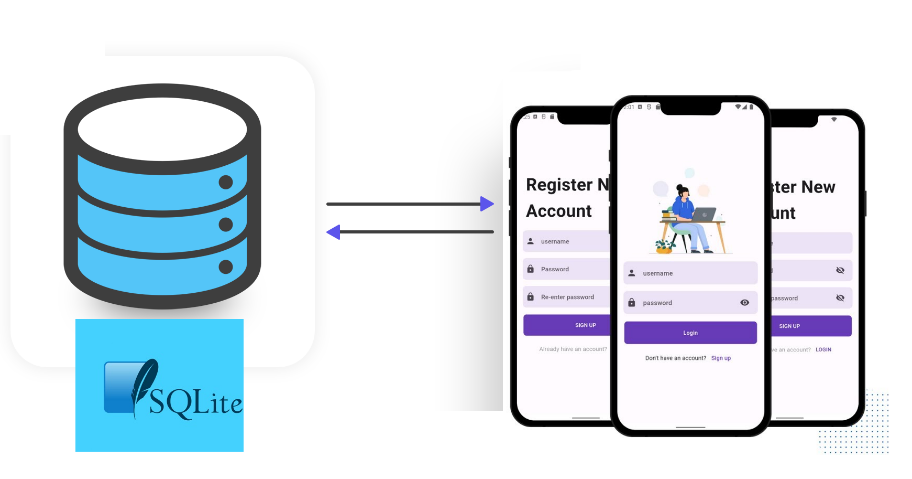
**Atelier 8 Flutter**

**Authentification en utilisant une base de données Sqlite**



# Introduction

SQLite est un moteur de base de données léger qui fournit un système de gestion de base de données relationnelle dans un package petit et efficace.

La persistance des données est un aspect essentiel de nombreuses applications mobiles, et c'est là qu'intervient SQLite.

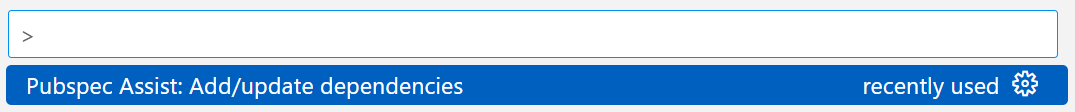
SQLite est un choix idéal pour le développement d'applications mobiles car il est léger, rapide et offre un support transactionnel. Il est également largement pris en charge sur de nombreuses plateformes Android, iOS et Web.

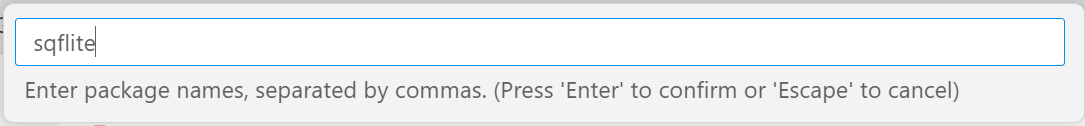
# Installation de dépendances

Pour commencer à utiliser SQLite et Flutter, vous devez ajouter le package **sqflite** à votre projet. Le package sqflite est un plugin Flutter qui fournit une implémentation SQLite légère et efficace pour le développement d'applications mobiles.

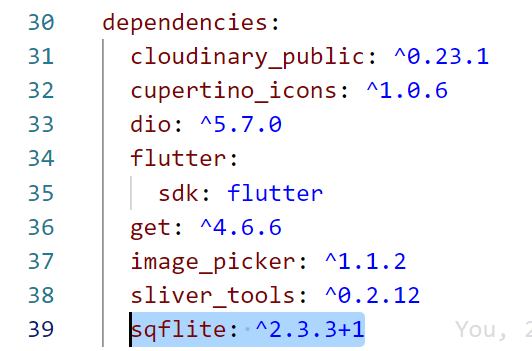
Site : <https://pub.dev/packages/sqflite>

CTRL SHIFT P



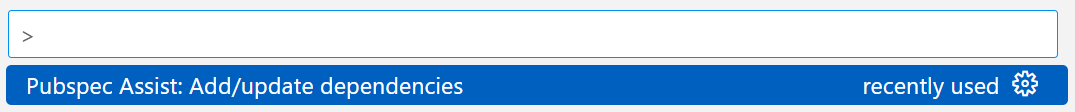


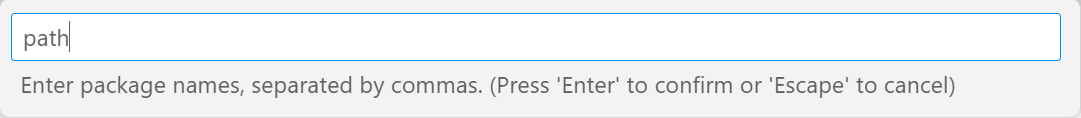




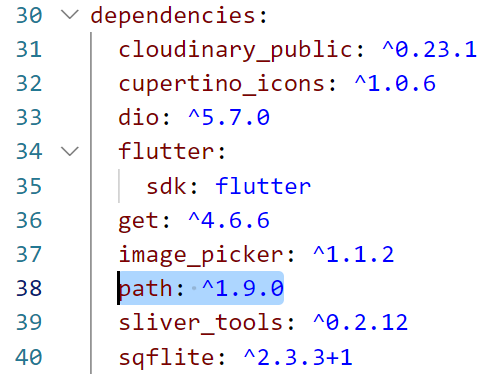
Le package **path** fournit des opérations courantes pour manipuler les chemins : joindre, diviser, normaliser, etc.

CTRL SHIFT P



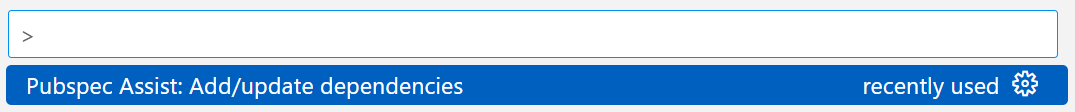


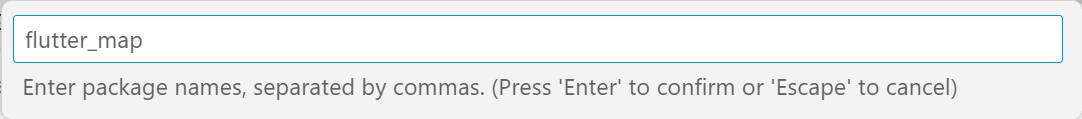




**flutter-map** est un package open source pour Flutter qui fournit un widget de carte hautement personnalisable. Avec Flutter Map, les développeurs peuvent créer des cartes interactives dans leurs applications Flutter, y compris des fonctionnalités telles que des marqueurs, des polygones, etc.

