Documentation Technique Bibliotheque

Contexte
Le projet vise à la gestion Utilisateurs , des livres et de leurs auteurs sur l'application afin de gérer une bibliothèque
Objectif du Projet

L'objectif principal est de crée une application mobile pour la rendre plus ergonomique et performante pour une gestion d'une bibliotheque. Cette application est destinée à un usage local car la base de données est locale à l'application,

Sommaire :

-Information sur l'application

- Model

- Vue

-VueModel

-Base de donnée

Information sur l'application

Cette application est crée grâce au Framework Flutter qui fonctionne avec le langage Dart ,L'éditeur de code utilisé est Android Studio

Ce projet à une architecture MVVM (Model, Vue, Vue Model)

Model

Dans les model nous avons User, Livres , Auteur

Avec des notions objet comme des getters, setters, constructeur et leurs propriétés propres à chaqu'un :

Auteur

```
class Auteur {
  late int? _idAuteur = 0;
  late String _nomAuteur = '';

}
```

Livre

```
class Livre {
  late int? _idLivre;
  late String _nomLivre;
  late Auteur _auteur;
  late String? _jacketPath;

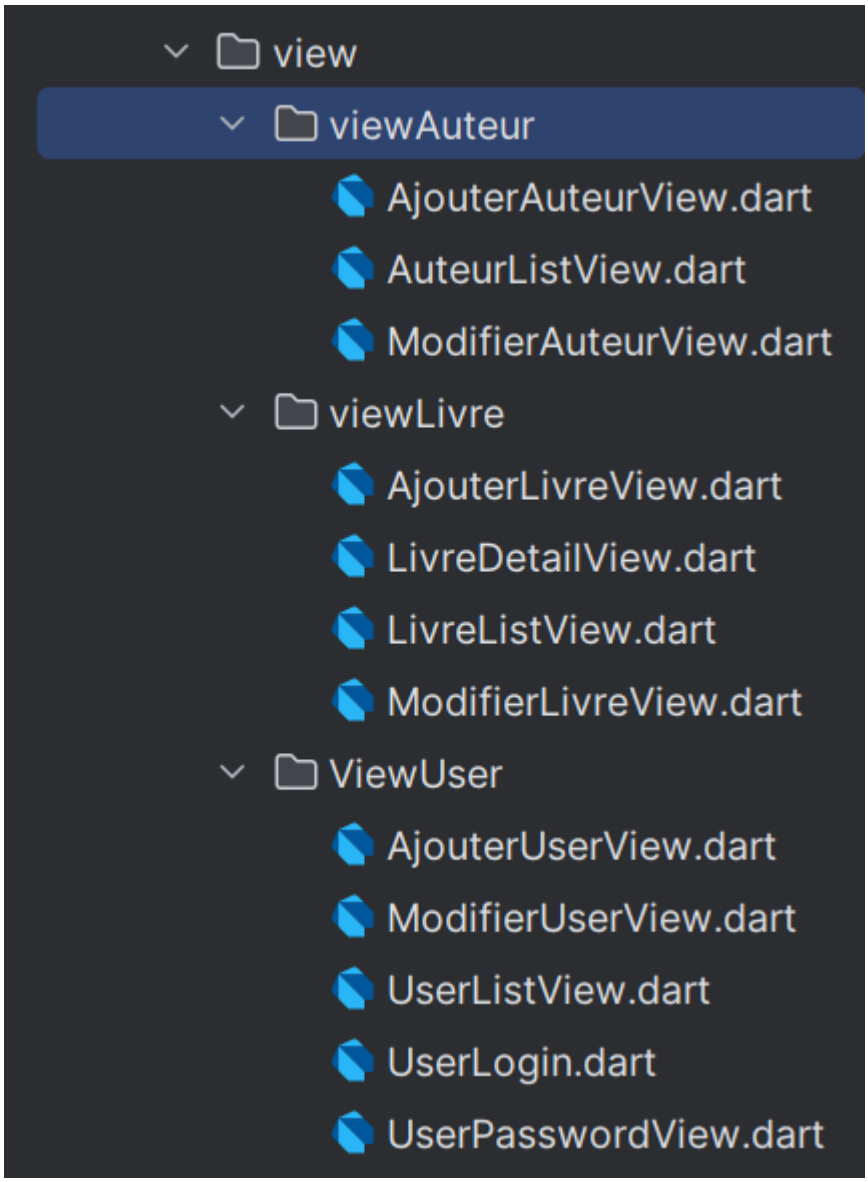
}
```

