

**SEB-312 Mobile Application Development**

**LAB # 12**

**LAB Title**

|  |
| --- |
| Implementing Shared Preferences to store simple key-value pairs. Creating an SQLite database in a Flutter app to store and retrieve data. |

**Assessment of CLO: 04, PLO: 05**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** |  | | |
| **Roll No.** |  | | |
| **Semester** |  | **Session** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Perf. Level**  **Criteria** | **Excellent**  **(2.5)** | **Good**  **(2)** | **Satisfactory**  **(1.5)** | **Needs Improvement**  **(0 ~ 1)** | **Marks Obtained** |
| **1** | Project Execution & Implementation | Fully functional, optimized, and well-structured. | Minor errors, mostly functional. | Some errors, requires guidance. | Major errors, non-functional, or not Performed. |  |
| **2** | Results & Debugging  Or Troubleshooting | Accurate results with effective debugging  Or Troubleshooting. | Mostly correct, some debugging Or Troubleshooting needed. | Partial results, minimal debugging  Or Troubleshooting. | Incorrect results, no debugging Or Troubleshooting, or not attempted. |  |
| **3** | Problem-Solving & Adaptability  (VIVA) | Creative approach, efficiently solves challenges. | Adapts well, minor struggles. | Some adaptability, needs guidance. | Lacks innovation or no innovation, unable to solve problems. |  |
| **4** | Report Quality & Documentation | Clear, structured, with detailed visuals. | Mostly clear, minor gaps. | Some clarity issues, missing details. | Poorly structured, lacks clarity, or not submitted. |  |
| **Total Marks Obtained Out of 10** | | | | | |  |

**Experiment evaluated by**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor’s Name** | **Sidra Khatoon** | | |
| **Date** |  | **Signature** |  |

**Objective**

The objective of lab is implementing Shared Preferences to store simple key-value pairs and also creating an SQLite database in a Flutter app to store and retrieve data.

**Instructions**

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

**What is SharedPreferences?**

SharedPreferences is only used to store the user's small amount of data within the application, such as settings and data (not too much; if too much, then a database is required). SharedPreferences in Flutter are kept in XML format. It is also compatible with iOS and Android.

For small-value storage, we do not use SQLite since doing so requires writing complex code and supporting classes.

Shared Preferences is a tool used to store data locally on the user’s device. The data that can be stored can be a string, integer, Boolean, double, or list. Shared Preferences is often used to store data that is temporary or not too sensitive, such as login tokens, application language, or user settings.

**Why use Shared Preferences?**

Here are several reasons to use Shared Preferences:

* Data stored in Shared Preferences will not be lost when the application is uninstalled or updated.
* Shared Preferences can be accessed from anywhere in the application.
* Shared Preferences is relatively easy to use.

**How to use Shared** **Preferences in Flutter**

To use Shared Preferences in Flutter, you need to add the shared\_preferences package to your project. You can do this by running the following command in the terminal:



Once the package is added, you can create an instance of SharedPreferences by using the getInstance() method.



To store data in Shared Preferences, you can use the setString(), setInt(), setBool(), setDouble(), or setStringList() methods.



First we build basic UI for app in which we have one text field and button. There will text widget which show the stored value.

**Step 1:**

Open new project in flutter. Then install shared\_ preferences package by running command in terminal such as :

flutter pub add shared\_preferences

flutter pub get

**Step 2:**

Import library of shared preferences in main.dart file. shared\_preferences.dart is the library used to store and retrieve small amounts of data locally.

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

**Step 3:**

Now step the code main function and myApp class. runApp() initializes the app. MyApp returns a MaterialApp with a title and home screen.

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

debugShowCheckedModeBanner: false,

      title: 'Shared Preferences Demo',

      home: MyHomePage(),

    );

  }

}

**Step 4:**

Now, creating a stateful widget for MyHomePage class. StatefulWidget is used because we are handling dynamic state (saving and retrieving data).

class MyHomePage extends StatefulWidget {

  @override

  \_MyHomePageState createState() => \_MyHomePageState();

}

**Step 5:**

Define State and Variables for shared prefernces. In this code,\_key is the identifier used in SharedPreferences and \_value holds the data entered by the user.

class \_MyHomePageState extends State<MyHomePage> {

  final String \_key = "key";

  String \_value = "value";

}

**Step 6:**

Now we creating method of \_loadSavedValue() which is stored data getting from user input. This method is called in initState() method of stateful widget. initState() runs when the widget is first created. \_loadSavedValue() gets the stored value from SharedPreferences. If no value is found, it defaults to an empty string.

