TIERS LIMITED SUMMER INTERNSHIP 2024

MOBILE APP DEVELOPMENT

SQLite

CLASS # 16

TIERS
Limited

Overview

In this class, we will learn about SQLite, a powerful and efficient database engine that can be used within Flutter applications for local data storage. We'll cover the basics of setting up and using SQLite in Flutter, including creating databases, performing CRUD (Create, Read, Update, Delete) operations, and managing database connections.

Objectives

- Understand what SQLite is and its use cases in Flutter applications.
- Set up SQLite in a Flutter project.
- Learn how to create, read, update, and delete data in an SQLite database.
- Manage database connections and handle asynchronous operations.

Introduction to SQLite

SQLite is a lightweight, disk-based database that doesn't require a separate server process and allows access to the database using a nonstandard variant of the SQL query language. SQLite is used in mobile applications for local storage due to its efficiency and ease of use.

Step 1: Dependencies

```
sqflite: ^latest version
path_provider: ^latest version
```

Step 2: Create a Student Class

This class represents a student with an ID, name, age, and semester. It includes methods for serializing and deserializing the student object to and from JSON.

```
class Student {
  final String id;
  final String name;
  final int age;
  final String semester;

  Student({
    required this.id,
    required this.name,
    required this.age,
    required this.semester,
  });

  Map<String, dynamic> toJson() => {
        'id': id,
        'name': name,
        'age': age,
        'semester': semester,
      };
  factory Student.fromJson(Map<String, dynamic> json) => Student(
        id: json['id'],
        name: json['name'],
        age: json['age'],
        semester: json['semester'],
      );

  String toJsonString() => json.encode(toJson());

  factory Student.fromJsonString(String jsonString) =>
Student.fromJson(json.decode(jsonString));

}
```

Step 3: Create DatabaseHelper Class

This class handles all the SQLite operations, such as adding, retrieving, and removing student data.

```
  Future init() async {
    final Directory appDocumentsDir = await getApplicationDocumentsDirectory();
    String dirPath = appDocumentsDir.path;
    String dbPath = dirPath + "/students.db";

    db = await openDatabase(
      dbPath,
      version: 1,
      onCreate: (db, version) async {
        await db.execute(
            "CREATE TABLE $tableName ($colId INTEGER PRIMARY KEY, $colName
               TEXT, $colAge INTEGER, $colSemester INTEGER);"
        );
      },
    );
  }

  Future saveStudentRecord(Student student) async {
    await db!.insert(tableName, student.toJson());
  }

  Future removeStudentRecord(int id) async {
    await db!.delete(tableName, where: "$colId=?", whereArgs: [id]);
  }

  Future<List<Student>> getAllStudents() async {
    List<Student> studentsList = [];
    List<Map<String, dynamic>> allRows = await db!.query(tableName);

    for (var row in allRows) {
      studentsList.add(Student.fromJson(row));
    }
    return studentsList;
  }
```

Step 4: Main Application

The main application sets up a `ListView.builder` to display the list of students and includes an add button to add new students.

```
List<Student> students = [];
  final dbHelper = DatabaseHelper();
  bool isLoading = true;

  @override
  void initState() {
    super.initState();
    dbHelper.init().then((_) {
      _fetchStudents();
    });
  }

  void _fetchStudents() async {
    final fetchedStudents = await dbHelper.getAllStudents();
    setState(() {
      students = fetchedStudents;
      isLoading = false;
    });
  }

  void _addStudent() async {
    await dbHelper.saveStudentRecord(Student(id: 1, name: 'John Doe', age: 20,
semester: '4th'));
    _fetchStudents();
  }

  ListView.builder(
        itemCount: students.length,
        itemBuilder: (context, index) {
          final student = students[index];
          return ListTile(
            title: Text(student.name),
            subtitle:Text(
                'Age: ${student.age}, Semester: ${student.semester}'
            ),
          );
        },
   ),
```

Explanation of Key Components

- **Student Class**: This class includes methods for serializing and deserializing a student object to and from JSON.
- **DatabaseHelper Class**: Manages adding, retrieving, and removing students in Database.
- **StudentScreen Class**: Displays a list of students and includes an add button for adding new students.

Key Points

- **initState()**: Initializes the state and loads the student data.
- **_fetchStudents()**: Asynchronously fetches the student data from Database.
- **_addStudent()**: Adds a new student to Database and updates the list.
- **ListView.builder**: Displays the list of students dynamically.
- **CircularProgressIndicator**: Shows a loading indicator while the data is being fetched.

By following these steps, you can efficiently use SQLite to manage offline data storage in your Flutter applications.

Exercises:

Implement Add, Edit, Remove and Load Operations using SQLite.

- Use Dialogs for add and edit operations
- Use Dialog "Are you sure to delete?" Yes or No when user press on delete button.