Flutter TP 2 – SQFLite

Flutter met à disposition une dépendance qui permet de manipuler les bases de données de type SQLite : **SQfLite**

L'objectif de cette séance de travaux pratiques est de découvrir et manipuler les bases de données SQLite avec Flutter

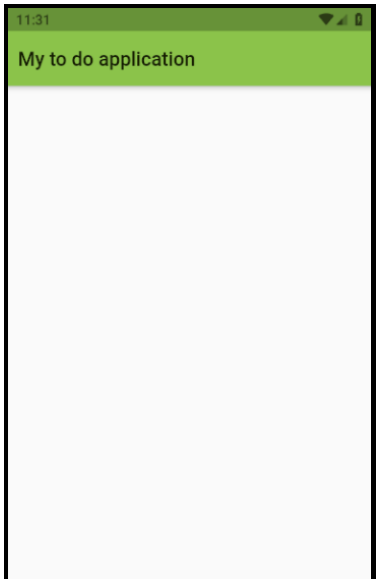
Exercice : Application *To do*

a. Création de l'interface de base de l'application

- Créez un nouveau projet Flutter avec nom de projet **todo**
- Nous allons garder le squelette du projet seulement, pour cela remplacer le contenu de votre **lib/main.dart** par le code suivant :

Remarque :

- Pour la version dart ultérieur à 2.9 , rajouter la ligne \\@dart2.9 au début De votre fichier dart afin d'éviter les erreurs de null safety



Source : link

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo BD SQLite',
      theme: ThemeData(
        primarySwatch: Colors.lightGreen,
      ),
      home: MyHomePage(title: 'My to do application'),
    );
  }
}

  class MyHomePage extends StatefulWidget {
    MyHomePage({Key key, this.title}) : super(key: key);
    final String title;
    @override
    _MyHomePageState createState() => _MyHomePageState();
  }
  class _MyHomePageState extends State<MyHomePage> {
    @override
    Widget build(BuildContext context) {
      return Scaffold(
        appBar: AppBar(
          // Here we take the value from the MyHomePage object that was created by
          // the App.build method, and use it to set our appbar title.
          title: Text(widget.title),
        ),
        body: Center(),
  // This trailing comma makes auto-formatting nicer for build methods.
      );
    }
  }
```

Flutter TP 2 – SQFLite

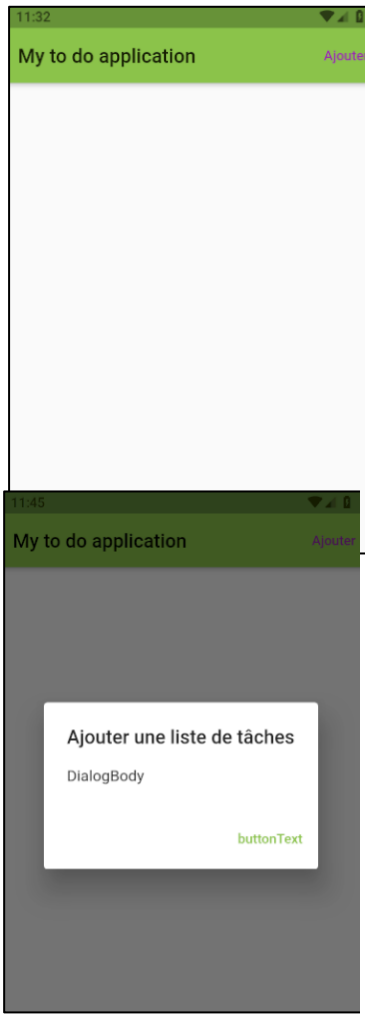
- Pour ajouter des éléments à la liste **todo** qui seront stockés dans la base de données, nous ajoutons un bouton à la **AppBar**. Pour cela ajoutez à **l'AppBar** un composant action qui contiendra un TextButton comme ceci :

```
actions: [TextButton(onPressed: null, child: const Text("Ajouter", style:
TextStyle (color:Colors. purple)))],
```

- Nous ajoutons ensuite à la méthode **onPressed**() la fonction qui nous permettra d'ajouter des éléments à liste todo.
- Pour cela nous utilisons un composant **Alert dialogue**. Ajoutez la fonction suivante dans la classe **_MyHomePageState.**

```
    body: Center(),
// This trailing comma makes auto-formatting nicer for build methods.
    ); // Scaffold
  }