@override

  void initState() {

    super.initState();

    \_loadSavedValue();

  }

  // Function to load saved value from SharedPreferences

  void \_loadSavedValue() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    setState(() {

      \_value = prefs.getString(\_key) ?? ""; // Using ?? "" to handle null case

    });

  }

**Step 7:**

In this step, we building basic UI of our app. Our UI consists on three widgets. One is textfield which takes input form user. Second button which store data with shared preferences. Third is text widget, which is used to display the stored on UI of app.

* Text: Displays the current value.
* TextField: Accepts user input and updates \_value.
* ElevatedButton: Saves the current value when pressed.

@override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("Shared Preferences Demo"),

        centerTitle: true,

        backgroundColor: Colors.tealAccent,

      ),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: <Widget>[

            Text("Value: $\_value"),

            Padding(

              padding: const EdgeInsets.all(16.0),

              child: TextField(

                decoration: InputDecoration(

                  hintText: "Enter value",

                  border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue),

                ),

                ),

                onChanged: (value) {

                  setState(() {

                    \_value = value;

                  });

                },

              ),

            ),

            ElevatedButton(

              child: Text("Save"),

              onPressed: () {

                \_saveValue();

              },

            ),

          ],

        ),

      ),

    );

  }

**Step 8:**

We creating method of save value which store data in device storage. This method call in button widget.

* Gets an instance of SharedPreferences.
* Stores the current \_value under the \_key.

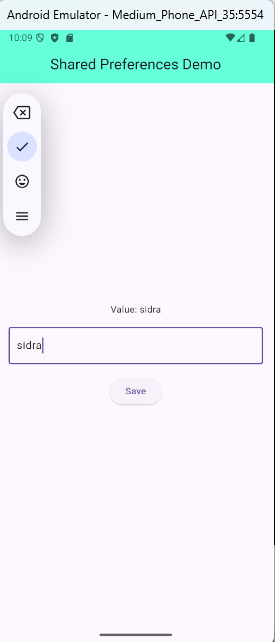
  // Function to save value to SharedPreferences

  void \_saveValue() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    await prefs.setString(\_key, \_value);

  }



**Complete Code:**

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'Shared Preferences Demo',

      home: MyHomePage(),

    );

  }

}

class MyHomePage extends StatefulWidget {

  @override

  \_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

  final String \_key = "key";

  String \_value = "value";

  @override

  void initState() {

    super.initState();

    \_loadSavedValue();

  }

  // Function to load saved value from SharedPreferences

  void \_loadSavedValue() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    setState(() {

      \_value = prefs.getString(\_key) ?? ""; // Using ?? "" to handle null case

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("Shared Preferences Demo"),

        centerTitle: true,

        backgroundColor: Colors.tealAccent,

      ),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: <Widget>[

            Text("Value: $\_value"),

            Padding(

              padding: const EdgeInsets.all(16.0),

              child: TextField(

                decoration: InputDecoration(

                  hintText: "Enter value",

                  border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue),

                ),

                ),

                onChanged: (value) {

                  setState(() {

                    \_value = value;

                  });

                },

              ),

            ),

            ElevatedButton(

              child: Text("Save"),

              onPressed: () {

                \_saveValue();

              },

            ),

          ],

        ),

      ),

    );

  }

  // Function to save value to SharedPreferences

  void \_saveValue() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    await prefs.setString(\_key, \_value);

  }

}

**Storing Data in SQLite database in Flutter**

In Flutter apps we need persistent data storage. This means storing data/information within the app itself, even when the app is closed or the device is turned off. Here’s where SQLite comes in.

**What is SQLite?**

SQLite is a lightweight relational database management system (RDBMS) that’s embedded within the application.

**Why Use SQLite in Flutter?**

Here are some key benefits of using SQLite with Flutter:

• Lightweight and Fast: SQLite boasts a small footprint and efficient performance, perfect for mobile apps where resources are limited.

