Documentation Technique – Projet Bibliothèque

TABLE DES MATIERES

1. INTRODUCTION

1.1 Présentation du projet

Le projet consiste à créer application mobile du nom de bibliothèque, qui vas permettre une fois installer de gérer des auteurs, des livres et même des comptes utilisateurs. L'application permettra donc de gérer des livres comme dans une bibliothèque avec un système de couverture. La gestion des livres et des auteurs pourra uniquement ce faire si l'on est connecté en admin.

1.2 Objectifs

Les objectifs principaux de ce projet sont :

- Développer une interface utilisateur intuitive pour la gestion des livres, auteurs.
- Mettre en place une base de données pour stocker les informations sur les livres, auteurs et users.
- Assurer une gestion efficace des différents éléments de l'application.

1.3 Public cible

Cette documentation s'adresse aux développeurs travaillant sur le projet ainsi qu'aux contributeurs potentiels.

1.4 Auteur du projet

Ce projet a été réalisé par Goncalves Mathéo dans le cadre de son apprentissage en BTS SIO.

2. OBJECTIFS DU PROJET

- Créer une application réactive et moderne.
- Afficher les éléments de manière attrayante avec des images pour les livres et des descriptions pour auteur.
- Géré les livres, utilisateur et auteurs via des vue adaptés.

3. ARCHITECTURE DU SYSTEME

3.1 Vue d'ensemble

Le système repose sur une architecture mobile-first, développée avec Flutter pour une expérience native et fluide :

- **Frontend & Backend (Full Mobile) :** Flutter avec Dart pour une gestion complète de l'interface et de la logique applicative.

- **Stockage local**: shared_preferences pour sauvegarder des préférences utilisateur et path_provider pour la gestion des fichiers.

- **Gestion de l'état :** provider pour la gestion des états et des interactions utilisateur.
- **Base de données : SQLite via sqflite pour le stockage des données en local.**

4. TECHNOLOGIES UTILISEES

4.1 Frontend & Backend (Flutter)

- **Flutter : Framework mobile open-source permettant le développement d'applications multi-plateformes avec un code unique.**

- **Dart : Langage de programmation utilisé pour coder l'ensemble de l'application.**

4.2 Gestion de l'état

- **Provider :** Bibliothèque permettant la gestion centralisée des états et la communication entre les widgets.

4.3 Base de données

- **SQLite (sqflite) : Base de données locale pour stocker les produits et formules, garantissant un accès rapide et hors-ligne.**4.4 Serveur local

4.4 Stockage local

- **Shared Preferences :** Stockage de données simples comme les préférences utilisateur.

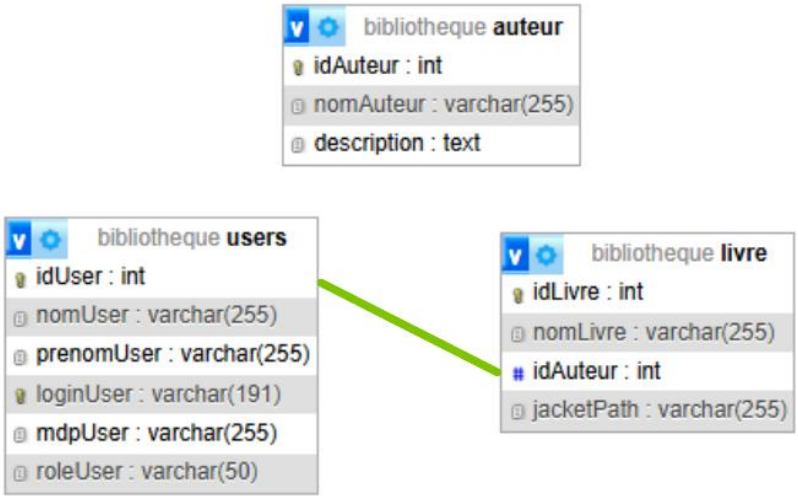
- **Path Provider :** Accès aux répertoires système pour la gestion des fichiers.

5. BASE DE DONNEES ACTUELLE

5.1 Structure de la base de données

La base de données utilisée dans l'application Flutter est nommée **bibliotheque.db**. Elle est implémentée localement à l'aide de la bibliothèque **sqflite**, qui permet la gestion d'une base de données SQLite sur un appareil mobile.

Cette base est conçue pour gérer un système de bibliothèque. Elle regroupe les informations relatives aux **auteurs**, **livres**, et **utilisateurs** de l'application.