  Future<void> ajouter() async {
  await showDialog<void>(
```

```
Future<void> ajouter() async {
await showDialog<void>(
    context: context,
    barrierDismissible: false, // si on appuie à l'extérieur
                               //ça ne va pas disparaitre
    builder: (BuildContext dialogContext) {
      return AlertDialog(
        title: const Text('Ajouter une liste de tâches'),
        content: const Text('DialogBody'),

        actions: <Widget>[
          TextButton(
            child: const Text('buttonText'),
            onPressed: () {
              Navigator.of(dialogContext).pop();
            },
          ),
        ],
      );
    },
  );
}
```



La fonction **Ajouter** est de type asynchrone, c'est pour cela qu'elle est de type **Future**. Il faudra aussi importer : `import 'dart:async';`

- Il faut aussi remplacer **null** par **ajouter**, dans **onPressed**

Flutter TP 2 – SQFLite

- Nous allons maintenant personnaliser cette boite de dialogue pour lui ajouter deux boutons **valider** et **annuler**
- Ajoutez également une variable string **nouvelletache** à classe **_MyHomePageState**

```
String nouvelletache;
```

- Et remplacez le code de la fonction ajouter par le code suivant :

```
Future<void> ajouter() async {
  await showDialog<void>(
    context: context,
    barrierDismissible: false,
    // false = user must tap button, true = tap outside dialog
    builder: (BuildContext dialogContext) {
      return AlertDialog(
        title: const Text('Ajouter une liste de taches'),
        //  content: Text('DialBody'),
        content: TextField(
          decoration: const InputDecoration(
            labelText: "Element",
            hintText: "exemple : ma prochaine tache",
          ),
          onChanged: (str) {
            nouvelleTache = str;
          },
        ),
        actions: <Widget>[
          TextButton(
            child: const Text('Annuler', style: TextStyle(color: Colors.red)),
            onPressed: () {
              //  Navigator.of(dialogContext).pop(); // Dismiss alert dialog
              Navigator.pop(dialogContext);
            },
          ),
          TextButton(
            child:
                const Text('Valider', style: TextStyle(color: Colors.blue)),
            onPressed: () {
              // Ajouter le code pour l'insertion dans la base de données
              Navigator.of(dialogContext).pop(); // Dismiss alert dialog
            },
          ),
        ],
      );
    },
  );
}
```

b. Préparation de la création de la base de données :

- Nous allons créer la classe qui gère la liste de taches associées à la base de données.
- Allez dans le dossier **lib,** créez un nouveau package en lui donnant le nom **model,** et dans ce nouveau package créer un fichier .**dart** avec le nom : **item.dart**

Flutter TP 2 – SQFLite

Ajoutez la classe **Item** :

```dart
class Item {
  int? id;
  String? nom;

  Item(Map<String, dynamic> map) {
    nom = map['nom'] ?? "";
  }

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'nom': nom,
    };
  }
}
```

c. Configuration de **SQFlite** (plugin Flutter de SQLite : https://pub.dev/packages/sqflite)

- Pour installer **SQflite** ajoutez la dépendance : sqflite: ^2.0.2+1 au fichier **pubspec.yaml**

```
dependencies:
  flutter:
    sdk: flutter
  sqflite: ^2.0.2+1
```

ou alors ouvrez le terminal de votre editeur et entrez la commande : flutter pub add sqflite

- Dans le même fichier, ajoutez également la dépendance vers le **path provider** : `path_provider: ^2.0.10` ou `faire (flutter pub add path_provider)`, qui permet d'obtenir les chemins d'accès vers les fichiers au sein de l'application (notamment les dossiers de l'application) comme indiqué ci-dessous :

```
dependencies:
  flutter:
    sdk: flutter
  sqflite: ^2.0.2+1
  path_provider: ^2.0.10
```

- Appliquer le changement en appuyant sur **Pub get** sur Android Studio (ou faire la commande flutter pub get ou dart pub get)

d. Création de la base de données :

- Créez un nouveau fichier **databaseClient.dart** dans le dossier **model**
- Nous commençons par créer la classe **DatabaseClient.** Complétez la classe par le getter et la méthode **create**() qui va créer la base de données :

```dart
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart' as p;
import 'dart:io'; // pour l'utilisation de la classe Directory
import 'package:todo/models/item.dart';

class DatabaseClient {
  Database? _database;

  Future<Database?> get database async {
    return _database ?? await create();
  }

