Flutter

Training Assignments

Persist data with SQLite

Overview

In this lab, we are going to build a small Flutter app that uses SQLite to persist data.

The app has a floating button that can be used to show a bottom sheet. That bottom sheet contains 2 text fields corresponding to “title” and “description”. These text fields are used to create a new “item” or update an existing “item”.

The saved “items” are fetched from the SQLite database and displayed with a list view. There are an update button and a delete button associated with each “item”.

Database Structure

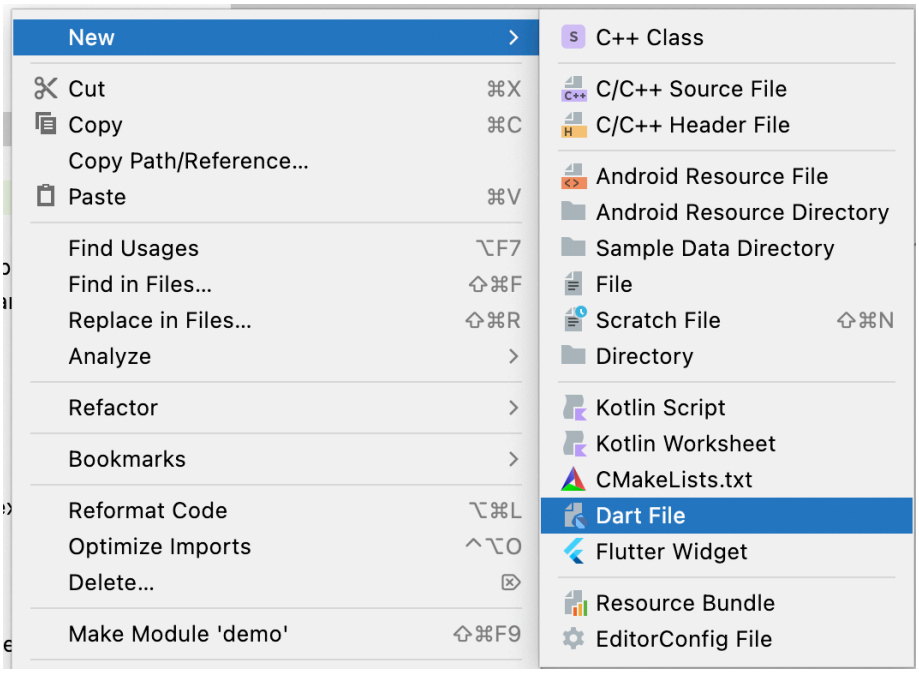
We are going to create an SQLite database called **demo.db**. It has only a single table named **items**. Below is the structure of the table:

Column	Type	Description
id	INTEGER	The id of an item and auto increment
title	TEXT	The name of an item
description	TEXT	The detail of an item
createdAt	TIMESTAMP	The time that the item was created. It will be automatically added by SQLite

Tasks

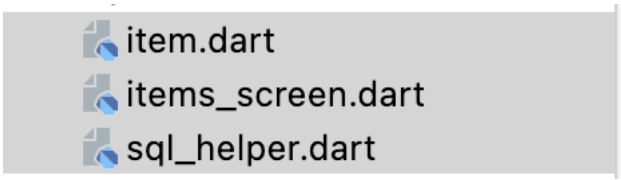
1. Create a new Flutter project.

+ In the **lib** folder, add a new file named **sql_helper.dart** and **item.dart**



+ In the lib folder, add a new Flutter Widget file named **items_screen.dart**

+ The project structure:



+ Install the **sqflite** plugin (**sqflite**: ^2.1.0+1)

```
dependencies:
  flutter:
    sdk: flutter

  english_words: ^4.0.0

  sqflite: ^2.1.0+1
```

2. Full code in **item.dart**

```dart
class Item {
  final int? id;
  final String? title;
  final String? description;
  final String? createdAt;

  Item({this.id, this.title, this.description, this.createdAt});

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

3. Full code in **sql_helper.dart**