• Serverless and Easy to Use: There’s no need to set up or manage a separate server. SQLite integrates seamlessly into your Flutter project.

• Cross-Platform Compatibility: SQLite works across Android, iOS, and even web platforms where Flutter is applicable.

**Getting Started with SQLite in Flutter**

To leverage SQLite in your Flutter applications, you’ll utilize the popular sqflite package. This package provides a Dart API for interacting with the SQLite database. It offers functionalities for:

* Creating and opening databases
* Defining tables and columns
* Performing CRUD operations (Create, Read, Update, Delete) on data
* Querying the database to retrieve specific information

**Step 1: Adding dependencies**

In order to use SQLite in Flutter we need two dependencies i.e. sqflite and path. Add these dependencies in the pubspec.yaml file.

flutter pub add sqflite

flutter pub add path

flutter pub get

**Step 2: Import Required Packages**

import 'package:flutter/material.dart';

import 'package:sqflite/sqflite.dart';

import 'package:path/path.dart';

sqflite: For SQLite database operations.

path: To construct paths for the database file.

**Step 3: Main Function and Root Widget**

MyApp is the root widget, setting up the Material app.

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'Flutter Demo',

      theme: ThemeData(

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

        useMaterial3: true,

      ),

      home: const MyHomePage(title: 'Flutter Demo Home Page'),

    );

  }

}

**Step 4: Home Page with State**

This sets up a StatefulWidget for our homepage because we'll update the UI after database changes.

class MyHomePage extends StatefulWidget {

  const MyHomePage({super.key, required this.title});

  final String title;

  @override

  State<MyHomePage> createState() => \_MyHomePageState();

}

**Step 5: Declare Variables and Controllers**

These controllers manage the text fields. The databaseService will handle all DB operations.

class \_MyHomePageState extends State<MyHomePage> {

  final databaseService = DatabaseService();

  String? reg\_no, name, cell\_no;

  TextEditingController tcRegNo = TextEditingController();

  TextEditingController tcName = TextEditingController();

  TextEditingController tcCellNo = TextEditingController();

}

**Step 6: Database Helper Class**

The Database Service will provide the following services:

• Create database

• Create tables

• Add/insert data in tables

• Retrieve data from tables

• Update data in tables

• Delete data from tables

**Initialize Database:**

class DatabaseService {

  static Database? db;

  Future<Database> getDatabase() async {

    final dbDirPath = await getDatabasesPath();

    final dbPath = join(dbDirPath, "student\_db.db");

    final database = await openDatabase(

        dbPath,

        onCreate: (db, version) {

          db.execute('''

            CREATE TABLE students(reg\_no TEXT, name TEXT NOT NULL, cell\_no TEXT)

          ''');

        },

        version: 1

    );

    return database;

  }

  Future<Database> getDB() async {

    if (db != null ) return db!;

    db = await getDatabase();

    return db!;

  }

**Insert Student**

Future<void> addStudent(String reg\_no, String name, cell\_no) async {

    final db = await getDB();

      await db.insert(

        "students",

        { "reg\_no": reg\_no,

          "name": name,

          "cell\_no": cell\_no

        }

    );

  }

**Get all Student**

  Future<List<Map<String, dynamic>>> getStudents() async{

    final db = await getDB();

    final data = await db.query("students");

    for(var d in data) {

      print(d);

    }

    return data;

  }

**Delete Student**

  Future<void> deleteStudent(String regNo) async {

    final db = await getDB();

    await db.delete(

      'students',

      where: 'reg\_no = ?',

      whereArgs: [regNo],

    );

  }

}

**Step 7: UI Layout in build()**

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("SQFlite Database Demo"),

        centerTitle: true,

        backgroundColor: Colors.cyan,

      ),

      body: Padding(

        padding: const EdgeInsets.all(16.0),

        child: Column(

          children: [

            TextField(

              decoration: InputDecoration(

                hintText: "Reg. No.",

                border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue)

                )

                ),

              controller: tcRegNo,

            ),

             SizedBox(height: 10),

            TextField(

              decoration: InputDecoration(

                hintText: "Name",

                  border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue)

                )

                ),

              controller: tcName,

            ),

             SizedBox(height: 10),

            TextField(

              decoration: InputDecoration(

                hintText: "Cell No",

                  border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue)

                )

