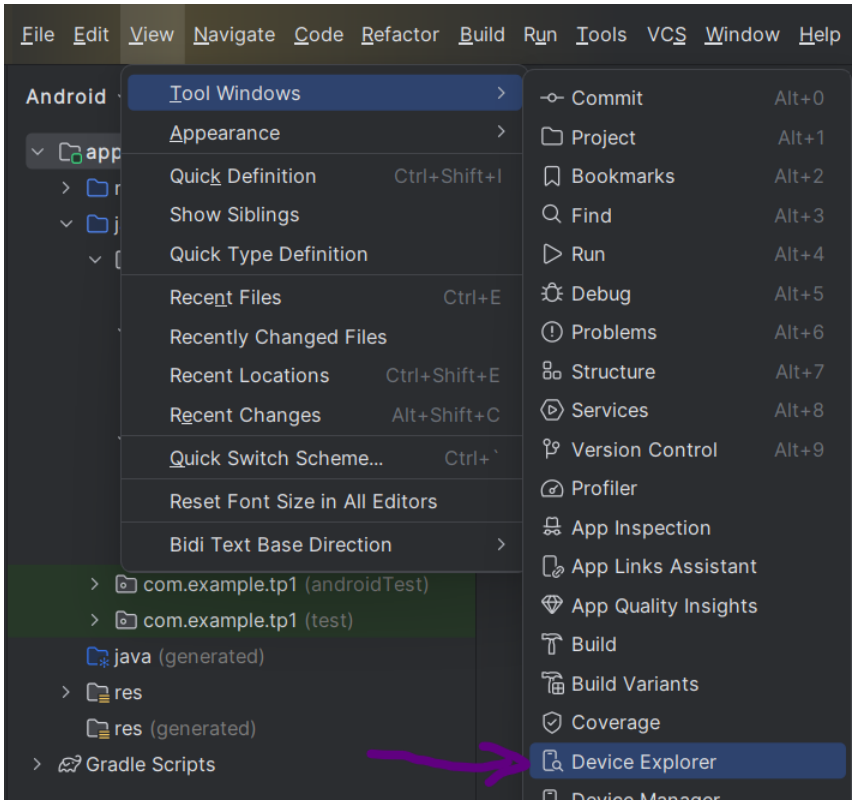

Fig. 1 : Accès à l'outil Device Explorer

3. À ce stade, l'outil doit apparaître en bas à gauche.

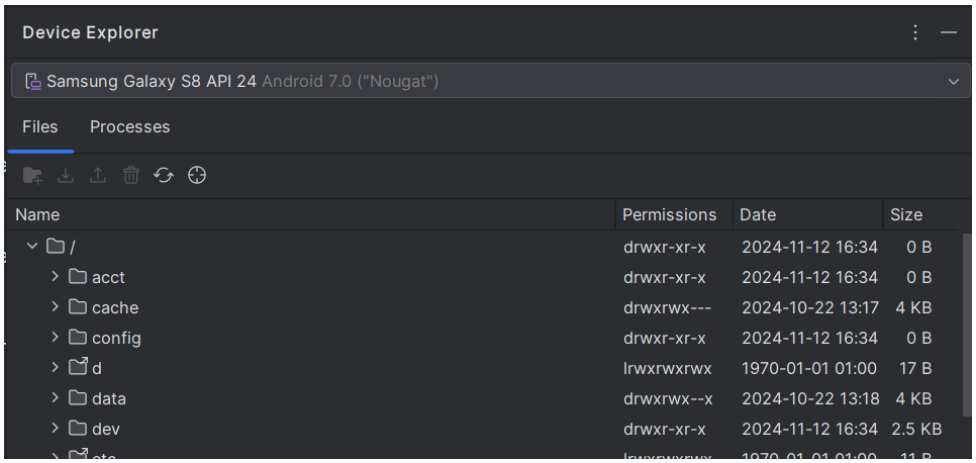

FIG. 2 : l'outil Device Explorer

4. Ensuite, naviguer vers le fichier **bdTP1.sqlite** créé à partir de ce chemin : */data/-data/<nom_du_paquet_de_l'application>/databases*

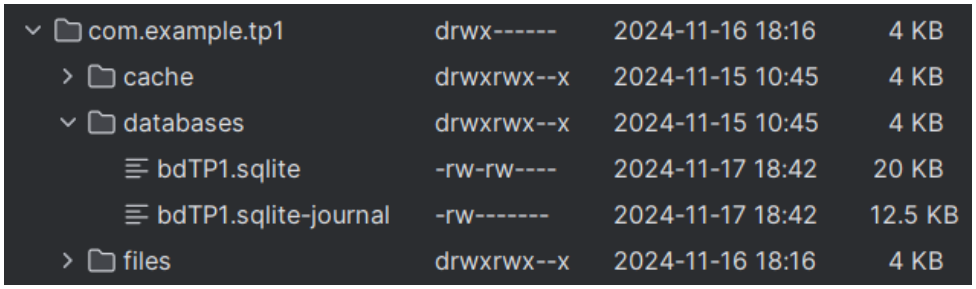

FIG. 3 : Emplacement du fichier

5. Maintenant, télécharger ce fichier SQLite sur l'ordinateur avec un clic droit sur celui-ci, puis en sélectionnant l'option *save as*. Ensuite, il suffit de choisir le dossier de destination souhaité.