```dart
class SQLHelper {
  // id: the id of a item
  // title, description: name and description of your activity
  // created_at: the time that the item was created.
  // It will be automatically handled by SQLite
  static Future<void> createItemTable(Database database) async {
    await database.execute('''CREATE TABLE items(
        id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        title TEXT,
        description TEXT,
        createdAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
      )''');
  }

  static Future<Database> db() async {
    return openDatabase(
      'demo.db',
      version: 1,
      onCreate: (Database database, int version) async {
        await createItemTable(database);
      },
    );
  }

  // Create new item (journal)
  static Future<int> createItem(Item item) async {
    final db = await SQLHelper.db();
```

```dart

    final id = await db.insert('items', item.toMap(),
        conflictAlgorithm: ConflictAlgorithm.replace);

    return id;
  }

  // Read all items (journals)
  static Future<List<Map<String, dynamic>>> getItems() async {
    final db = await SQLHelper.db();

    return db.query('items', orderBy: "id");
  }

  // Read a single item by id
  // The app doesn't use this method but I put here in case you want to see it
  static Future<List<Map<String, dynamic>>> getItem(int id) async {
    final db = await SQLHelper.db();

    return db.query('items', where: "id = ?", whereArgs: [id], limit: 1);
  }

  // Update an item by id
  static Future<int> updateItem(Item item) async {
    final db = await SQLHelper.db();

    final result = await db
        .update('items', item.toMap(), where: "id = ?", whereArgs: [item.id]);
    return result;
  }

  // Delete
  static Future<void> deleteItem(int id) async {
    final db = await SQLHelper.db();

    try {
      await db.delete("items", where: "id = ?", whereArgs: [id]);
    } catch (err) {
      debugPrint("Something went wrong when deleting an item: $err");
    }
  }
}
```

4. Full code in **item_screen.dart**

```dart
class ItemsScreen extends StatelessWidget {
  const ItemsScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: _HomePage(),
    );  // MaterialApp
  }
}

class _HomePage extends StatefulWidget {
  const _HomePage({Key? key}) : super(key: key);

  @override
  State<_HomePage> createState() => _HomePageState();
}

class _HomePageState extends State<_HomePage> {
  // All journals
  List<Map<String, dynamic>> _journals = [];

  bool _isLoading = true;

  // This function is used to fetch all data from the database
  Future<void> _refreshJournals() async {
    final data = await SQLHelper.getItems();

    setState(() {
      _journals = data;
      _isLoading = false;
    });
  }

  @override
  void initState() {
    super.initState();
    _refreshJournals(); // Loading the diary when the app starts
  }

  final TextEditingController _titleController = TextEditingController();
  final TextEditingController _descriptionController = TextEditingController();
```

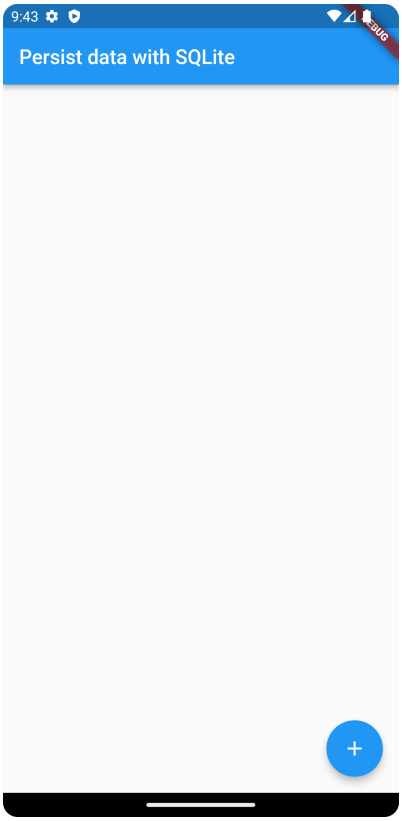
```dart