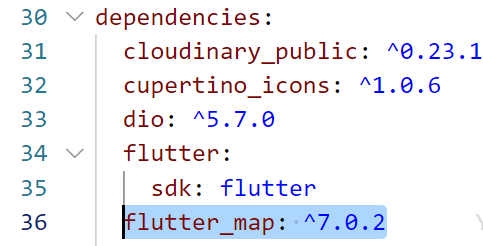
CTRL SHIFT P





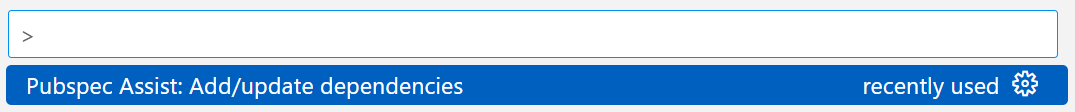
Site : <https://docs.fleaflet.dev/>

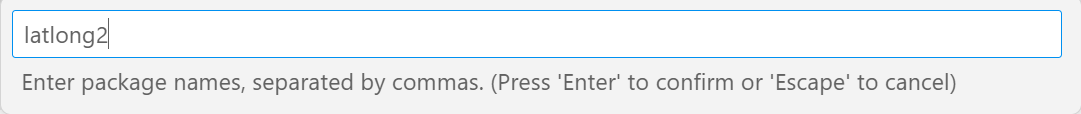




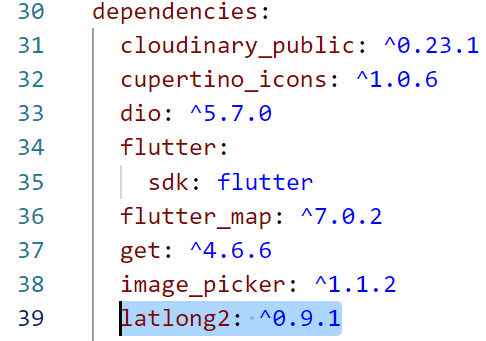
**latlong2** : Bibliothèque légère pour le calcul courant de la latitude et de la longitude.

CTRL SHIFT P









# Modèle d'utilisateur

Créez un modèle pour représenter les utilisateurs.



class User{

  int? id;

  String username;

  String password;

  User({this.id, required this.username, required this.password});

  Map<String, dynamic> toMap() {

    return {

      'id': id,

      'username': username,

      'password': password,

    };

  }

  factory User.fromMap(Map<String, dynamic> map) {

    return User(

      id: map['id'],

      username: map['username'],

      password: map['password'],

    );

  }

}

# Data Source Locale avec Sqflite (SQLite)

Créer le fichier qui s’agit d’un Data Source pour gérer directement la base de données SQLite. Ce code utilisera SQLite avec sqflite pour gérer une base de données locale où les informations des utilisateurs peuvent être stockées et récupérées. La classe utilise le pattern Singleton pour garantir qu'il n'y ait qu'une seule instance de UserLocalDataSource, et des méthodes sont fournies pour insérer et récupérer des utilisateurs.

Créer le fichier /lib/data/datasource/localdatasource/user\_local\_data\_source.dart



import 'package:myflutterapplication/data/datasource/models/users.model.dart';

import 'package:sqflite/sqflite.dart';

import 'package:path/path.dart';

class UserLocalDataSource {

  static final UserLocalDataSource \_instance = UserLocalDataSource.\_internal();

  static Database? \_database;

  factory UserLocalDataSource() {

    return \_instance;

  }

  UserLocalDataSource.\_internal();

  Future<Database> get database async {

    if (\_database != null) return \_database!;

    \_database = await \_initDatabase();

    return \_database!;

  }

  Future<Database> \_initDatabase() async {

    String path = join(await getDatabasesPath(), 'app.db');

    return await openDatabase(path, version: 1, onCreate: \_onCreate);

  }

  Future \_onCreate(Database db, int version) async {

    await db.execute(

      '''CREATE TABLE users(id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, password TEXT)''',

    );

  }

Future<Map<String, dynamic>> insertUser(User user) async {

  print(user);

  final db = await database;

  // Insérer l'utilisateur et récupérer l'ID inséré

  final int userId = await db.insert('users', {

    'username': user.username,

    'password': user.password,

  });

  // Retourner une Map avec les informations de l'utilisateur, y compris l'ID

  return {

    'id': userId,

    'username': user.username,

    'password': user.password,

  };

}

  Future<User?> getUser(String username, String password) async {

    final db = await database;

    final result = await db.query(

      'users',

      where: 'username = ? AND password = ?',

      whereArgs: [username, password],

    );

    if (result.isNotEmpty) {

      return User(

        id: result.first['id'] as int,

        username: result.first['username'] as String,

        password: result.first['password'] as String,

      );

    }

    return null;

  }

}

**Explication**

path : Ce package est utilisé pour manipuler les chemins de fichiers de manière cross-platform, comme l'obtention du chemin de la base de données.

La classe UserLocalDataSource utilise le pattern Singleton pour s'assurer qu'il n'y ait qu'une seule instance de cette classe dans toute l'application.

class UserLocalDataSource {

static final UserLocalDataSource \_instance = UserLocalDataSource.\_internal();

static Database? \_database;

factory UserLocalDataSource() { return \_instance; }

UserLocalDataSource.\_internal();

}

\_instance : Une seule instance de UserLocalDataSource est créée ici.

factory UserLocalDataSource() : Cette méthode retourne toujours la même instance de UserLocalDataSource, ce qui garantit que la classe est un singleton.

\_internal() : Ce constructeur privé est utilisé pour empêcher la création de multiples instances de la classe.

La méthode database crée et retourne une instance de la base de données si elle n'existe pas encore, ou retourne l'instance existante si elle est déjà initialisée.

Future<Database> get database async {

if (\_database != null) return \_database!;

\_database = await \_initDatabase();

return \_database!;

}

La base de données est initialisée par \_initDatabase() si elle n'existe pas encore.

getDatabasesPath() : Obtient le chemin où les bases de données SQLite sont stockées.

openDatabase() : Ouvre la base de données à un chemin donné (ici 'app.db'), avec la version 1.

La méthode \_onCreate est appelée lors de la première création de la base de données.

Future<Database> \_initDatabase() async {

String path = join(await getDatabasesPath(), 'app.db');

return await openDatabase(path, version: 1, onCreate: \_onCreate);

}

Lorsque la base de données est créée, une table users est également créée avec les colonnes id (clé primaire auto-incrémentée), username, et password.

insertUser() : Cette méthode permet d'insérer un nouvel utilisateur dans la base de données. Le nom d'utilisateur et le mot de passe sont stockés dans la table users.

db.insert() : Insère un nouvel utilisateur dans la table et retourne l'ID généré pour cet utilisateur.

La méthode retourne une map contenant les informations de l'utilisateur, y compris l'ID.

getUser() : Cette méthode récupère un utilisateur en fonction du nom d'utilisateur et du mot de passe fournis.

db.query() : Exécute une requête pour chercher un utilisateur dans la table users avec les arguments username et password.

Si un utilisateur correspondant est trouvé, un objet User est retourné. Si aucun utilisateur n'est trouvé, null est retourné.