User

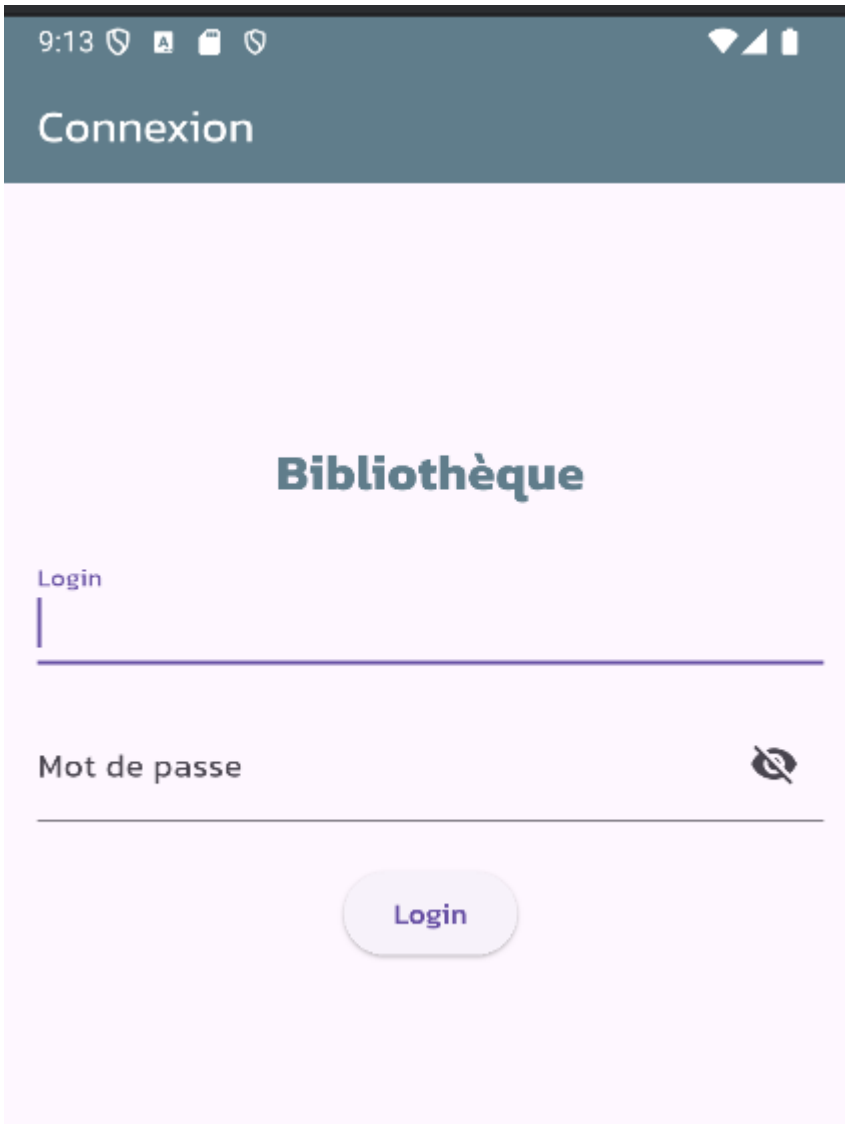
```
class User {
  late int _idUser;
  late String _nomUser;
  late String
_prenomUser;
  late String _loginUser;
  late String _mdpUser;
  late String _roleUser;
```

Vue

Voici mes différentes vues


view
viewAuteur
AjouterAuteurView.dart
AuteurListView.dart
ModifierAuteurView.dart
viewLivre
AjouterLivreView.dart
LivreDetailView.dart
LivreListView.dart
ModifierLivreView.dart
ViewUser
AjouterUserView.dart
ModifierUserView.dart
UserListView.dart
UserLogin.dart
UserPasswordView.dart

Voici Login



UserLogin.dart

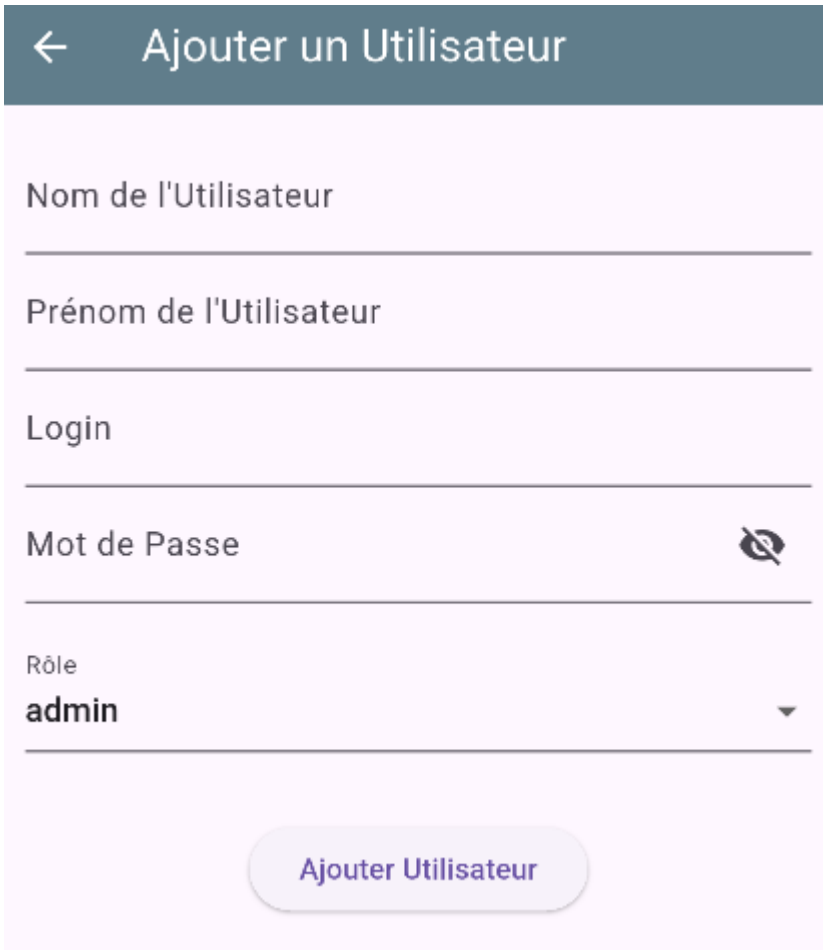
Pour cette vue ci-dessus j’ai crée un widget afin d’afficher le mot de passe ou non