  Future create() async {
    Directory directory = await getApplicationDocumentsDirectory();
    String database_directory = p.join(directory.path, 'database.db');

    var bdd =
        await openDatabase(database_directory, version: 1, onCreate: _onCreate);
    return bdd;
  }

  Future _onCreate(Database db, int version) async {
    await db.execute(
        "CREATE TABLE IF NOT EXISTS item (id INTEGER PRIMARY KEY, nom TEXT)");
    print("database created!");
    await db.insert("item", {'nom': 'tache1'});
  }
}
```

e. Lecture des données depuis la base de données :

- Nous allons maintenant ajouter dans la classe **DatabaseClient** les fonctions permettant d'écrire et de lire depuis la base de données
- Ajoutez l'import de item.dart pour utiliser la classe **item** :

```dart
import 'package:todo/model/item.dart';
```

Remarques:
A noter que le chemin vers votre fichier item.dart peut changer selon le nom de votre package, en règle générale veillez mettre : `import 'package:package_name/model/item.dart';`

Écriture des données et lecture des données : Ajoutez les fonctions qui permettent l'écriture des données :

Flutter TP 2 – SQFLite

```dart
// Ecriture des données
  Future<Item> ajouItem(Item item) async {
    Database? maDatabase = await database; // fait appel au get database
    item.id = await maDatabase?.insert('item',
        item.toMap()); // la fontion insert renvoie un id qui va initialiser l'id de l'item
en cours
    return item;
  }

  // Lecture des données
  Future<List<Item>> allItem() async {
    Database? maDatabase = await database;
    List<Map<String, dynamic>>? resultat = await maDatabase?.rawQuery(
        'select * FROM item'); // Une liste de maps, chaque enregistrement est une //map
composée de clés valeurs nom, nom et id id
    List<Item> items = [];
    resultat?.forEach((map) {
      Item item = Item(map);
      items.add(item);
    });
    return items;
  }
```

Affichage des éléments :

- Nous allons maintenant afficher les éléments de la base de données
- Avant cela nous allons créer un composant qui sera affiché quand il n y'a aucune tâche à afficher
- Créez un package widgets et dans ce package créez le fichier **donnees_vide.dart**
- Ajoutez la classe **DonneesVides** suivante :

```dart
import 'package:flutter/material.dart';

class DonneesVides extends StatelessWidget {
  const DonneesVides({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text(
        "Aucune donnée n'est présente",
        textScaleFactor: 2.5,
        textAlign: TextAlign.center,
        style: TextStyle(color: Colors.red, fontStyle: FontStyle.italic),
      ),
    );
  }
}
```

Flutter TP 2 – SQFLite

- Revenons au fichier **main.dart** (fichier qui contient le design principal)

- Ajoutez une liste d'items dans la classe class _MyHomePageState la déclaration de liste des taches :

```
List<Item> items;
```

- Ajoutez l'import de ***item.dart***

```
import 'package:todo/model/item.dart';
```
et aussi :
```
import 'dart:async';
```

- Si la liste est vide c'est le widget **DonnesVides** qui va être affiché
- Dans la classe **_MyHomePageState**, nous allons changer le contenu de la section **body** pour y placer la liste des taches. Mais si la liste est vide le widget **DonnesVides** sera affiché

- Ajoutez l'import de **DonneesVides** :
```
import 'package:todo/widgets/donnees_vides.dart';
```

- Remplacez la section body: Center() par :

- body: (items == null || items.length == 0)
  ? DonnesVides() : null,

- Quand la liste n'est pas vide, il faudra afficher les éléments de la liste

- Pour cela nous allons utiliser le composant **ListView** Builder (**ListView.builder** creates a scrollable, linear array of widgets.) qui affichera l'ensemble des taches de la base de données. Chaque tâche sera affichée par le composant Flutter **ListTile** .

- Remplacez le code du **body** précédent par le suivant

```
body: (items == null || items.length == 0)
    ? DonnesVides()
    : ListView.builder
        (
      itemCount: items.length,
      itemBuilder: (Context, i) {
        Item item = items[i];
        return ListTile
        (
          title: const Text(item.nom),
        );
      }),
```

Flutter TP 2 – SQFLite

- Pour le moment rien ne va changer au niveau de l'application car nous n'avons pas encore chargé les données en mémoire

- Pour cela, ajoutons la fonction ***recuperer()*** qui chargera les données depuis la base de données en mémoire, toujours dans classe class _MyHomePageState, après la fonction ***ajouter()***