# Le Repository

Créer le fichier /lib/data/repositories/user.repository.dart



import 'package:myflutterapplication/data/datasource/localdatasource/user\_local\_data\_source.dart';

import 'package:myflutterapplication/data/datasource/models/users.model.dart';

class UserRepository  {

  final UserLocalDataSource localDataSource;

  UserRepository({required this.localDataSource});

  Future<bool> authenticate(String username, String password) async {

    final user = await localDataSource.getUser(username, password);

    return user != null;

  }

  Future<Map> registerUser(String username, String password) async {

  // Insère l'utilisateur et récupère les informations retournées sous forme de Map

  final user = await localDataSource.insertUser(User(id: 0, username: username, password: password));

print(user);

  return user;

}

}

**Explication**

Ce code représente un repository qui fait l'interface entre la source de données locale (gérée par UserLocalDataSource) et les opérations liées aux utilisateurs dans l'application. Ce repository permet de simplifier les interactions avec la base de données en encapsulant la logique de gestion des utilisateurs (authentification et enregistrement).

authenticate() : Cette méthode prend un nom d'utilisateur et un mot de passe, puis utilise localDataSource.getUser() pour chercher un utilisateur dans la base de données.

Si l'utilisateur est trouvé, la méthode retourne true, sinon false. Cela permet de savoir si l'authentification a réussi ou échoué.

await localDataSource.getUser(username, password) : Appelle la méthode dans UserLocalDataSource pour vérifier si un utilisateur avec ces informations existe dans la base de données.

registerUser() : Cette méthode permet de créer un nouvel utilisateur. Elle prend un nom d'utilisateur et un mot de passe, crée un objet User avec un id initialisé à 0 (car il sera auto-incrémenté lors de l'insertion en base de données), et appelle localDataSource.insertUser() pour insérer l'utilisateur dans la base de données.

await localDataSource.insertUser() : Insère l'utilisateur dans la base de données via la méthode du UserLocalDataSource et retourne les informations de l'utilisateur sous forme de Map.

# L’Entité

Créer le fichier /lib/domain/entities/user.entity.dart



class UserEntity {

  final int id;

  final String username;

  final String password;

  UserEntity({required this.id, required this.username, required this.password});

}

# UseCase pour l'authentification

Créer le fichier /lib/domain/usecases/user.usecase.dart



import 'package:myflutterapplication/data/repositories/user.repository.dart';

import 'package:myflutterapplication/domain/entities/user.entity.dart';

class AuthenticateUserUseCase{

  final UserRepository \_repository;

   AuthenticateUserUseCase({required UserRepository repository})

      : \_repository = repository;

  Future<bool> call(String username, String password) {

    return \_repository.authenticate(username, password);

  }

 Future<UserEntity?> register(String username, String password) async {

  final result = await \_repository.registerUser(username, password);

  // Vérifier que le map n'est pas null ou vide

  if (result.isNotEmpty) {

    return UserEntity(

      id: result['id'] ?? "",

      username: result['username'] ?? "",

      password: result['password'] ?? "",

    );

  }

  return null;

}

}

**Explication**

Ce code fait partie de la couche Use Case dans une architecture de type Clean Architecture. Il encapsule la logique métier associée à l'authentification et à l'enregistrement d'un utilisateur. Le but d'un use case est d'exécuter une action spécifique dans le domaine de l'application, ici l'authentification et l'enregistrement d'un utilisateur, en interagissant avec le repository.

call() : Cette méthode sert à authentifier un utilisateur. Elle délègue la logique d'authentification à la méthode authenticate() du UserRepository.

Elle retourne un Future<bool> qui indique si l'utilisateur a été authentifié avec succès ou non.

Utiliser call() permet à l'instance de AuthenticateUserUseCase d'être appelée directement comme une fonction.

register() : Cette méthode est utilisée pour enregistrer un nouvel utilisateur. Elle fait appel à la méthode registerUser() du repository pour insérer un utilisateur dans la base de données.

await \_repository.registerUser() : La méthode attend que le repository enregistre l'utilisateur et renvoie les informations sous forme de Map.

Vérification de la Map : Si la Map retournée par le repository contient des données (n'est pas vide), on crée une instance de UserEntity à partir des données reçues (id, username, et password). Si la Map est vide, la méthode retourne null.

UserEntity : Représente l'utilisateur au niveau de la couche domaine. C'est une abstraction qui est indépendante de la façon dont les données sont stockées ou obtenues.

# Le controller

Créer le fichier /lib/presentation/controllers/user.controller.dart



import 'package:get/get.dart';

import 'package:myflutterapplication/domain/usecases/user.usecase.dart';

class AuthController extends GetxController {

  final AuthenticateUserUseCase \_userUseCase;

  AuthController({required AuthenticateUserUseCase userUseCase})

      : \_userUseCase = userUseCase;

  var isAuthenticated = false.obs;

 Future<bool> login(String username, String password) async {

  final res = await \_userUseCase.call(username, password);

  isAuthenticated.value = res;

  return res; // Retourner le succès ou l'échec

}

  Future<void> register(String username, String password) async {

    await \_userUseCase.register(username, password);

  }

}

**Explication**

Le contrôleur interagit avec le use case AuthenticateUserUseCase pour authentifier et enregistrer des utilisateurs, tout en maintenant l'état de l'authentification observable dans l'interface utilisateur.

isAuthenticated : Cette propriété est un observable (obs) de type bool, initialisé à false. GetX permet d'observer cette variable et de réagir automatiquement à tout changement dans l'interface utilisateur (par exemple, mise à jour de l'UI lorsqu'un utilisateur est authentifié).

En utilisant .obs, GetX crée un réactif. Dès que la valeur change, toutes les parties de l'interface qui observent isAuthenticated seront mises à jour.

login() : Cette méthode permet de gérer le processus de connexion d'un utilisateur.

Elle appelle la méthode call() du use case AuthenticateUserUseCase pour vérifier les informations d'identification (nom d'utilisateur et mot de passe).

isAuthenticated.value = res : Si l'authentification réussit, la valeur de isAuthenticated est mise à jour (observable), ce qui déclenche une mise à jour des parties de l'interface qui écoutent cet observable.

return res; : Retourne le résultat de l'authentification pour que la méthode appelante puisse en faire usage (par exemple, affichage d'un message de succès ou d'échec).

register() : Cette méthode permet d'enregistrer un nouvel utilisateur.

Elle appelle la méthode register() du use case AuthenticateUserUseCase pour ajouter un nouvel utilisateur avec un nom d'utilisateur et un mot de passe dans la base de données.

Cette méthode ne retourne rien et n'a pas besoin de mettre à jour l'état observable dans ce cas.