Tables Principales

1. Table `auteur`

Cette table stocke les informations de chaque auteur. Chaque enregistrement représente un auteur unique.

- idAuteur INTEGER : Identifiant unique de l'auteur (clé primaire, auto-incrémentée)
- nomAuteur TEXT : Nom de l'auteur (obligatoire)

2. Table `typenourriture`

Cette table contient les livres disponibles dans la bibliothèque. Chaque livre peut être relié à un auteur.

- idLivre INTEGER : Identifiant unique du livre (clé primaire, auto-incrémentée)
- nomLivre TEXT : Nom du livre (obligatoire)
- idAuteur INTEGER : Référence vers l'auteur (clé étrangère vers AUTEUR.idAuteur)
- jacketPath TEXT : Chemin d'accès vers l'image de couverture du livre

3. Table `users`

Cette table gère les utilisateurs de l'application, qu'il s'agisse d'administrateurs ou de simples lecteurs.

- idUser INTEGER : Identifiant unique de l'utilisateur (clé primaire, auto-incrémentée)
- nomUser TEXT : Nom de l'utilisateur (obligatoire)
- prenomUser TEXT : Prénom de l'utilisateur (obligatoire)
- loginUser TEXT : Identifiant de connexion unique (obligatoire)

- mdpUser TEXT : Mot de passe chiffré de l'utilisateur (obligatoire)
- roleUser TEXT : Rôle de l'utilisateur (ex : admin, lecteur)
- email TEXT : Adresse email de l'utilisateur (ajoutée lors de la mise à jour de la base)

6. L'APPLICATION

6.1 Structure de l'application

Dans une application Flutter, l'architecture est souvent divisée en trois parties principales :

- **Vues (Views)** : Ce sont les écrans et widgets qui composent l'interface utilisateur (ex. home_screen.dart), où les utilisateurs interagissent avec l'application.
- **Modèle (Model)** : Il représente les données ou objets utilisés dans l'application (ex. user.dart), souvent sous forme de classes simples.
- **Repository** : Il gère l'accès aux données, qu'elles proviennent d'une API, d'une base de données, ou d'autres sources externes (ex. user_repository.dart).

Le fichier **main.dart** est le point d'entrée de l'application, où l'on configure les routes et lance l'application.

6.2 Model

1. Classe Auteur
Représente un auteur avec :

- Attributs : idAuteur (identifiant), nomAuteur (nom), description (description).
- Méthodes principales :
  - toMap() : Convertit l'auteur en Map pour l'insertion dans la base de données.
  - fromMap() : Crée un auteur à partir d'un Map récupéré de la base de données.

---

2. Classe Livre
Représente un livre avec :

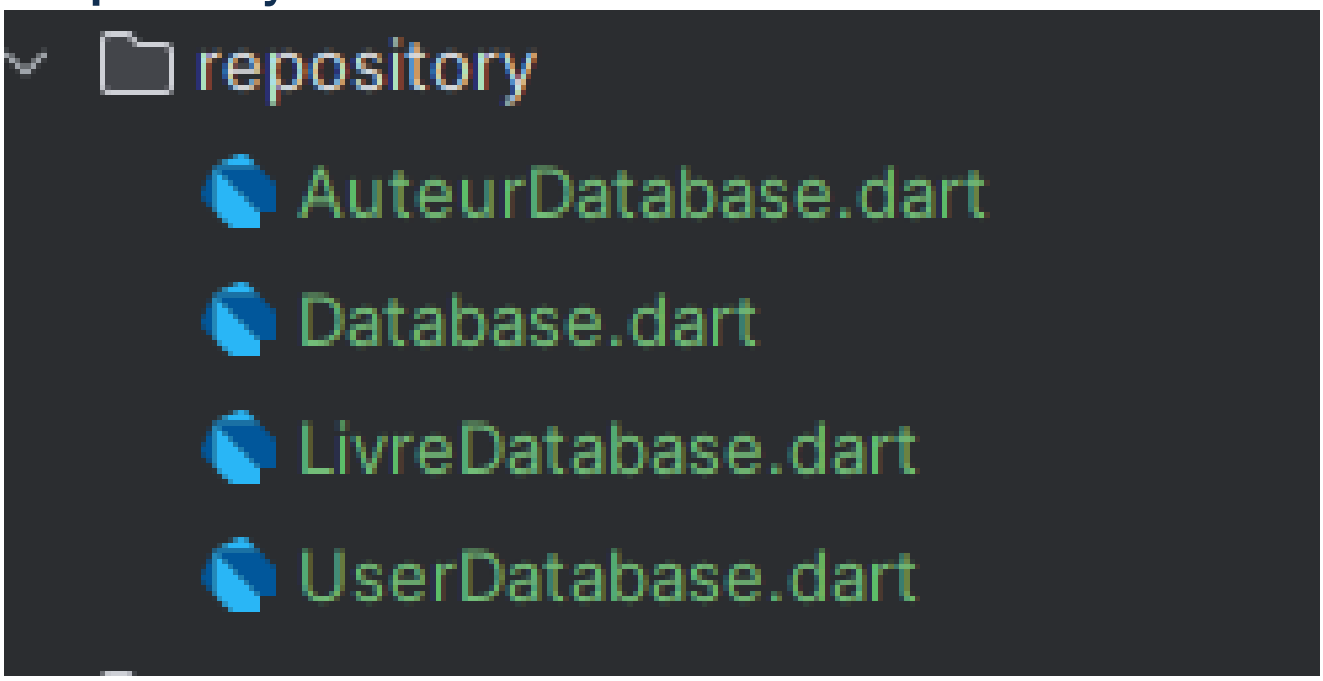
- Attributs : idLivre (identifiant), nomLivre (nom), idAuteur (référence à un auteur), jacketPath (chemin vers l'image de couverture).
- Méthodes principales :
  - toMap() : Convertit le livre en Map pour la base de données.
  - fromMap() : Crée un livre à partir d'un Map récupéré de la base.