Cette vue permet la connexion à l’application



UserListView.dart

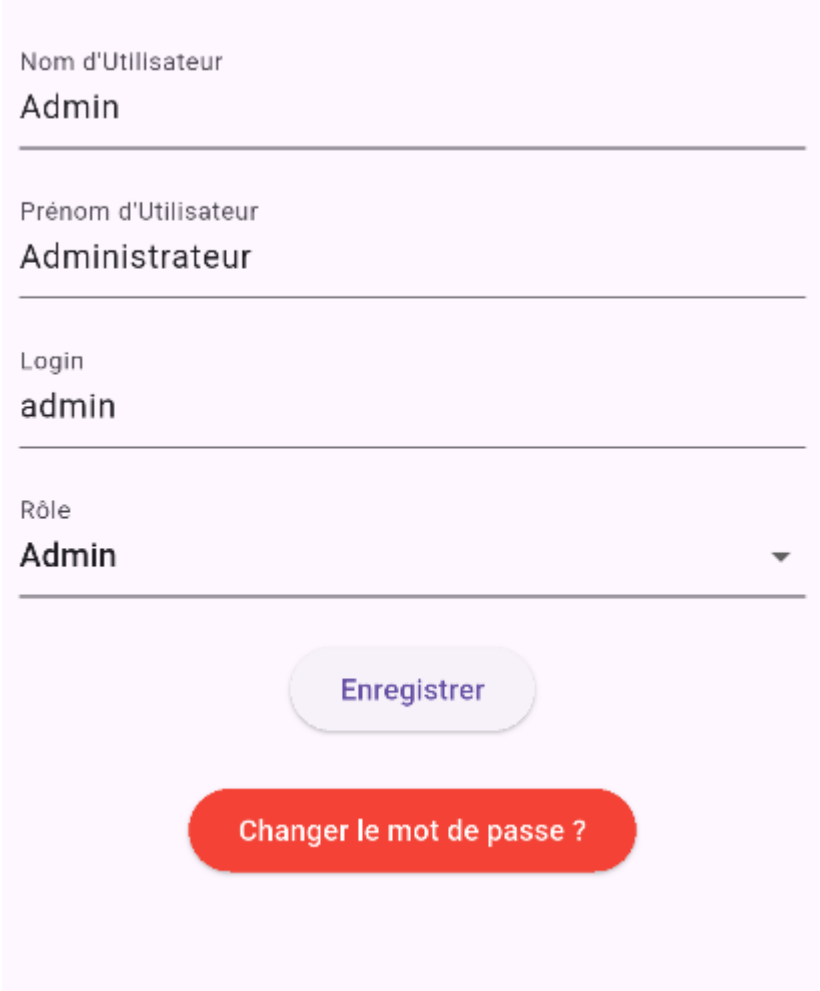
Cette vue sert à afficher les utilisateurs de l’application