# Le Widget de register

Créer



import 'package:flutter/material.dart';

import 'package:get/get.dart';

import 'package:myflutterapplication/presentation/controllers/user.controller.dart';

class Registerform extends StatefulWidget {

  const Registerform({super.key});

  @override

  State<Registerform> createState() => \_Registerform();

}

class \_Registerform extends State<Registerform> {

  // Initialisation du contrôleur

  final AuthController \_controller = Get.put(AuthController(

    userUseCase: Get.find(),

  ));

  final \_formKey = GlobalKey<FormState>();

  late TextEditingController \_usernameController;

  late TextEditingController \_passwordController;

  late TextEditingController \_passwordController2;

  // show the password or not

  bool \_isObscure = true;

  bool \_isObscure2 = true;

  @override

  void initState() {

    super.initState();

    \_usernameController = TextEditingController();

    \_passwordController = TextEditingController();

    \_passwordController2 = TextEditingController();

  }

  @override

  Widget build(BuildContext context) {

    return SingleChildScrollView(

      child: Form(

        key: \_formKey,

        child: Column(

          crossAxisAlignment: CrossAxisAlignment.start,

          children: <Widget>[

            TextFormField(

              controller: \_usernameController,

              decoration: const InputDecoration(

                  icon: Icon(Icons.person),

                  hintText: "Username",

                  labelText: "Username"),

              validator: (value) {

                if (value!.isEmpty) {

                  return 'Please enter the username';

                }

                return null;

              },

            ),

            TextFormField(

              obscureText: \_isObscure,

              controller: \_passwordController,

              decoration: InputDecoration(

                icon: const Icon(Icons.key\_rounded),

                hintText: "Password",

                labelText: "Password",

                // this button is used to toggle the password visibility

                suffixIcon: IconButton(

                    icon: Icon(

                        \_isObscure ? Icons.visibility : Icons.visibility\_off),

                    onPressed: () {

                      setState(() {

                        \_isObscure = !\_isObscure;

                      });

                    }),

              ),

              validator: (value) {

                if (value!.isEmpty) {

                  return 'Please enter the password';

                }

                return null;

              },

            ),

            TextFormField(

              obscureText: \_isObscure2,

              controller: \_passwordController2,

              decoration: InputDecoration(

                icon: const Icon(Icons.key\_outlined),

                hintText: "Retape Password",

                labelText: "Retape Password",

                // this button is used to toggle the password visibility

                suffixIcon: IconButton(

                    icon: Icon(

                        \_isObscure2 ? Icons.visibility : Icons.visibility\_off),

                    onPressed: () {

                      setState(() {

                        \_isObscure2 = !\_isObscure2;

                      });

                    }),

              ),

              validator: (value) {

                if (value != \_passwordController.text) {

                  return "Passwords do not match";

                }

                return null;

              },

            ),

            Center(

              child: ElevatedButton(

                onPressed: () {

                  // Retourne true si le formulaire est valide, sinon false

                  if (\_formKey.currentState!.validate()) {

                    print(\_usernameController.text);

                    print(\_passwordController.text);

                    try {

                      // Appelle la méthode du controller GetX avec les valeurs des champs

                      \_controller.register(

                        \_usernameController.text,

                        \_passwordController.text,

                      );

                      // Si l'ajout est réussi, afficher message

                      ScaffoldMessenger.of(context).showSnackBar(

                        const SnackBar(

                            content: Text('User added successfully')),

                      );

                      // Redirection vers la page home

                      Navigator.of(context).pushNamed('/');

                    } catch (error) {

                      // Si une erreur survient, afficher un message d'erreur

                      ScaffoldMessenger.of(context).showSnackBar(

                        SnackBar(content: Text('Error adding user: $error')),

                      );

                    }

                  }

                },

                style: ButtonStyle(

                  backgroundColor: WidgetStateProperty.all<Color>(

                    Colors.purpleAccent,

                  ),

                ),

                child: const Text("submit"),

              ),

            ),

          ],

        ),

      ),

    );

  }

}

# Le screen Register

Créer /lib/presentation/screens/register.screen.dart



import 'package:flutter/material.dart';

import 'package:myflutterapplication/presentation/widgets/registerform.widget.dart';

class Register extends StatelessWidget {

  const Register({super.key});

  @override

    Widget build(BuildContext context) {

      return Scaffold(

        appBar: AppBar(

          elevation: 15,

          backgroundColor: Colors.purpleAccent,

          title: const Text("Register"),

          leading: IconButton(

            onPressed: () {},

            icon: const Icon(Icons.app\_registration\_outlined),

          ),

            ),

        body: const Registerform(),

    );

  }

}

# Créer la route

Dans /lib/approuter.dart



Ajouter la route :

'/Settings': (context) => const Register(), // Route pour l'écran register

# Injection des dépendances

Ajouter dans /lib/main.dart l’injection de dépendance correspondante :



import 'package:flutter/material.dart';

import 'package:myflutterapplication/approuter.dart';

import 'package:get/instance\_manager.dart';

import 'package:myflutterapplication/data/datasource/localdatasource/user\_local\_data\_source.dart';

import 'package:myflutterapplication/data/datasource/services/categorie.service.dart';

import 'package:myflutterapplication/data/repositories/categorie.repository.dart';

import 'package:myflutterapplication/data/repositories/user.repository.dart';

import 'package:myflutterapplication/domain/usecases/categorie.usecase.dart';

import 'package:myflutterapplication/domain/usecases/user.usecase.dart';

import 'package:myflutterapplication/presentation/controllers/categorie.controller.dart';

import 'package:myflutterapplication/presentation/controllers/user.controller.dart';

// Fonction principale qui lance l'application Flutter

void main() {

  Get.put(CategorieService());

  Get.put(CategorieRepository(catserv: Get.find()));

  Get.put(CategorieUseCase(respository: Get.find()));

  Get.put(CategorieController(useCase: Get.find()));

// injection authentification

  Get.put(UserLocalDataSource());

  Get.put(UserRepository(localDataSource: Get.find()));

  Get.put(AuthenticateUserUseCase(repository: Get.find()));

  Get.put(AuthController(userUseCase: Get.find()));

  // Lance l'application en exécutant MyApp

  runApp(const MyApp());

}

// Définition de la classe stateless MyApp

class MyApp extends StatelessWidget {

  // Constructeur constant avec une clé facultative

  const MyApp({super.key});

  // Ce widget est la racine de l'application

  @override

  Widget build(BuildContext context) {

    // Retourne un MaterialApp configuré

    return MaterialApp(

       // Titre de l'application

      title: 'Flutter Application',

      // Thème de l'application avec une palette de couleurs personnalisée

      theme: ThemeData(

        // Utilisation d'un ColorScheme basé sur une couleur de départ

        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),

        // Activation de Material Design 3

        useMaterial3: true,

        ),

      // Désactivation du bandeau "DEBUG"

      debugShowCheckedModeBanner: false,

       initialRoute: '/',

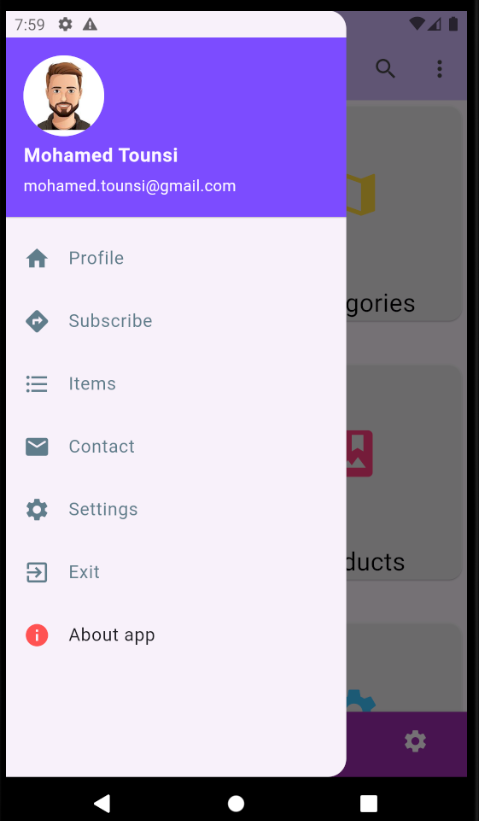
       routes: appRoutes(),  // Utilisation des routes depuis le fichier séparé