---

3. Classe User
Représente un utilisateur avec :

- Attributs : idUser (identifiant), nomUser (nom), prenomUser (prénom), loginUser (identifiant de connexion), mdpUser (mot de passe), roleUser (rôle).
- Méthodes principales :
  - toMap() : Convertit l'utilisateur en Map pour la base de données.
  - fromMap() : Crée un utilisateur à partir d'un Map.
  - Surcharge de l'opérateur == pour comparer les utilisateurs par leur idUser.

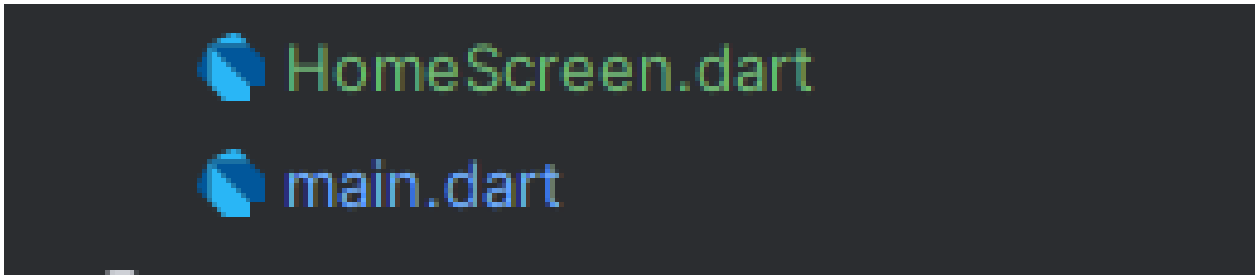
6.3 Repository



1. Le site de **AuteurDatabase** :
    - Cette classe gère les opérations de la base de données pour les auteurs.
    - Elle offre des méthodes pour ajouter, récupérer, mettre à jour, et supprimer des auteurs.
    - Elle utilise un DatabaseHelper pour interagir avec la base de données.
2. **LivreDatabase** :
    - Cette classe gère les opérations de la base de données pour les livres.
    - Elle offre des méthodes pour ajouter, récupérer, mettre à jour, et supprimer des livres.
    - Elle peut également supprimer les livres d'un auteur spécifique.
    - Elle utilise un DatabaseHelper et la classe Livre pour interagir avec la base de données.
3. **UserDatabase** :
    - Cette classe gère les opérations de la base de données pour les utilisateurs.
    - Elle permet de récupérer tous les utilisateurs, d'ajouter un nouvel utilisateur, de mettre à jour un utilisateur, et de supprimer un utilisateur.
    - Elle utilise le hachage bcrypt pour sécuriser les mots de passe.
    - Elle offre également une méthode pour vérifier les identifiants de connexion (login et mot de passe).

Chaque classe interagit avec la base de données via des requêtes SQL (insert, update, delete, query) et contient des méthodes pour gérer les données de manière sécurisée et fonctionnelle.