AjouterUserView

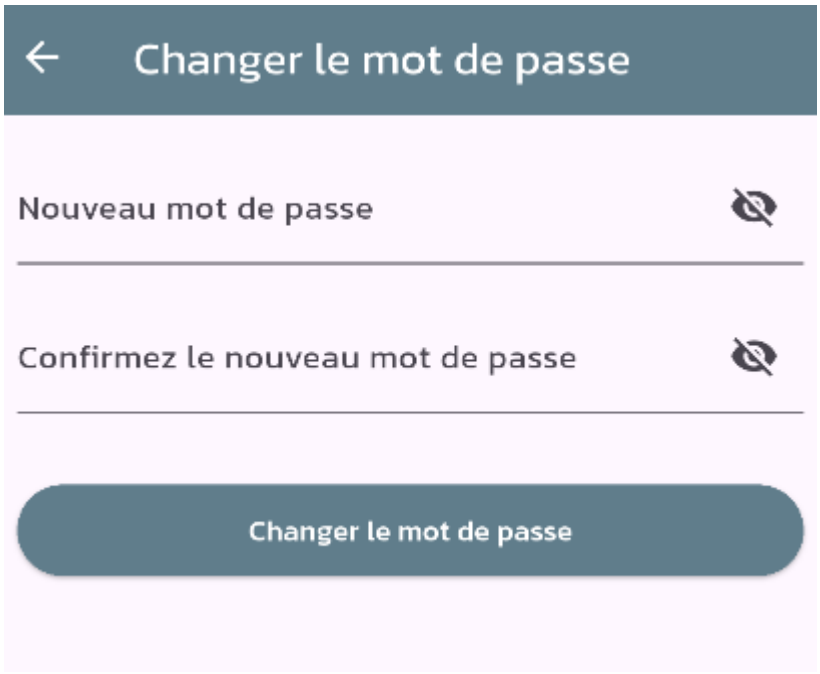
Cette vue sert à ajouter un utilisateur

Le widget vue précédemment à été réutiliser ici



ModifierUserView

Cette vue sert à modifier les informations de l’utilisateur

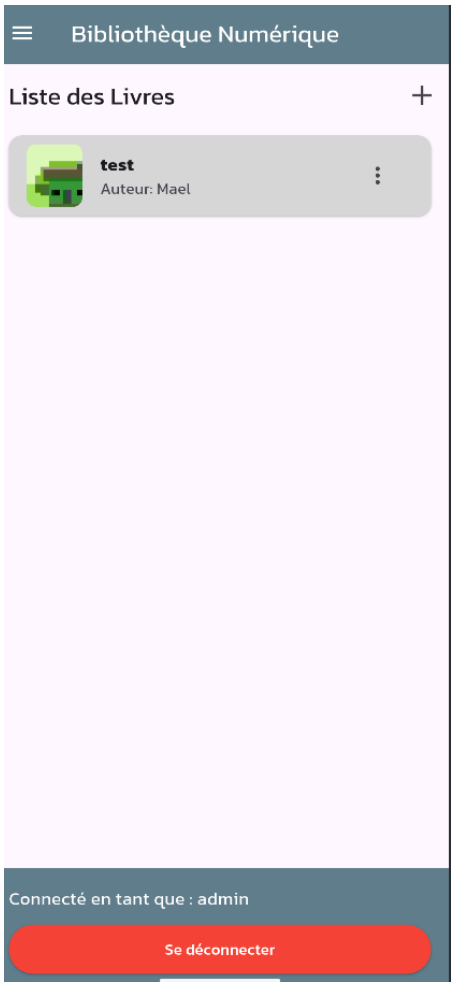


PasswordUserView

Ceci permet de changer le mot de passe du compte sur lequel nous faisons une modification



Voici un widget que j'ai utilisé sur toute l'application afin de vérifier l'intention de l'utilisateur afin d'éviter de potentiel erreur de manipulation



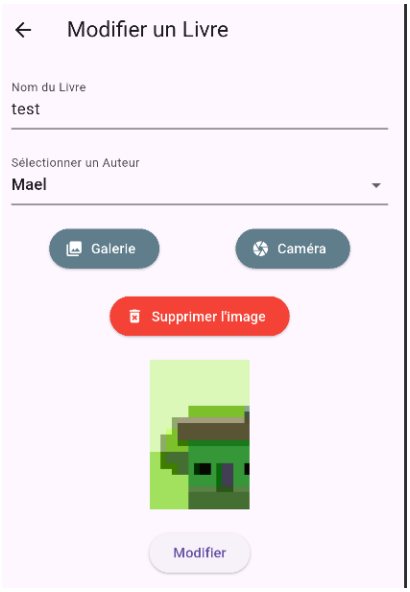
LivreListView

Elle permet de visionner la liste des livres existant dans l’application



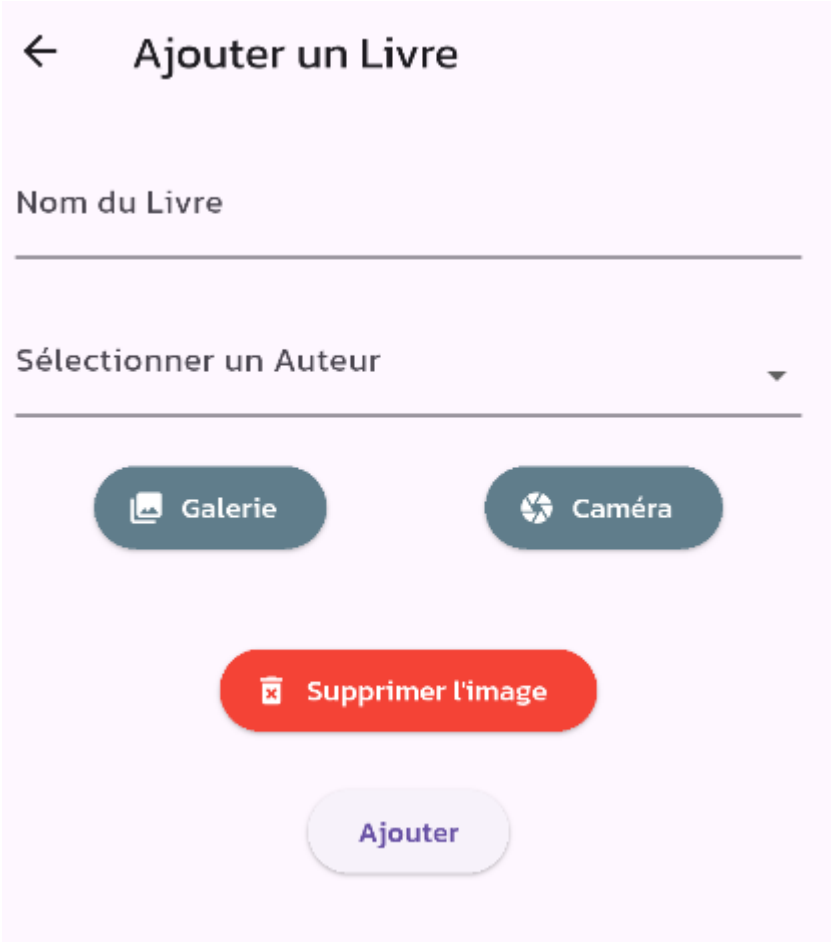
LivreDetailView

Permet d’avoir le détail du livre cliquer



ModifierLivreView

Permet la modification des informations du livre



AjouterLivreView

Cette vue sert a l'ajout de livre le menu deroulant recupere les auteurs qui ont été crée au préalable dans l'application