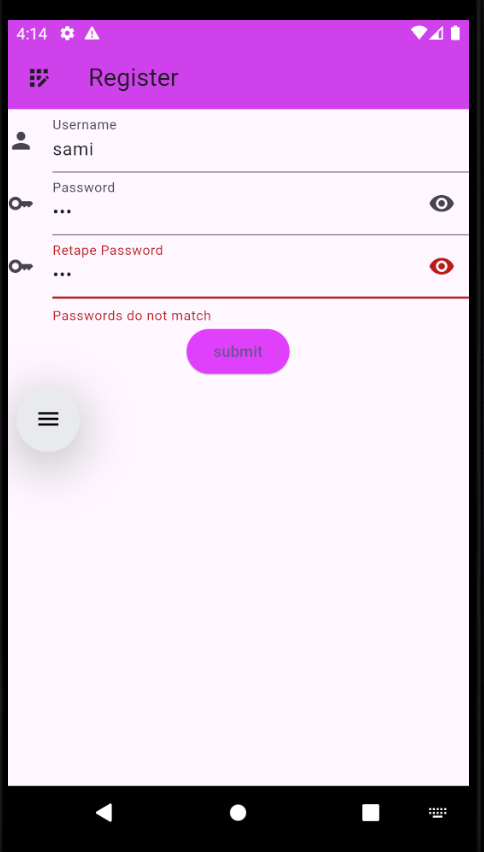
    );

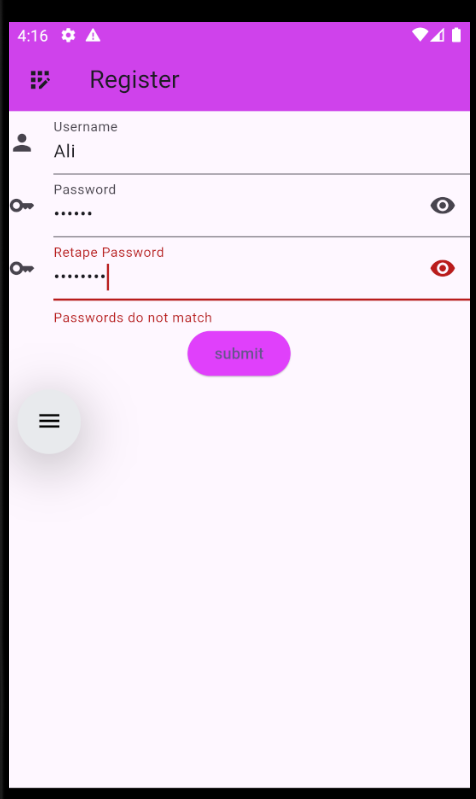
  }

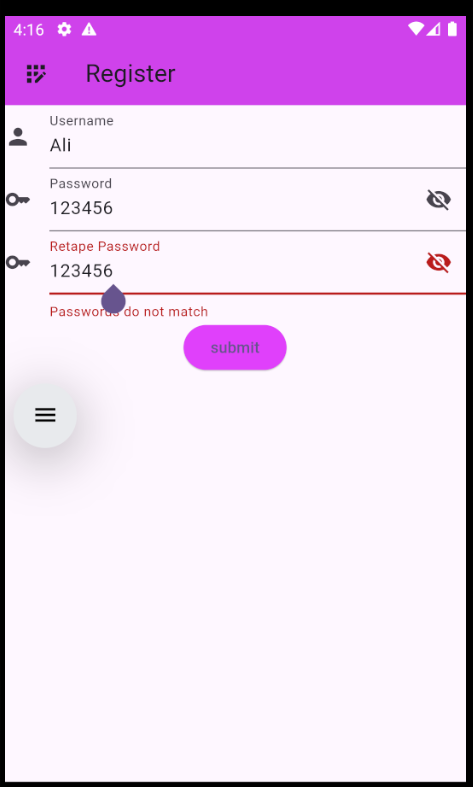
}

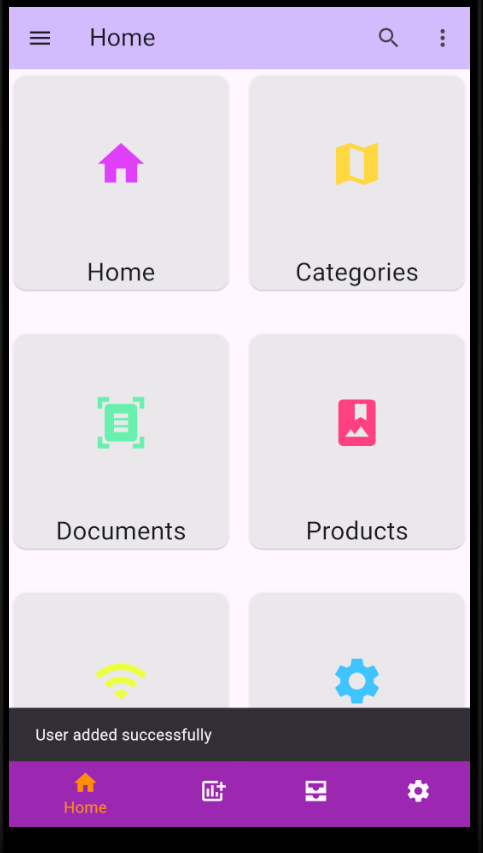
**Résultat**











# Widget form login

Créer le fichier /lib/presentation/widgets/loginform.widget.dart



import 'package:flutter/material.dart';

import 'package:get/get.dart';

import 'package:myflutterapplication/presentation/controllers/user.controller.dart';

class Loginform extends StatefulWidget {

  const Loginform({super.key});

  @override

  State<Loginform> createState() => \_Loginform();

}

class \_Loginform extends State<Loginform> {

  // Initialisation du contrôleur

  final AuthController \_controller = Get.put(AuthController(

    userUseCase: Get.find(),

  ));

  final \_formKey = GlobalKey<FormState>();

  late TextEditingController \_usernameController;

  late TextEditingController \_passwordController;

  // show the password or not

  bool \_isObscure = true;

  @override

  void initState() {

    super.initState();

    \_usernameController = TextEditingController();

    \_passwordController = TextEditingController();

  }

  @override

  Widget build(BuildContext context) {

    return SingleChildScrollView(

      child: Form(

        key: \_formKey,

        child: Column(

          crossAxisAlignment: CrossAxisAlignment.start,

          children: <Widget>[

            TextFormField(

              controller: \_usernameController,

              decoration: const InputDecoration(

                  hintText: "Username", labelText: "Username"),

            ),

            TextFormField(

              obscureText: \_isObscure,

              controller: \_passwordController,

              decoration: InputDecoration(

                hintText: "Password",

                labelText: "Password",

                // this button is used to toggle the password visibility

                suffixIcon: IconButton(

                    icon: Icon(

                        \_isObscure ? Icons.visibility : Icons.visibility\_off),

                    onPressed: () {

                      setState(() {

                        \_isObscure = !\_isObscure;