```
void recuperer()
{
        DataBaseClient().allItem().then((items)
        {
                setState(()
                {
                this.items = items;
                });
        });
}
```

- Ajoutez l'import de DataBaseClient.dart pour accéder aux fonctionnalités de la base de données :

```
import 'package:todo/model/DatabaseClient.dart';
```

- Nous allons appeler la fonction `recuperer()` dès le chargement du composant au sein de la fonction initState(). Ajoutez la fonction iniState() juste après les déclarations de nouvelletache et list<Item> :

```
@override
void initState() {
  // TODO: implement initState
  super.initState();
  recuperer();
}
```

- Nous allons maintenant compléter la partie du code qui permettra de rajouter un élément à la base de données quand le bouton **valider** est pressé. Ce bouton est géré dans la fonction ajouter et pour le moment, il fait simplement disparaitre la boite de dialogue.
- Nous avons actuellement le code suivant pour la fonction **onPressed** du bouton valider :

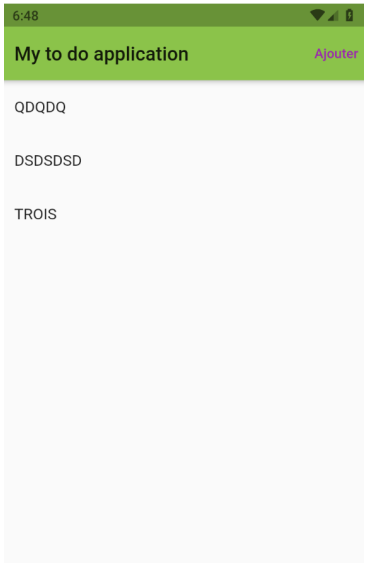
```
onPressed: () {
  //  Navigator.of(dialogContext).pop(); // Dismiss alert dialog
  Navigator.pop(dialogContext);
},
```

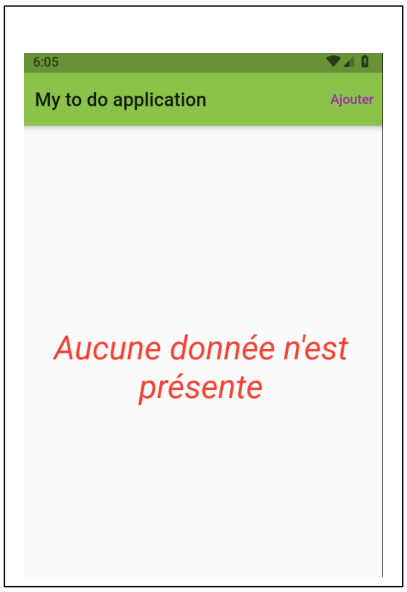
- Remplacez le code précédent de la fonction **onPresssed** par celui-ci :

```
onPressed: ()
{
  // Ajouter le code pour l'insertion dans la base de données
  if (nouvelletache != null)
{
    Map<String, dynamic> map = {'nom: nouvelletache};
    Item item = new Item();
    item.fromMap(map);
    DataBaseClient()
        .ajouItem(item)
        .then((i) => recuperer()); // add to bd (ajouter item)
and set to state (recuperer)
```

```
            nouvelletache = null;
          }

          Navigator.of(dialogContext).pop(); // Dismiss alert dialog
        },
```

- Maintenant il est possible de voir les éléments de la liste affichés puisque la fonction ajoutItem(item) est appelée depuis le bouton valider :





f. Surpression des éléments :

- Nous allons maintenant supprimer les éléments
- Nous allons ajouter dans la classe **DatabaseClient** la fonction permettant de supprimer un élément depuis la base de données
- Ajoutez la fonction suivante à la classe **DatabaseClient**

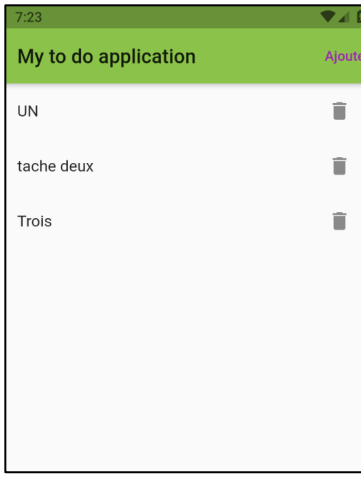
```
// Suppression des données
```