6.3 Main et HomeScreen



main.dart :

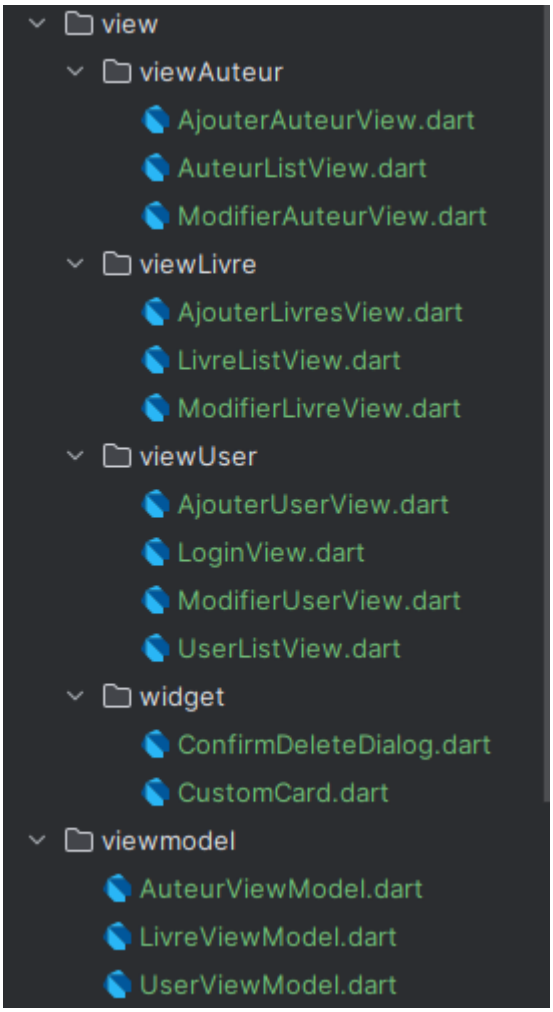
- Initialise la base de données avec un utilisateur administrateur par défaut.
- Configure les ChangeNotifierProvider pour les ViewModel des auteurs, livres et utilisateurs.
- Lance l'application avec un thème bleu et définit les routes pour naviguer entre les vues (LoginView, LivreListView, AuteurListView, UserListView).

HomeScreen.dart :

- Affiche l'écran principal avec une barre de navigation et un tiroir (menu).
- Permet de naviguer entre les vues pour gérer les auteurs, livres, et utilisateurs selon le rôle de l'utilisateur.

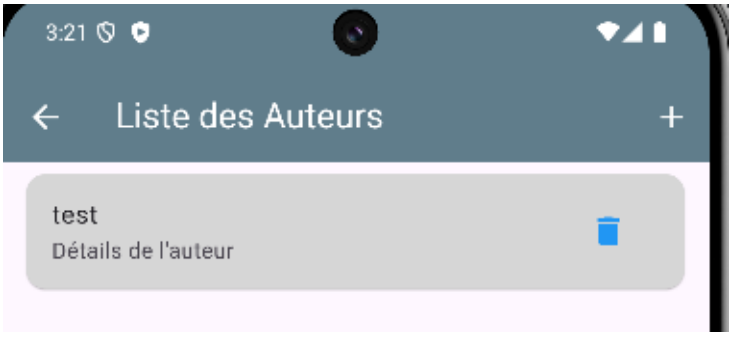
- Inclut une fonction de déconnexion qui redirige vers la page de connexion.