                      });

                    }),

              ),

            ),

            Center(

              child: ElevatedButton(

                onPressed: () {

                  if (\_formKey.currentState!.validate()) {

                    \_controller

                        .login(

                      \_usernameController.text,

                      \_passwordController.text,

                    )

                        .then((success) {

                      if (success) {

                        ScaffoldMessenger.of(context).showSnackBar(

                          const SnackBar(content: Text('User accepted')),

                        );

                        Navigator.of(context).pushNamed('/');

                      } else {

                        ScaffoldMessenger.of(context).showSnackBar(

                          const SnackBar(content: Text('Error: wrong user')),

                        );

                      }

                    }).catchError((error) {

                      // Gestion d'une éventuelle erreur inattendue

                      ScaffoldMessenger.of(context).showSnackBar(

                        SnackBar(content: Text('Error: $error')),

                      );

                    });

                  }

                },

                style: ButtonStyle(

                  backgroundColor: WidgetStateProperty.all<Color>(

                    Colors.purpleAccent,

                  ),

                ),

                child: const Text("submit"),

              ),

            ),

            Row(

              mainAxisAlignment: MainAxisAlignment.center,

              children: [

                const Text("Dont have an account? "),

                TextButton(

                  onPressed: () {

                    Navigator.pushNamed(context, '/register');

                  },

                  child: const Text(

                    "Sign Up",

                    style: TextStyle(color: Colors.purple),

                  ),

                ),

              ],

            ),

          ],

        ),

      ),

    );

  }

}

# Le screen Login

Créer le fichier /lib/presentation/screens/login.screen.dart



import 'package:flutter/material.dart';

import 'package:myflutterapplication/presentation/widgets/loginform.widget.dart';

class Login extends StatelessWidget {

  const Login({super.key});

  @override

    Widget build(BuildContext context) {

      return Scaffold(

        appBar: AppBar(

          elevation: 15,

          backgroundColor: Colors.purpleAccent,

          title: const Text("Login"),

          leading: IconButton(

            onPressed: () {},

            icon: const Icon(Icons.person\_2\_outlined),

          ),

            ),

        body: const Loginform(),

    );

  }

}

# Les routes

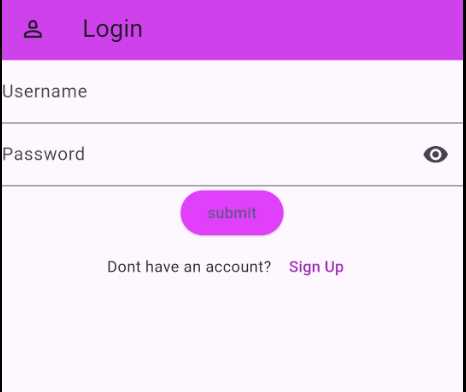
Ajouter les routes :

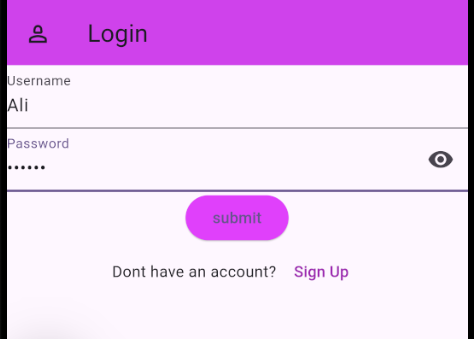


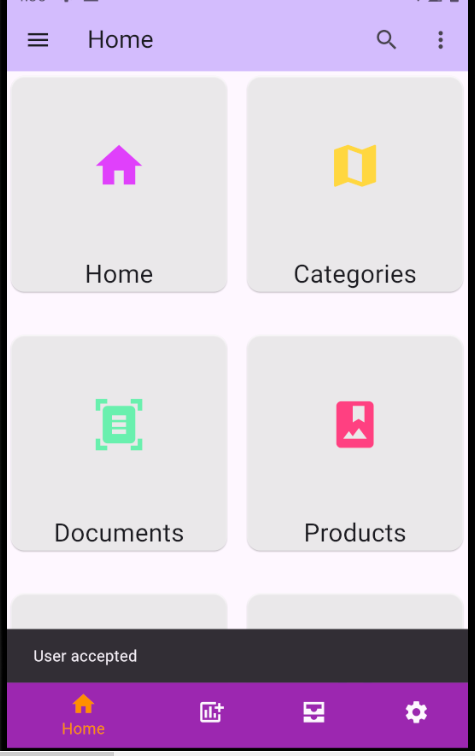
'/Settings': (context) => const Login(), // Route pour l'écran login

'/register': (context) => const Register(), // Route pour l'écran register

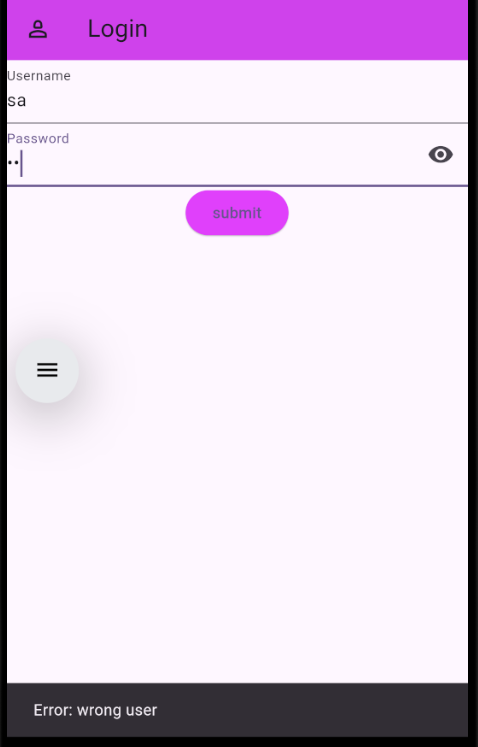
**Résultat**







En cas de saisie eronnée



# Le Widget Settings

Créer le fichier /lib/presentation/widgets/settings.widget.dart



import 'package:flutter/material.dart';

class SettingsWidget extends StatefulWidget {

  const SettingsWidget({super.key});

  @override

  SettingsWidgetState createState() => SettingsWidgetState();

}

class SettingsWidgetState extends State<SettingsWidget> {

  // Variables pour stocker l'état des Switch

  bool notificationsEnabled = false;

  bool locationEnabled = false;

  @override

  Widget build(BuildContext context) {

    return ListView(

      children: [

        ListTile(

          leading: const Icon(Icons.notifications),

          title: const Text('Notifications'),

          trailing: Switch(

            value: notificationsEnabled,

            onChanged: (value) {

              setState(() {

                notificationsEnabled = value; // Mise à jour de l'état

              });

              print('Notifications activées: $notificationsEnabled');

            },

          ),

        ),

        ListTile(

          leading: const Icon(Icons.location\_on),

          title: const Text('Location services'),

          trailing: Switch(

            value: locationEnabled,

            onChanged: (value) {

              setState(() {

                locationEnabled = value; // Mise à jour de l'état

              });

              print('Services de localisation activés: $locationEnabled');

            },

          ),

        ),

        ListTile(

          leading: const Icon(Icons.info),

          title: const Text('About'),

          onTap: () {},

        ),

      ],

    );

  }

}

**Explication**

SettingsWidget utilise un StatefulWidget pour afficher une liste de paramètres interactifs, les notifications et les services de localisation, avec la possibilité de les activer ou les désactiver via des interrupteurs (Switch).

notificationsEnabled : Variable booléenne qui stocke l'état des notifications (activé ou désactivé).

locationEnabled : Variable booléenne qui stocke l'état des services de localisation (activé ou désactivé).