```
Future<int?> delete(int id, String table) async {
    Database? maDatabase = await database;
    return await maDatabase?.delete(table, where: 'id = ?', whereArgs: [id]);
  }
```

- Dans le fichier **main.dart** le composant ListTile affiche chaque élément de liste, nous ajouterons la propriété 'bouton supprimer' à ce composant pour lancer la suppression d'une tâche.

- Ajoutez au composant ListTile la propriété suivante :

```
trailing: IconButton(
    icon: const Icon(Icons.delete),
    onPressed: ()
    {
      DataBaseClient().delete(item.id, 'item').then((i)
       {
        print("Nombre d'éléments supprimés est : $i");
        recuperer();
      });
    }),
```



g. Mise à jour des éléments :

- Nous allons maintenant modifier les éléments

- Nous allons ajouter dans la classe **DatabaseClient** la fonction **update** permettant de modifier un élément de la base de données

```
// update des données
```

```
Future<int?> updateItem(Item item) async {
    Database? maDatabase = await database;
    return await maDatabase
        ?.update('item', item.toMap(), where: 'id = ?', whereArgs: [item.id]);
  }
```

- Nous ajoutons aussi dans la classe **DatabaseClient** la fonction **update_or_insert** qui fait soit la mise à jour ou l'insertion :

Flutter TP 2 – SQFLite

```
// update or insert
```

```
Future<int?> update_or_insert(Item item) async {
    Database? maDatabase = await database;
    if (item.id == null) {
      item.id = await maDatabase?.insert('item', item.toMap());
    } else {
      return await maDatabase
          ?.update('item', item.toMap(), where: 'id = ?', whereArgs: [item.id]);
    }
}
```

- Pour appeler cette fonction update dans le fichier **main.dart,** nous ajouterons la propriété 'bouton update' au ListTile. Ajoutez au composant ListTile la propriété suivante :

```
-   leading: IconButton(
-                       icon: const Icon(Icons.edit),
-                       onPressed: (() => DatabaseClient().ajouItem(item))),
```

- Nous devons aussi changer la fonction ajouter. Nous devons lui ajouter un paramètre item. Si (item == null) nous appliquons une insertion sinon nous appliquons une mise à jour (les changements à effectuer sont en jaune sur le code en dessous) :

```
Future<void> ajouter(Item? item) async {
    final elem = item;
    await showDialog<void>(
      context: context,
      barrierDismissible: false,
      // false = user must tap button, true = tap outside dialog
      builder: (BuildContext dialogContext) {
        return AlertDialog(
          title: const Text('Ajouter une liste de taches'),
          //  content: Text('DialBody'),
          content: TextField(
            decoration: InputDecoration(
              labelText: "Element",
              hintText: item?.nom ?? "exemple : ma prochaine tache",
            ),
            onChanged: (str) {
              setState(() {
                nouvelleTache = str;
              });
            },
          ),
          actions: <Widget>[
            TextButton(
              child: const Text('Annuler', style: TextStyle(color: Colors.red)),
              onPressed: () {
                //  Navigator.of(dialogContext).pop(); // Dismiss alert dialog
                Navigator.pop(dialogContext);
              },
            ),
```

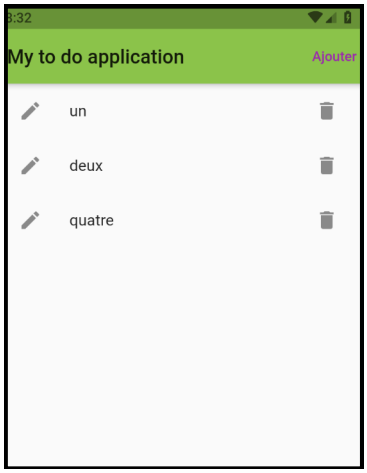
```
TextButton(
  child:
      const Text('Valider', style: TextStyle(color: Colors.blue)),
  onPressed: () {
    if (nouvelleTache != null) {
      Map<String, dynamic> map = {
        'id': elem?.id,
        'nom': nouvelleTache
      };
      Item item = Item(map);

      // add to bd (ajouter item) and set to state (recuperer)

      DatabaseClient()
          .update_or_insert(item)
          .then((value) => recuperer());
    }
    setState(() {
      nouvelleTache = null;
    });
```

- Il faut aussi modifier l'appel de la fonction ajouter dans le bouton 'Ajouter' qui est dans la appBar
- Remplacez onPressed: ajouter,

Par

```
onPressed: (() => ajouter(null)),
```

Voici le résultat final :



Pour ceux qui le souhaitent, le projet est disponible sur github à l'adresse :
https://github.com/Thierrynjike/TP2todo