                ),

              controller: tcCellNo,

            ),

             SizedBox(height: 10),

            ElevatedButton(onPressed: addRecord, child: Text("Save")),

             SizedBox(height: 10),

            ElevatedButton(onPressed: printData, child: Text("Print Data")),

              SizedBox(height: 10),

          // 👇👇👇 This is where you show the list of students

          Expanded(child: StudentsList()),

          ]

              ),

      ),

    );

  }

**Step 8: Add Record Function**

Adds a new student and clears input fields.

void addRecord() async {

    await databaseService.addStudent(tcRegNo.text, tcName.text, tcCellNo.text);

     // Rebuild the widget to show the new data

    // Clear fields after saving

    tcRegNo.clear();

    tcName.clear();

    tcCellNo.clear();

  }

**Step 9: Print Data Function**

Gets data from DB and forces a UI rebuild.

void printData() async {

  await databaseService.getStudents();

  setState(() {});

}

**Step 10: Print Data Function**

Deletes a student from the DB and refreshes the list.

  Future<void> deleteStudent(BuildContext context, String regNo) async {

  await databaseService.deleteStudent(regNo);

  if (!mounted) return;

  setState(() {});

  ScaffoldMessenger.of(context).showSnackBar(

    SnackBar(content: Text("Student deleted")),

  );

}

**Step 11: Displaying Students with FutureBuilder**

Uses FutureBuilder to load data from the DB. ListTile displays each record with a delete icon.

Widget StudentsList() {

  return FutureBuilder<List<Map<String, dynamic>>>(

    future: databaseService.getStudents(),

    builder: (context, snapshot) {

      if (snapshot.hasData) {

        if (snapshot.data!.isEmpty) {

          return Text("No students found.");

        }

        return ListView.builder(

          itemCount: snapshot.data!.length,

          itemBuilder: (context, index) {

            final student = snapshot.data![index];

            return ListTile(

              trailing: IconButton(onPressed: ()=> deleteStudent(context, student['reg\_no']),

              icon:  Icon(Icons.delete, color: Colors.red)),

              title: Text(student['name']),

              subtitle: Text('Reg No: ${student['reg\_no']}, Cell No: ${student['cell\_no']}',

              ),

            );

          },

        );

      } else if (snapshot.hasError) {

        return Text('Error: ${snapshot.error}');

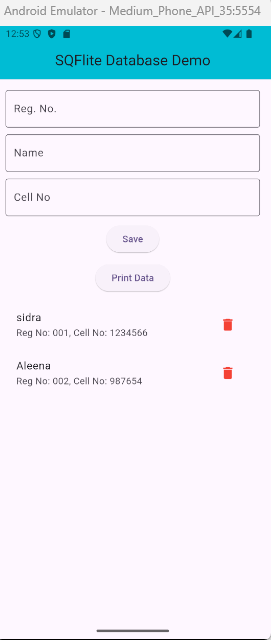
      }

      return CircularProgressIndicator();

    },

  );

}



**Complete Code**

import 'package:flutter/material.dart';

import 'package:sqflite/sqflite.dart';

import 'package:path/path.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'Flutter Demo',

      theme: ThemeData(

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

        useMaterial3: true,

      ),

      home: const MyHomePage(title: 'Flutter Demo Home Page'),

    );

  }

}

class MyHomePage extends StatefulWidget {

  const MyHomePage({super.key, required this.title});

  final String title;

  @override

  State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

  final databaseService = DatabaseService();

  String? reg\_no, name, cell\_no;

  TextEditingController tcRegNo = TextEditingController();

  TextEditingController tcName = TextEditingController();

  TextEditingController tcCellNo = TextEditingController();

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("SQFlite Database Demo"),

        centerTitle: true,

        backgroundColor: Colors.cyan,

      ),

      body: Padding(

        padding: const EdgeInsets.all(16.0),

        child: Column(

          children: [

            TextField(

              decoration: InputDecoration(

                hintText: "Reg. No.",

                border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue)

                )

                ),

              controller: tcRegNo,

            ),

             SizedBox(height: 10),

            TextField(

              decoration: InputDecoration(

                hintText: "Name",

                  border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue)

                )