Ces variables sont modifiées lorsque l'utilisateur interagit avec les switchs.

Switch : Un interrupteur (Switch) est utilisé comme trailing (élément à droite) pour activer ou désactiver les notifications.

value : Lié à la variable d'état notificationsEnabled / locationEnabled.

onChanged : Lorsque l'utilisateur change l'état du Switch, la méthode setState() est appelée, ce qui met à jour la valeur de notificationsEnabled / locationEnabled et réaffiche le widget avec le nouvel état.

On utilise la méthode setState() pour mettre à jour l'interface lorsque l'utilisateur interagit avec les Switch.

# Screen Settings

Créer le fichier /lib/presentation/screens/settings.screen.dart



import 'package:flutter/material.dart';

import 'package:myflutterapplication/presentation/widgets/settings.widget.dart';

class SettingsScreen extends StatelessWidget {

  const SettingsScreen({super.key});

  @override

  Widget build(BuildContext context) {

     return Scaffold(

        appBar: AppBar(

          elevation: 15,

          backgroundColor: const Color.fromARGB(255, 225, 122, 188),

          title: const Text("Settings and Help"),

          leading: IconButton(

            onPressed: () {},

            icon: const Icon(Icons.settings),

          ),

            ),

        body: const SettingsWidget(),

    );

  }

}

# Le lien vers Settings

Dans le fichier /lib/presentation/widgets/loginform.widget.dart, modifier le lien vers le screen de settings.



…..

      .then((success) {

                      if (success) {

                        ScaffoldMessenger.of(context).showSnackBar(

                          const SnackBar(content: Text('User accepted')),

                        );

                        Navigator.of(context).pushNamed('/**settingsDetails**');

                      } else {

                        ScaffoldMessenger.of(context).showSnackBar(

                          const SnackBar(content: Text('Error: wrong user')),

                        );

                      }

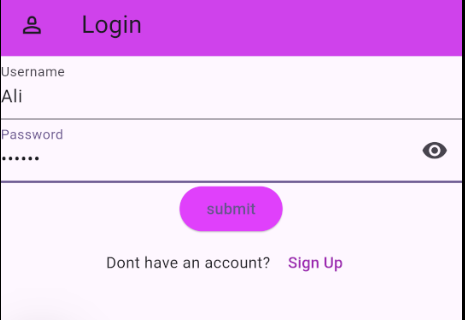
…..

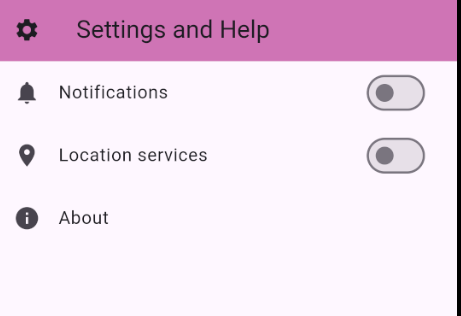
Il ne faut pas oublier de déclarer la route dans /lib/approuter.dart



 '/settingsDetails': (context) => const SettingsScreen(), // Route pour l'écran SettingsScreen

**Résultat**





# Map screen

Créer le fichier /lib/presentation/screens/map.screen.dart



import 'package:flutter/material.dart';

import 'package:flutter\_map/flutter\_map.dart';

import 'package:latlong2/latlong.dart';

class MapScreen extends StatefulWidget {

  const MapScreen({super.key});

  @override

  MapScreenState createState() => MapScreenState();

}

class MapScreenState extends State<MapScreen> {

  final MapController \_mapController = MapController();

  @override

  void initState() {

    super.initState();

  }

  LatLng latLng = const LatLng(34.757245951399064, 10.772160363349858);

  @override

  Widget build(BuildContext context) {

    return FlutterMap(

      mapController: \_mapController,

      options: MapOptions(

        initialCenter: latLng,

        initialZoom: 13.0,

      ),

      children: [

        TileLayer(

          urlTemplate: "https://tile.openstreetmap.org/{z}/{x}/{y}.png",

        ),

        MarkerLayer(

          markers: [

            Marker(

              point: latLng,

              width: 60,

              height: 60,

              alignment: Alignment.topCenter,

              child: Icon(

                Icons.location\_pin,

                color: Colors.red.shade700,

                size: 60,

              ),

            ),

          ],

        ),

      ],

    );

  }

}

**Explication**

L'URL https://tile.openstreetmap.org/{z}/{x}/{y}.png est un modèle utilisé pour accéder aux tuiles (ou "tiles") cartographiques fournies par OpenStreetMap. Ces tuiles sont des images carrées qui, ensemble, forment la carte visible dans les applications. Le modèle d'URL inclut trois variables : {z}, {x}, et {y}, qui déterminent quelle tuile spécifique est chargée à un certain niveau de zoom et à une certaine position sur la carte.

Voici ce que signifient ces variables :

1. **{z} (Zoom Level)** : Cette variable représente le niveau de zoom. Un niveau de zoom de 0 représente le monde entier dans une seule tuile, et chaque niveau de zoom supplémentaire multiplie le nombre de tuiles par 4. Par exemple :
   * Niveau 0 (z=0) : 1 tuile (pour la planète entière).
   * Niveau 1 (z=1) : 4 tuiles.
   * Niveau 2 (z=2) : 16 tuiles.
   * Plus le niveau de zoom est élevé, plus la carte est détaillée.
2. **{x} (Tile Column)** : Cela représente la colonne de la tuile sur une grille. À chaque niveau de zoom, le monde est divisé en plusieurs colonnes, et cette variable spécifie la colonne où se trouve la tuile.
3. **{y} (Tile Row)** : Cela représente la ligne de la tuile dans la grille. Comme pour les colonnes, chaque niveau de zoom divise le monde en plusieurs lignes, et cette variable détermine quelle ligne spécifique de tuiles est chargée.