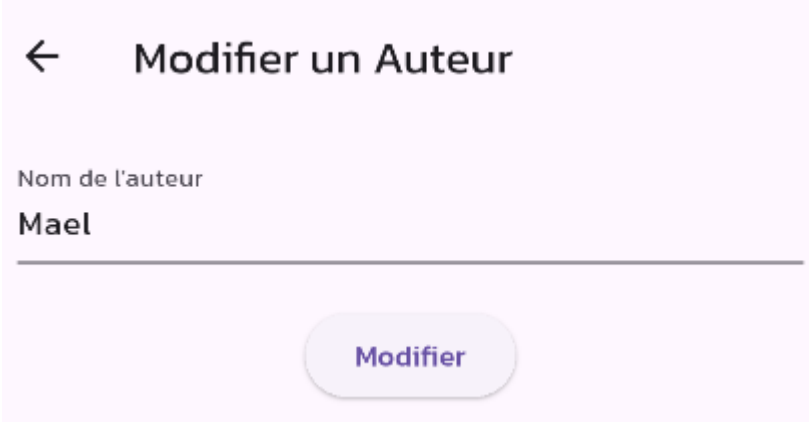
AuteurListView

Cette vue permet d'afficher sous forme de liste les auteurs



AjouterAuteurView

Permet l’ajout d’un auteur dans l’application



ModifierAuteurView

Cette vue permet la modification des Auteurs

VueModel

Les vue Model servent de pont entre le modèle de données (Model) et la vue (View).

- Transforme les données pour la vue.
- Gère l’état et la logique utilisateur.
- Permet une architecture propre et testable.
- Favorise la réactivité de l’interface.

Les vuesmodel sont AuteurViewModel, LivreViewModel,UserViewModel

Voici AuteurViewModel

```
class AuteurViewModel with ChangeNotifier{
  final AuteurDataBase _auteurDb = AuteurDataBase();
  List <Auteur> _auteurs = [];

  List <Auteur> get auteurs => _auteurs;

  Future<void> chargerAuteurs() async {
    final List<Map<String, dynamic>> auteursMap = await
_auteurDb.obtenirTousLesAuteurs();
    _auteurs = auteursMap.map((map) =>
Auteur.fromMap(map)).toList();
    notifyListeners();
  }

  Future <void> ajouterAuteur(String nomAuteur) async {
    await _auteurDb.ajouterAuteur(nomAuteur);
    await chargerAuteurs();
  }
  Future <void> mettreAJourAuteur(int idAuteur, String nomAuteur)
async {
    await _auteurDb.mettreAJourAuteur(idAuteur, nomAuteur);
    await chargerAuteurs();
  }
  Future <void> supprimerAuteur(int idAuteur) async {
    await _auteurDb.supprimerAuteur(idAuteur);
    await chargerAuteurs();

  }
```

Voici

```
 Future<void> chargerAuteurs() async {
   final db = await _dbClient.database;
   final List<Map<String, dynamic>> auteursMap = await
 db.query('AUTEUR');
   _auteurs = auteursMap.map((map) => Auteur.fromMap(map)).toList();
   notifyListeners(); // Notifier les auditeurs de la mise à jour
 }
```

les methode de LivreViewModel

```
Future<void> chargerLivres() async {
  print("Récupération des livres...");
  final livresMap = await _livreDatabase.obtenirTousLesLivres(); // Appel à la méthode du
repository
  print("Livres récupérés: $livresMap");

  // Utiliser Future.wait pour récupérer les livres et leurs auteurs
  final List<Livre?> livres = await Future.wait(
    livresMap.map((livreMap) async {
      final auteur = await _livreDatabase.obtenirAuteurParId(livreMap['idAuteur']);
      // Vérifiez si l'auteur a été trouvé
      if (auteur == null) {
        print("Auteur non trouvé pour l'ID: ${livreMap['idAuteur']}");
        return null; // Retournez null si l'auteur n'est pas trouvé
      } else {
        print("Auteur trouvé: ${auteur.nomAuteur}");
        return Livre.fromMap(livreMap, auteur); // Créer le livre avec l'auteur trouvé
      }
    }),
  );

  // Filtrer les livres pour ne garder que ceux qui ne sont pas null
  _livres = livres.where((livre) => livre != null).cast<Livre>().toList();

  // Notifier les auditeurs de la mise à jour
  notifyListeners();
}
```

```
Future<void> ajouterLivre(String nomLivre, int idAuteur ,String? jacketPath) async {
  final db = await _dbClient.database;
  await db.insert('LIVRE', {
    'nomLivre': nomLivre,
    'idAuteur': idAuteur,
    'jacket': jacketPath,
  });
  // Recharger la liste des livres après ajout
  await chargerLivres();
}
```

```
Future<void> mettreAJourLivre(int idLivre, String nomLivre, int idAuteur , String? jacketPath)
async {
  final db = await _dbClient.database;
  await db.update(
    'LIVRE',
    {'nomLivre': nomLivre, 'idAuteur': idAuteur , 'jacket': jacketPath},
    where: 'idLivre = ?',
    whereArgs: [idLivre],
  );
  // Recharger la liste des livres après mise à jour
  await chargerLivres();
}
```

```
// Méthode pour supprimer un livre
Future<void> supprimerLivre(int idLivre) async {
  final db = await _dbClient.database;
  await db.delete('LIVRE', where: 'idLivre = ?', whereArgs: [idLivre]);
  await chargerLivres();
}
```

```
Future<void> getLivreById(int idLivre) async {
  try {
    // Appel à la méthode getLivreById de la base de données
    Livre? livreRecupere = await _livreDatabase.getLivreById(idLivre);

    // Vérifier si un livre a été trouvé
    if (livreRecupere != null) {
      _livre = livreRecupere;
    } else {
      // Si aucun livre n'est trouvé, assigner null
      _livre = null;
    }

    // Notifier les auditeurs que l'état a changé
    notifyListeners();
  } catch (e) {
    // Gérer l'erreur si une exception se produit
    print("Erreur lors de la récupération du livre: $e");
    _livre = null;
    notifyListeners();
  }
}
```