6.3 VUES



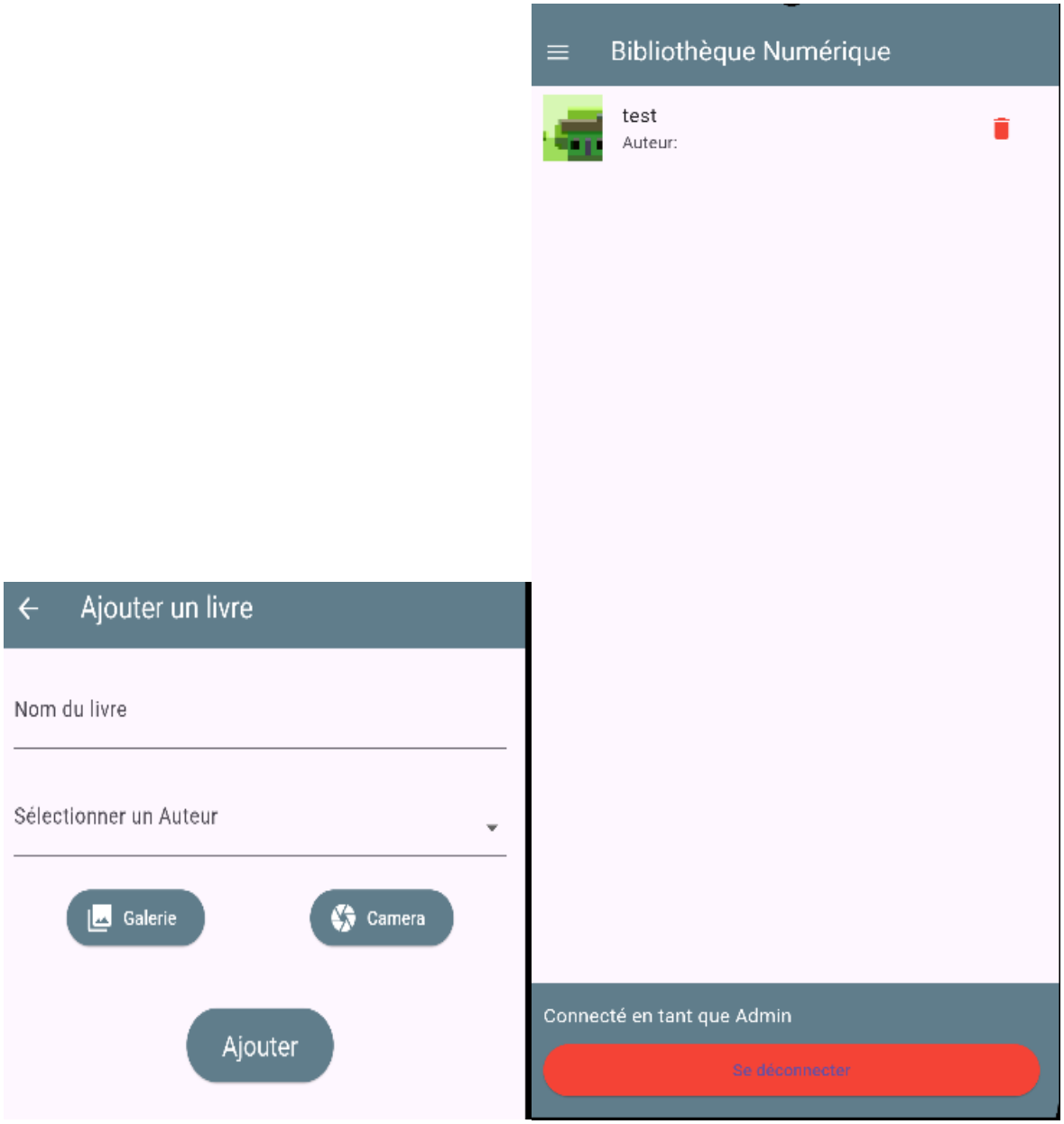
AuteurListView.dart :

- Affiche la liste des auteurs récupérés via AuteurViewModel (avec chargerAuteurs()).
- Permet aux administrateurs d'ajouter, modifier ou supprimer des auteurs.
    - Le bouton "Ajouter" navigue vers AjouterAuteurView si l'utilisateur est un admin.
    - Les boutons "Modifier" et "Supprimer" permettent respectivement de naviguer vers ModifierAuteurView et de confirmer la suppression d'un auteur.
- Affiche une boîte de dialogue d'aide avec une description de la vue.



LivreListView.dart :

- Affiche la liste des livres récupérés via LivreViewModel (avec chargerLivres()).
- Permet aux administrateurs d'ajouter, modifier ou supprimer des livres.
    - Le bouton "Ajouter" navigue vers AjouterLivresView si l'utilisateur est un admin.
    - Les boutons "Modifier" et "Supprimer" permettent respectivement de naviguer vers ModifierLivreView et de confirmer la suppression d'un livre.
- Affiche l'image de couverture du livre (si disponible) et l'auteur associé à chaque livre via AuteurViewModel.


Bibliothèque Numérique
test
Auteur:
Connecté en tant que Admin
Se déconnecter
Ajouter un livre
Nom du livre
Sélectionner un Auteur
Galerie
Camera
Ajouter

LoginView.dart :

- Affiche un formulaire de connexion avec des champs pour le login et le mot de passe.
- Utilise UserViewModel pour authentifier l'utilisateur avec la méthode login().
    - Si la connexion échoue, un message d'erreur est affiché via SnackBar.
    - En cas de succès, la vue redirige vers HomeScreen avec les informations d'utilisateur (nom et rôle).
- Permet de naviguer vers AjouterUserView pour la création de compte en cas de besoin.