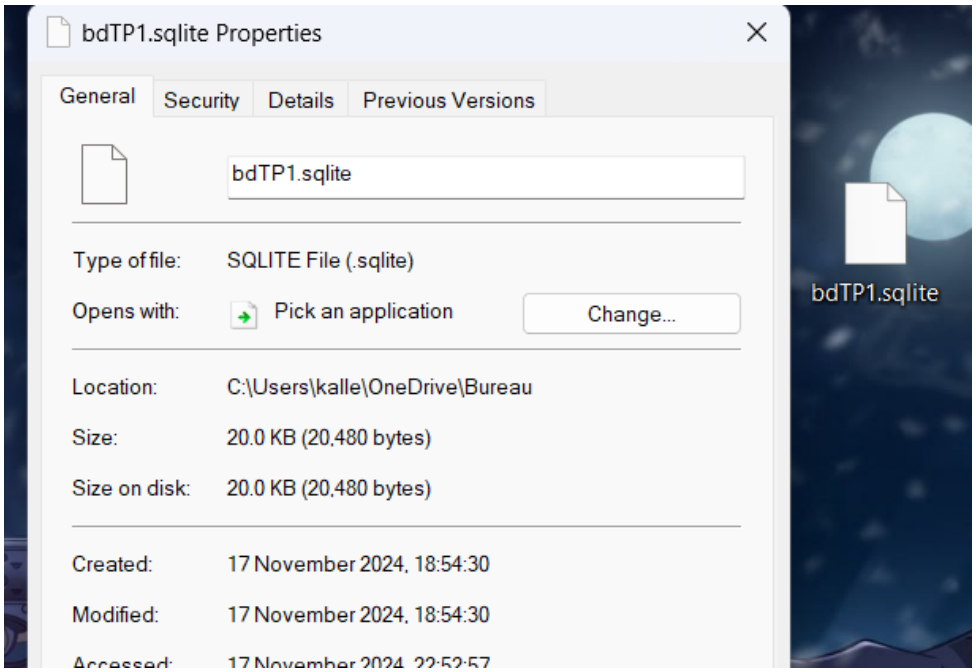

Fig. 4 : Fichier téléchargé en locale

6. Enfin, nous pouvons voir le contenu de ce fichier comme les profils insérés par les utilisateurs et la structure de la table à partir des outils en ligne comme **SQLite Viewer** sous les URLs suivantes :

   - 1er Site : https ://inloop.github.io/sqlite-viewer/
   - 2ème Site : https ://sqliteviewer.app/#

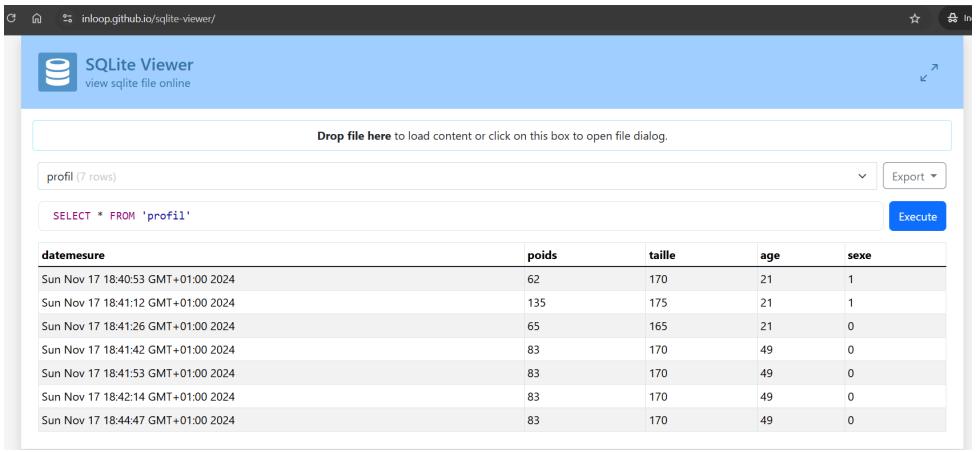

Fig. 5 : Aperçu du fichier SQLite sur le site 1

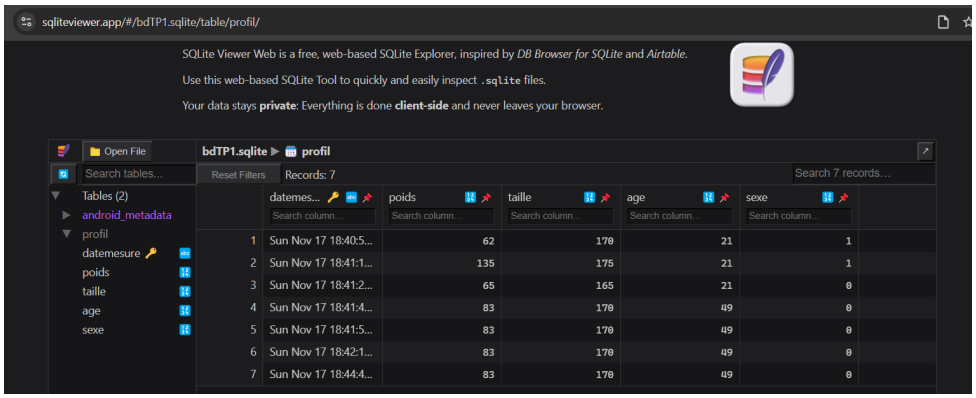

FIG. 6 : Aperçu du fichier SQLite sur le site 2

Conclusion

Ce document couvre de manière détaillée la persistance des données dans une application Android en utilisant SQLite. Cette approche offre une gestion efficace et légère des données utilisateur. Grâce à la création de classes comme `MySQLiteOpenHelper` et `AccesLocal`, nous avons intégré la base de données dans l'architecture de l'application, permettant la sauvegarde et la récupération des profils utilisateurs.