Voici les methodes de UserViewModel

```
// Charger les utilisateurs
Future<void> chargerUtilisateur() async {
  _isLoading = true;
  notifyListeners();
  try {
    _utilisateur = await _userDatabase.obtenirTousLesUtilisateurs();
  } finally {
    _isLoading = false;
    notifyListeners();
  }
}
```

```
// Ajouter un utilisateur
Future<void> ajouterUser({
  required String nomUser,
  required String prenomUser,
  required String loginUser,
  required String mdpUser,
  required String roleUser,
}) async {
  await _userDatabase.ajouterUser(
    nomUser: nomUser,
    prenomUser: prenomUser,
    loginUser: loginUser,
    mdpUser: mdpUser,
    roleUser: roleUser,
  );
  await chargerUtilisateur();
}
```

```
Future<void> mettreAJourUser({
  required int idUser,
  required String nomUser,
  required String prenomUser,
  required String loginUser,
  required String mdpUser,
  required String roleUser,
}) async {
  await _userDatabase.mettreAJourUser(
    idUser,
    nomUser,
    prenomUser,
    loginUser,
    mdpUser,
    roleUser,
  );
  await chargerUtilisateur();
}
```

```
// Supprimer un utilisateur
Future<void>
supprimerUser(int idUser)
async {
  await
_userDatabase.supprimerUtilis
ateur(idUser);
  await chargerUtilisateur();
}
```

```
Future<String?> login(String login, String password) async {
  if (login.isEmpty || password.isEmpty) {
    return 'Veuillez remplir tous les champs';
  }

  var user = await _userDatabase.verifierLogin(login, password);
  if (user != null) {
    _userName = user.nomUser;
    _userRole = user.roleUser;

    SharedPreferences prefs = await
SharedPreferences.getInstance();
    await prefs.setString('userName', user.nomUser);
    await prefs.setString('userRole', user.roleUser);
    notifyListeners();
    return null;a
  } else {
    return 'Login ou mot de passe incorrect';
  }
}
```

```
// Déconnexion utilisateur
Future<void> logout() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  await prefs.remove('userName');
  await prefs.remove('userRole');
  notifyListeners();
}

// Changer le mot de passe d'un utilisateur
Future<void> changerMotDePasse(int idUser, String
nouveauMotDePasse) async {
  await _userDatabase.changerMotDePasse(idUser,
nouveauMotDePasse);
  notifyListeners();
}
```

C’est le code des vuemodels utiliser dans l’application les commentaires peuvent guider la compréhension des méthodes