                ),

              controller: tcName,

            ),

             SizedBox(height: 10),

            TextField(

              decoration: InputDecoration(

                hintText: "Cell No",

                  border: OutlineInputBorder(

                  borderSide: BorderSide(width: 4.0, color: Colors.blue)

                )

                ),

              controller: tcCellNo,

            ),

             SizedBox(height: 10),

            ElevatedButton(onPressed: addRecord, child: Text("Save")),

             SizedBox(height: 10),

            ElevatedButton(onPressed: printData, child: Text("Print Data")),

              SizedBox(height: 10),

          // 👇👇👇 This is where you show the list of students

          Expanded(child: StudentsList()),

          ]

              ),

      ),

    );

  }

void addRecord() async {

    await databaseService.addStudent(tcRegNo.text, tcName.text, tcCellNo.text);

     // Rebuild the widget to show the new data

    // Clear fields after saving

    tcRegNo.clear();

    tcName.clear();

    tcCellNo.clear();

  }

void printData() async {

  await databaseService.getStudents();

  setState(() {});

}

  Future<void> deleteStudent(BuildContext context, String regNo) async {

  await databaseService.deleteStudent(regNo);

  if (!mounted) return;

  setState(() {});

  ScaffoldMessenger.of(context).showSnackBar(

    SnackBar(content: Text("Student deleted")),

  );

}

Widget StudentsList() {

  return FutureBuilder<List<Map<String, dynamic>>>(

    future: databaseService.getStudents(),

    builder: (context, snapshot) {

      if (snapshot.hasData) {

        if (snapshot.data!.isEmpty) {

          return Text("No students found.");

        }

        return ListView.builder(

          itemCount: snapshot.data!.length,

          itemBuilder: (context, index) {

            final student = snapshot.data![index];

            return ListTile(

              trailing: IconButton(onPressed: ()=> deleteStudent(context, student['reg\_no']),

              icon:  Icon(Icons.delete, color: Colors.red)),

              title: Text(student['name']),

              subtitle: Text('Reg No: ${student['reg\_no']}, Cell No: ${student['cell\_no']}',

              ),

            );

          },

        );

      } else if (snapshot.hasError) {

        return Text('Error: ${snapshot.error}');

      }

      return CircularProgressIndicator();

    },

  );

}

}

class DatabaseService {

  static Database? db;

  Future<Database> getDatabase() async {

    final dbDirPath = await getDatabasesPath();

    final dbPath = join(dbDirPath, "student\_db.db");

    final database = await openDatabase(

        dbPath,

        onCreate: (db, version) {

          db.execute('''

            CREATE TABLE students(reg\_no TEXT, name TEXT NOT NULL, cell\_no TEXT)

          ''');

        },

        version: 1

    );

    return database;

  }

  Future<Database> getDB() async {

    if (db != null ) return db!;

    db = await getDatabase();

    return db!;

  }

  Future<void> addStudent(String reg\_no, String name, cell\_no) async {

    final db = await getDB();

      await db.insert(

        "students",

        { "reg\_no": reg\_no,

          "name": name,

          "cell\_no": cell\_no

        }

    );

  }

  Future<List<Map<String, dynamic>>> getStudents() async{

    final db = await getDB();

    final data = await db.query("students");

    for(var d in data) {

      print(d);

    }

    return data;

  }

  Future<void> deleteStudent(String regNo) async {

    final db = await getDB();

    await db.delete(

      'students',

      where: 'reg\_no = ?',

      whereArgs: [regNo],

    );

  }

}

# **Assessment:**

**Task 1: Application using shared preferences**

Create a simple app like this and store one more value of email with help TextField and show this email locally.

**Task 2: Basics of SQLite**

1. What is SQLite, and why is it suitable for mobile applications?
2. Explain the difference between SQLite and other relational databases like MySQL or PostgreSQL.
3. Mention three features of SQLite.

**Task 3: SQLite in Flutter**

1. How does Flutter support SQLite database integration?
2. What is the role of the sqflite package in Flutter?
3. Explain the lifecycle of a database in a Flutter app.

**Task 4: SQL Commands**

1. Write the SQL command to:
2. Create a table named users with columns id (INTEGER, PRIMARY KEY), name (TEXT), and age (INTEGER).
3. Insert a row into the users table.
4. Retrieve all users above the age of 25.
5. Update the name of a user where id = 1.
6. Delete a user where id = 3.