// This function will be triggered when the floating button is pressed
// It will also be triggered when you want to update an item
void _showForm(int? id) async {
  if (id != null) {
    // id == null -> create new item
    // id != null -> update an existing item
    final existingJournal =
        _journals.firstWhere((element) => element['id'] == id);
    _titleController.text = existingJournal['title'];
    _descriptionController.text = existingJournal['description'];
  }

  showModalBottomSheet(
      context: context,
      elevation: 5,
      isScrollControlled: true,
      builder: (_) => Container(
            padding: EdgeInsets.only(
              top: 15,
              left: 15,
              right: 15,
              // this will prevent the soft keyboard from covering the text fields
              bottom: MediaQuery.of(context).viewInsets.bottom + 120,
            ),  // EdgeInsets.only
            child: Column(
              mainAxisSize: MainAxisSize.min,
              crossAxisAlignment: CrossAxisAlignment.end,
              children: [
                TextFormField(
                  controller: _titleController,
                  decoration: const InputDecoration(hintText: 'Title'),
                ),  // TextFormField
                const SizedBox(
                  height: 10,
                ),  // SizedBox
                TextFormField(
                  controller: _descriptionController,
                  decoration: const InputDecoration(hintText: 'Description'),
                ),  // TextFormField
                const SizedBox(
                  height: 20,
                ),  // SizedBox
```

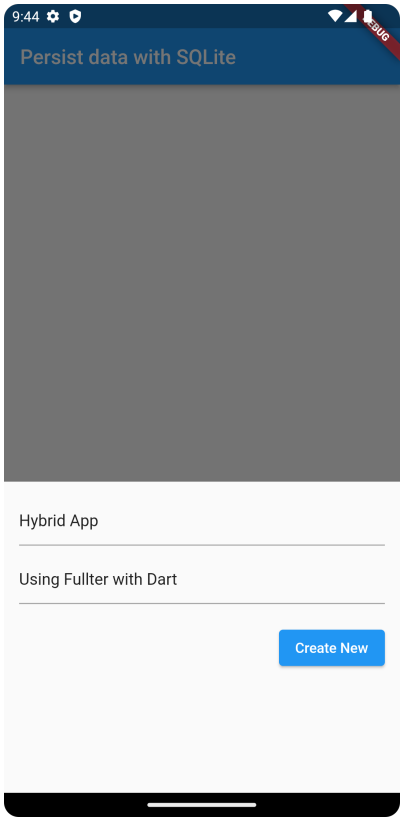
```
90                ElevatedButton(
91                  onPressed: () async {
92                    // Save new journal
93                    if (id == null) {
94                      await _addItem();
95                    }
96
97                    if (id != null) {
98                      await _updateItem(id);
99                    }
100
101                   // Clear the text fields
102                   _titleController.text = '';
103                   _descriptionController.text = '';
104
105                   // Close the bottom sheet
106                   // if you're not sure your widget is mounted.
107                   if (!mounted) return;
108
109                   Navigator.of(context).pop();
110                 },
111                 child: Text(id == null ? 'Create New' : 'Update'),
112               )  // ElevatedButton
113             ],
114           ),  // Column
115         ));  // Container
116   }
117
118   // Insert a new journal to the database
119   Future<void> _addItem() async {
120     await SQLHelper.createItem(Item(
121         title: _titleController.text,
122         description: _descriptionController.text));
123
124     _refreshJournals();
125   }
126
127   // Update an existing journal
128   Future<void> _updateItem(int id) async {
129     await SQLHelper.updateItem(Item(
130         id: id,
131         title: _titleController.text,
132         description: _descriptionController.text));
133
134     _refreshJournals();
135   }
```

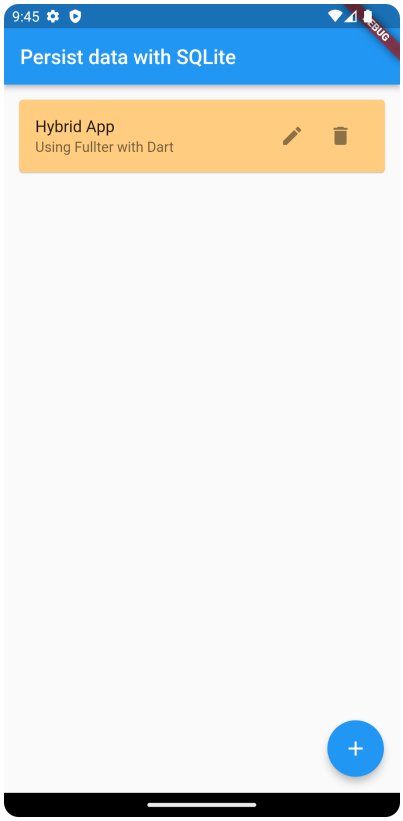
```dart