Base de donnée

La base de données est crée en local directement dans l’application mobile

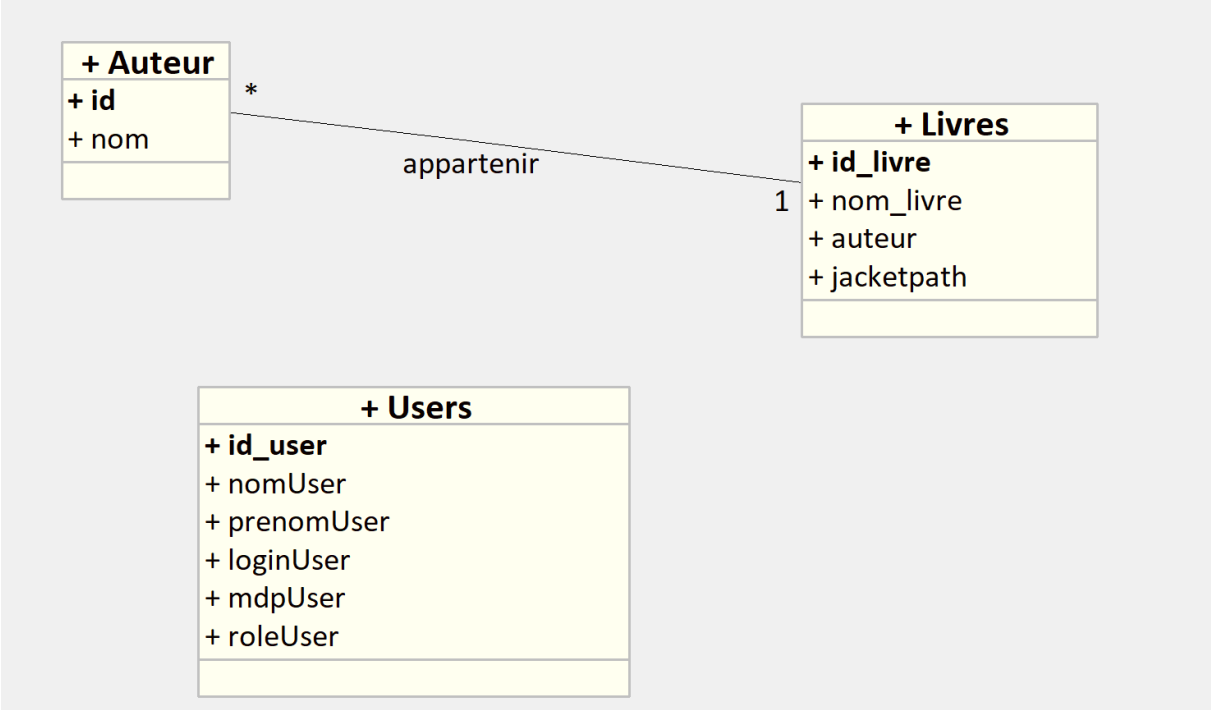
Comme ceci

```
  Future<void> _onCreate(Database db, int version) async {
    // Create `auteur` table
    await db.execute('''
      CREATE TABLE auteur(
        idAuteur INTEGER PRIMARY KEY AUTOINCREMENT,
        nomAuteur TEXT NOT NULL
      )
    ''');

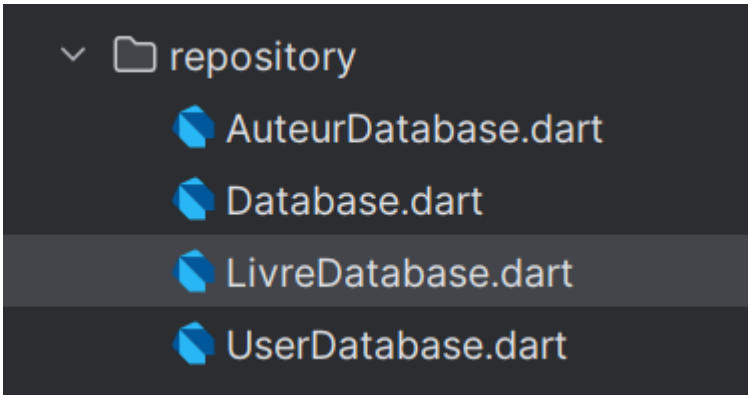
    // Create `livre` table
    await db.execute('''
      CREATE TABLE livre(
        idLivre INTEGER PRIMARY KEY AUTOINCREMENT,
        nomLivre TEXT NOT NULL,
        idAuteur INTEGER,
        jacket TEXT,
        FOREIGN KEY(idAuteur) REFERENCES
auteur(idAuteur)
      )
    ''');

    // Create `users` table
    await db.execute('''
  CREATE TABLE IF NOT EXISTS USERS(
    idUser INTEGER PRIMARY KEY AUTOINCREMENT,
    nomUser TEXT NOT NULL,
    prenomUser TEXT NOT NULL,
    loginUser TEXT NOT NULL UNIQUE,
    mdpUser TEXT NOT NULL,
    roleUser TEXT NOT NULL
  )
''');
```

Voici le modele UML des données



Il y' a une relation entre les livres et les auteurs qui permet de voir l'appartenance d'un livre à un auteur



Mes fichiers pour la gestion de la base de données servent à faire les requetes SQL pour interagir avec les donnée ces méthodes sont utiliser dans le viewModel

Voici les methode de LivreDatabase

```dart
// Méthode pour ajouter un livre
Future<int> ajouterLivre(String nomLivre, int authorId , String? jacketPath) async {
  final db = await _dbClient.database;
  return await db.insert('LIVRE', {
    'nomLivre': nomLivre,
    'idAuteur': authorId,
    'jacket': jacketPath,
  });
}

// Méthode pour récupérer tous les auteurs
Future<List<Map<String, dynamic>>> obtenirTousLesLivres() async {
  final db = await _dbClient.database;
  return await db.query('LIVRE');
}

// Mettre à jour un livre
Future<int> mettreAJourLivre(int livreId, String livreName, int authorId , String? jacketPath) async {
  final db = await _dbClient.database;
  return await db.update(
    'Livre',
    {'nomLivre': livreName, 'idAuteur': authorId , 'jacket': jacketPath},
    where: 'idLivre = ?',
    whereArgs: [livreId],
  );
}
```

```dart
// Supprimer un livre
Future<int> supprimerLivre(int livreId) async {
  final db = await _dbClient.database;
  return await db.delete(
    'LIVRE',
    where: 'idLivre = ?',
    whereArgs: [livreId],
  );
}

// Récupérer un auteur par son ID
// Méthode pour récupérer un auteur par son ID
Future<Auteur?> obtenirAuteurParId(int idAuteur) async {
  final db = await _dbClient.database;

  final List<Map<String, dynamic>> maps = await db.query(
    'AUTEUR', // Nom de la table
    where: 'idAuteur = ?', // Clause WHERE
    whereArgs: [idAuteur], // Argument (ID de l'auteur)
    limit: 1, // On ne veut qu'un seul résultat
  );

  if (maps.isNotEmpty) {
    // Si on trouve un auteur, on retourne un objet Auteur
    return Auteur.fromMap(maps.first);
  } else {
    // Si aucun auteur n'est trouvé, on retourne null
    return null;
  }
}
```

```dart
Future<Livre?> getLivreById(int idLivre) async {
  final db = await _dbClient.database;

  // Récupérer le livre par son ID
  final List<Map<String, dynamic>> maps = await db.query(
    'LIVRE',
    where: 'idLivre = ?',
    whereArgs: [idLivre],
    limit: 1,
  );

  if (maps.isNotEmpty) {
    // Récupérer les données du livre
    var livreMap = maps.first;
    int auteurId = livreMap['idAuteur'];

    // Récupérer l'auteur du livre
    Auteur? auteur = await obtenirAuteurParId(auteurId);

    if (auteur != null) {
      // Retourner l'objet Livre avec l'auteur
      return Livre.fromMap(livreMap, auteur);
    } else {
      // L'auteur n'a pas été trouvé
      return null;
    }
  } else {
    // Aucun livre trouvé
    return null;
  }
```