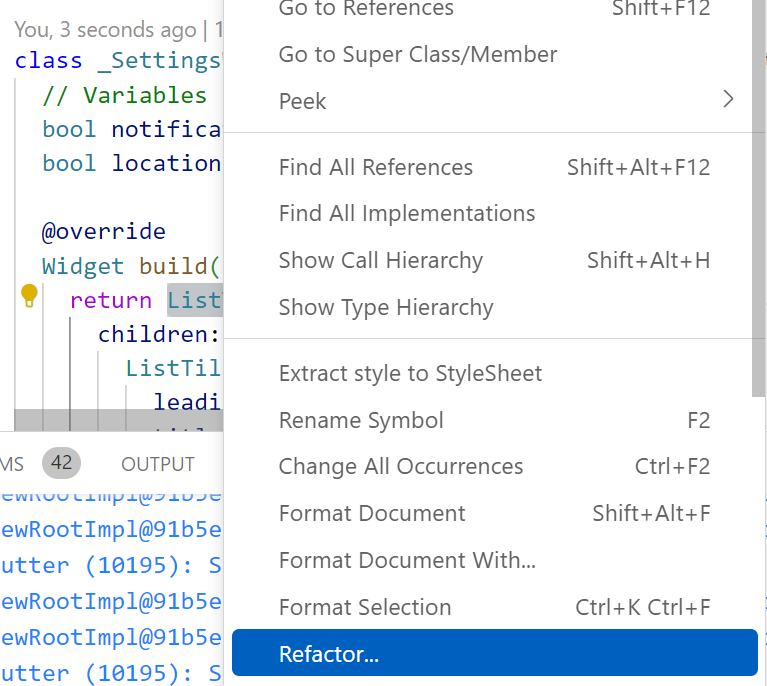
Chaque combinaison unique de {z}, {x}, et {y} représente une tuile spécifique sur la carte. L’application charge dynamiquement plusieurs tuiles pour former la carte complète en fonction de la zone visible et du niveau de zoom choisi.

Les valeurs pour {z}, {x}, et {y} sont calculées automatiquement en fonction de la position actuelle de la carte (latitude/longitude) et du niveau de zoom.

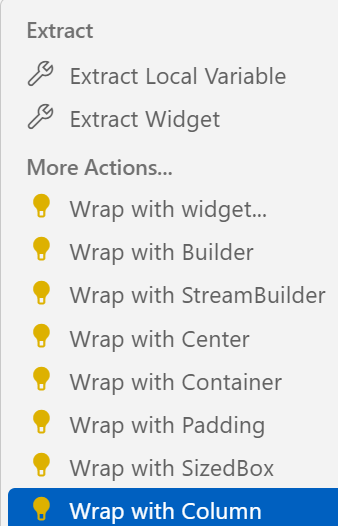
Par la suite, apporter les modifications suivantes dans settings.widget.dart, pour lorsqu’on active locationEnabled, le MapScreen sera affiché.



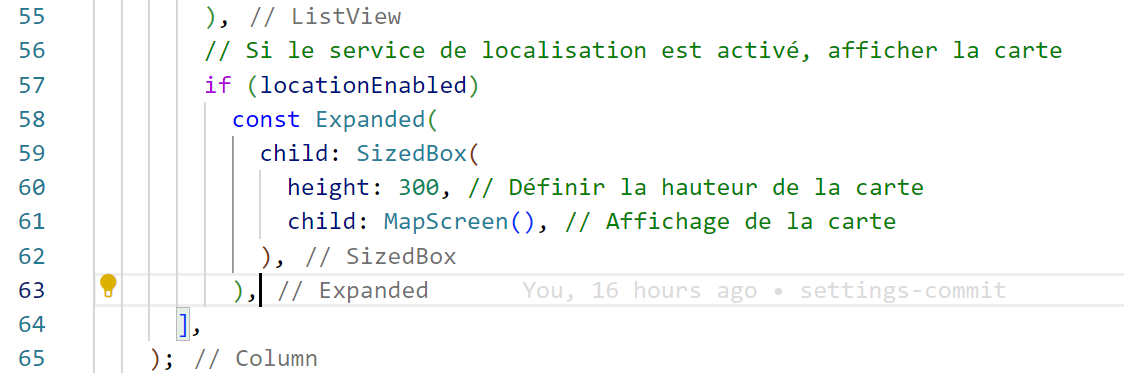
Sélectionner ListView puis bouton droit et choisir Refactor.



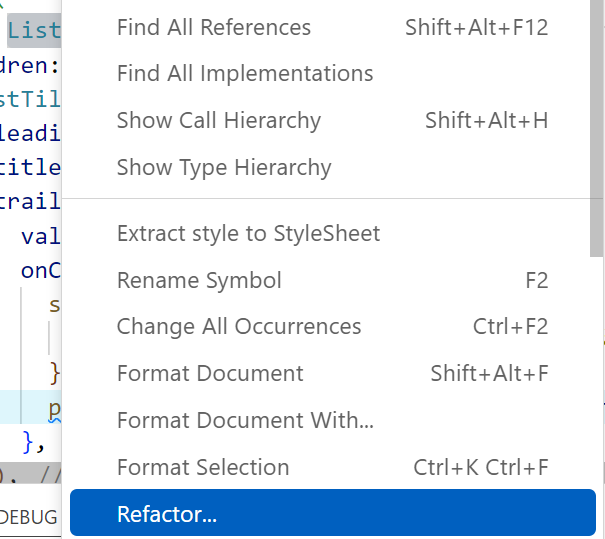
Puis choisir Wrap with Column.



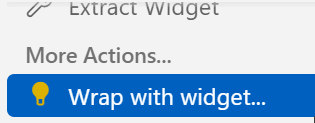
Puis à la suite de ListView on ajoutera ce code :



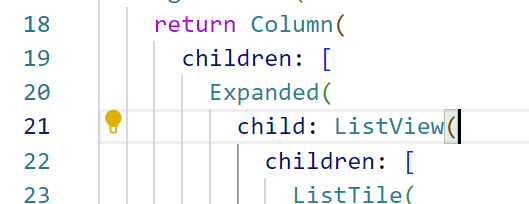
Puis de nouveau sélectionner ListView puis bouton droit et choisir refactor



Puis choisir



Puis taper Expanded.



Le code complet deviendra :

import 'package:flutter/material.dart';

import 'package:myflutterapplication/presentation/screens/map.screen.dart';

class SettingsWidget extends StatefulWidget {

  const SettingsWidget({super.key});

  @override

  SettingsWidgetState createState() => SettingsWidgetState();

}

class SettingsWidgetState extends State<SettingsWidget> {

  // Variables pour stocker l'état des Switch

  bool notificationsEnabled = false;

  bool locationEnabled = false;

  @override

  Widget build(BuildContext context) {

    return Column(

      children: [

        Expanded(

          child: ListView(

            children: [

              ListTile(

                leading: const Icon(Icons.notifications),

                title: const Text('Notifications'),

                trailing: Switch(

                  value: notificationsEnabled,

                  onChanged: (value) {

                    setState(() {

                      notificationsEnabled = value; // Mise à jour de l'état

                    });

                    print('Notifications activées: $notificationsEnabled');

                  },

                ),

              ),

              ListTile(

                leading: const Icon(Icons.location\_on),

                title: const Text('Location services'),

                trailing: Switch(

                  value: locationEnabled,

                  onChanged: (value) {

                    setState(() {

                      locationEnabled = value; // Mise à jour de l'état

                    });

                    print('Services de localisation activés: $locationEnabled');

                  },

                ),

              ),

              ListTile(

                leading: const Icon(Icons.info),

                title: const Text('About'),

                onTap: () {

                  // Action à définir pour la section About

                },

              ),

            ],

          ),

        ),

        // Si le service de localisation est activé, afficher la carte

        if (locationEnabled)

          const Expanded(

            child: SizedBox(

              height: 300,

              child: MapScreen(), // Affichage de la carte

            ),

          ),

      ],

    );

  }

}

**Résultat**