  // Delete an item
  Future<void> _deleteItem(int id) async {
    await SQLHelper.deleteItem(id);

    // if you're not sure your widget is mounted.
    if (!mounted) return;

    ScaffoldMessenger.of(context).showSnackBar(const SnackBar(
      content: Text('Successfully deleted a journal!'),
    ));  // SnackBar

    _refreshJournals();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Persist data with SQLite'),
      ),  // AppBar
      body: _isLoading
          ? const Center(
              child: CircularProgressIndicator(),
            )  // Center
          : ListView.builder(
              itemCount: _journals.length,
              itemBuilder: (context, index) => Card(
                color: Colors.orange[200],
                margin: const EdgeInsets.all(15),
                child: ListTile(
                    title: Text(_journals[index]['title']),
                    subtitle: Text(_journals[index]['description']),
                    trailing: SizedBox(
                      width: 100,
                      child: Row(
                        children: [
                          IconButton(
                            icon: const Icon(Icons.edit),
                            onPressed: () => _showForm(_journals[index]['id']),
                          ),  // IconButton
                          IconButton(
                            icon: const Icon(Icons.delete),
                            onPressed: () =>
                                _deleteItem(_journals[index]['id']),
                          ),  // IconButton
                        ],
                      ),  // Row
                    )),  // SizedBox, ListTile
              ),  // Card
            ),  // ListView.builder
```

```
187         floatingActionButton: FloatingActionButton(
188 +         child: const Icon(Icons.add),
189           onPressed: () => _showForm(null),
190         ),  // FloatingActionButton
191       );  // Scaffold
192     }
193   }
```

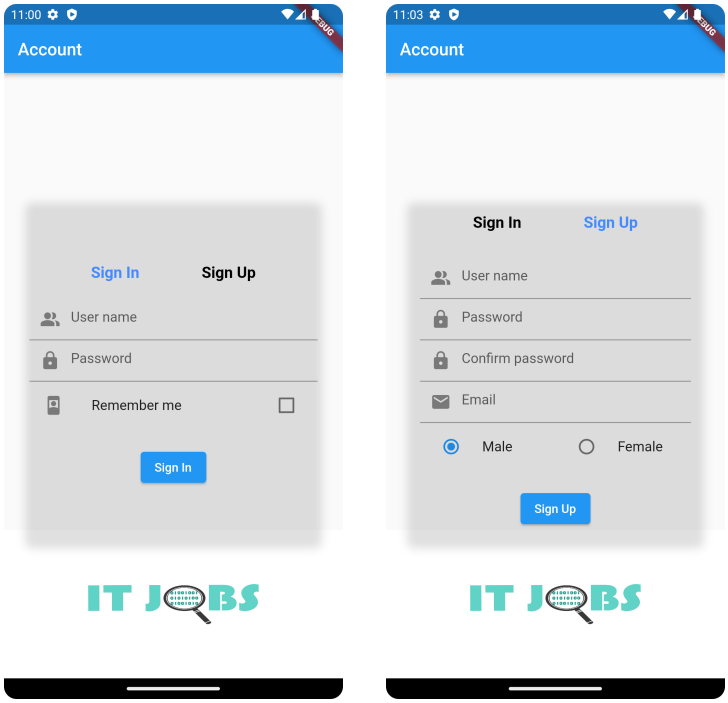
Run the app







Extra tasks



Database Structure

You are going to create an SQLite database called **news.db**. It has only a single table named **account**. Below is the structure of the table:

Column	Type	Description
username	TEXT	Primary key
password	TEXT	
email	TEXT	
gender	INTEGER	Boolean values are stored as integers 0 (female) and 1 (male)
createdAt	TIMESTAMP	The time that the item was created. It will be automatically added by SQLite

--THE END—