Voici pour AuteurDatabase

```dart
  Future<int> ajouterAuteur(String nomAuteur) async {
    final db = await _dbClient.database;
    return await db.insert('AUTEUR', {'nomAuteur': nomAuteur});
  }

  Future<List<Map<String, dynamic>>> obtenirTousLesAuteurs() async {
    final db = await _dbClient.database;
    return await db.query('AUTEUR');
  }

  Future<int> mettreAJourAuteur(int idAuteur, String nomAuteur) async {
    final db = await _dbClient.database;
    return await db.update('AUTEUR', {'nomAuteur': nomAuteur}, where: 'idAuteur = ?', whereArgs: [idAuteur]);
  }

  Future<int> supprimerAuteur(int idAuteur) async {
    final db = await _dbClient.database;
    return await db.delete('AUTEUR', where: 'idAuteur = ?', whereArgs: [idAuteur]);
  }

  Future<List<Map<String, dynamic>>> obtenirLesAuteursParOrdreAlphabetique() async {
    final db = await _dbClient.database;
    return await db.query('Auteur', orderBy: 'nomAuteur ASC');
  }
}
```

Et Voici pour UserDatabase

```dart
Future<List<User>> obtenirTousLesUtilisateurs() async {
  final db = await _dbClient.database;
  final List<Map<String, dynamic>> result = await db.query('USERS');

  return result
      .map((userMap) =>
      User(
        idUser: userMap['idUser'],
        nomUser: userMap['nomUser'],
        prenomUser: userMap['prenomUser'],
        loginUser: userMap['loginUser'],
        mdpUser: userMap['mdpUser'],
        roleUser: userMap['roleUser'],
      ))
      .toList();
}
```

```dart
// Ajouter un utilisateur
Future<int> ajouterUser({
  required String nomUser,
  required String prenomUser,
  required String loginUser,
  required String mdpUser, // Mot de passe en clair à hacher
  required String roleUser,
}) async {
  final db = await _dbClient.database;

  // Hachage du mot de passe
  String hashedPassword = BCrypt.hashpw(mdpUser, BCrypt.gensalt());
  // Insert a new user into the database
  return await db.insert('USERS', {
    'nomUser': nomUser,
    'prenomUser': prenomUser,
    'loginUser': loginUser,
    'mdpUser': hashedPassword,
    'roleUser': roleUser,
  });
}
```

Avec le hachage des mots de passe

```dart
Future<int> mettreAJourUser(int idUser, String nomUser, String prenomUser, String loginUser, String mdpUser, String roleUser) async {
  final db = await _dbClient.database;
  Map<String, dynamic> values = {
    'nomUser': nomUser,
    'prenomUser': prenomUser,
    'loginUser': loginUser,
    'roleUser': roleUser,
  };

  // Vérifie si le mot de passe n'est pas vide avant de le hacher
  if (mdpUser.isNotEmpty) {
    String hashedPassword = BCrypt.hashpw(mdpUser, BCrypt.gensalt());
    values['mdpUser'] = hashedPassword;  // Met à jour le mot de passe avec le mot de passe haché
  } else {
    values.remove('mdpUser');  // Ne met pas à jour le mot de passe si c'est vide
  }

  // Mise à jour de l'utilisateur dans la base de données
  return await db.update(
    'USERS',
    values,
    where: 'idUser = ?',
    whereArgs: [idUser],
  );
}
```

```dart
// Delete a user from the USERS table based on their ID
Future<int> supprimerUtilisateur(int idUser) async {
  final db = await _dbClient.database;

  // Delete the user from the 'USERS' table where the ID matches
  return await db.delete(
    'USERS',
    where: 'idUser = ?',
    whereArgs: [idUser],
  );
}
```

```dart
// Verify a user's credentials (login and password)
Future<User?> verifierLogin(String login, String password) async {
  final db = await _dbClient.database;

  final List<Map<String, dynamic>> result = await db.query(
    'USERS',
    where: 'loginUser = ?',
    whereArgs: [login],
  );

  if (result.isNotEmpty) {
    final userMap = result.first;

    // Vérifiez que le mot de passe existe avant de tenter la vérification
    if (userMap['mdpUser'] != null && BCrypt.checkpw(password, userMap['mdpUser'])) {
      return User(
        idUser: userMap['idUser'],
        nomUser: userMap['nomUser'],
        prenomUser: userMap['prenomUser'],
        loginUser: userMap['loginUser'],
        mdpUser: userMap['mdpUser'],
        roleUser: userMap['roleUser'],
      );
    }
  }

  return null;
}
```

```
  // }
  Future<void> changerMotDePasse(int idUser, String nouveauMotDePasse) async {
    final db = await _dbClient.database;
    String hashedPassword = BCrypt.hashpw(nouveauMotDePasse, BCrypt.gensalt());

    await db.update(
      'USERS',
      {'mdpUser': hashedPassword},
      where: 'idUser = ?',
      whereArgs: [idUser],
    );
  }
}
```

Workflow de l’utilisation de l’application

Vidéo de l’utilisation de l’application Mobile

./workflow.mp4