Lab: Using Local Storage with Flutter

Estimated time: 30 mins

Welcome to this hands-on lab, where you will develop a Student Management application using Flutter. Initially, you will create a basic app to manage student data. In the second part, you will add local storage to ensure data persistence across sessions.

Objectives

After completing this lab, you will be able to:

- Implement CRUD operations for student data
- Integrate local storage to save data persistently

About Cloud IDE

Running the lab

This lab is designed to be completed using the Skills Network Cloud IDE environment. You will write and execute Flutter code directly within the IDE.

About Skills Network Cloud IDE

Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands-on labs for course and project-related labs. It is an open-source IDE (Integrated Development Environment).

Important notice about this lab environment

Please be aware that sessions for this lab environment are not persistent. Every time you connect to this lab, a new environment is created for you. Any data you may have saved in the earlier session will be lost. Plan to complete these labs in a single session to avoid losing your data.

Step 1: Create the Flutter project

You will start by creating a new Flutter project using the command line provided by the Cloud IDE. This step sets up the project with all necessary Flutter files and directory structure for developing the application.

```
flutter create student_manager
cd student_manager
```

- `flutter create student_manager`: This command creates a new Flutter project named `student_manager`.
- `cd student_manager`: Change directory to the newly created project folder to start developing the application.

Step 2: Define the student model

In this step, you will define a model for student data. This model will help structure the data used throughout the application. You'll create a `Student` class with properties for ID, first name, last name, age, and major.

```
// file: lib/student_model.dart
class Student {
  final int id;
  final String firstName;
  final String lastName;
  final int age;
  final String major;
  Student({this.id, required this.firstName, required this.lastName, required this.age, required this.major});
  factory Student.fromJson(Map<String, dynamic> json) {
    return Student(
      id: json['id'],
      firstName: json['firstName'],
      lastName: json['lastName'],
      age: json['age'],
      major: json['major'],
    );
  }
  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'firstName': firstName,
      'lastName': lastName,
      'age': age,
      'major': major,
    };
  }
}
```

- `class Student`: Defines a class named `Student`.
- `final`: Fields are immutable.
- `factory Student.fromJson`: A factory constructor for creating a `Student` instance from a map.
- `Map<String, dynamic> toJson()`: A method to convert a `Student` instance to a map.

Step 3: Create the student provider

This step involves creating a provider for managing the state of student data in your app. The provider will include functions to add, remove, and update students.

```
// file: lib/student_provider.dart
import 'package:flutter/material.dart';
import 'student_model.dart';
class StudentProvider with ChangeNotifier {
  List<Student> _students = [];
  List<Student> get students => _students;
```

```
  void addStudent(Student student) {
    _students.add(student);
    notifyListeners();
  }
  void removeStudent(int id) {
    _students.removeWhere((student) => student.id == id);
    notifyListeners();
  }
  void updateStudent(Student updatedStudent) {
    var index = _students.indexWhere((student) => student.id == updatedStudent.id);
    if (index != -1) {
      _students[index] = updatedStudent;
      notifyListeners();
    }
  }
}
```

- `class StudentProvider with ChangeNotifier`: A class to manage the list of students.
- `notifyListeners()`: Notifies listeners about changes to rebuild the UI.
- Methods to add, remove, and update student data in the provider's list.

Step 4: Create the Edit Student screen

This step involves creating a screen to add or edit student details. You will implement form fields to capture student information.

```
// file: lib/edit_student_screen.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'student_model.dart';
import 'student_provider.dart';
class EditStudentScreen extends StatefulWidget {
  final Student? student;
  const EditStudentScreen({Key? key, this.student}) : super(key: key);
  @override
  _EditStudentScreenState createState() => _EditStudentScreenState();
}
class _EditStudentScreenState extends State<EditStudentScreen> {
  final TextEditingController _firstNameController = TextEditingController();
  final TextEditingController _lastNameController = TextEditingController();
  final TextEditingController _ageController = TextEditingController();
  final TextEditingController _majorController = TextEditingController();
  @override
  void initState() {
    super.initState();
    if (widget.student != null) {
      _firstNameController.text = widget.student!.firstName;
      _lastNameController.text = widget.student!.lastName;
      _ageController.text = widget.student!.age.toString();
      _majorController.text = widget.student!.major;
    }
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.student == null ? 'Add Student' : 'Edit Student'),
      ),
      body: Padding(
```

```
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: <Widget>[
              TextField(
                controller: _firstNameController,
                decoration: InputDecoration(labelText: 'First Name'),
              ),
              TextField(
                controller: _lastNameController,
                decoration: InputDecoration(labelText: 'Last Name'),
              ),
              TextField(
                controller: _ageController,
                keyboardType: TextInputType.number,
                decoration: InputDecoration(labelText: 'Age'),
              ),
              TextField(
                controller: _majorController,
                decoration: InputDecoration(labelText: 'Major'),
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () => _saveStudent(context),
                child: Text('Save Student'),
              ),
            ],
          ),
        ),
      );
    }
    void _saveStudent(BuildContext context) {
      final student = Student(
        id: widget.student?.id ?? DateTime.now().millisecondsSinceEpoch,
        firstName: _firstNameController.text,
        lastName: _lastNameController.text,
        age: int.parse(_ageController.text),
        major: _majorController.text,
      );
      if (widget.student == null) {
        Provider.of<StudentProvider>(context, listen: false).addStudent(student);
      } else {
        Provider.of<StudentProvider>(context, listen: false).updateStudent(student);
      }
      Navigator.pop(context);
    }
    @override
    void dispose() {
      _firstNameController.dispose();
      _lastNameController.dispose();
      _ageController.dispose();
      _majorController.dispose();
      super.dispose();
    }
  }
```

- The screen allows for adding a new student and editing an existing one, determined by whether a student object is passed to it.
- Uses `TextEditingController` to manage form inputs.
- `_saveStudent`: A method that either adds a new student or updates an existing one in the provider and then navigates back.

Step 5: Create the home screen

In this step, you will develop the home screen that lists all students. Each student's information will be displayed, and you can navigate to the edit screen by tapping on a student.

```
// file: lib/home_screen.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'student_model.dart';
import 'student_provider.dart';
import 'edit_student_screen.dart';
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Student List'),
      ),
      body: Consumer<StudentProvider>(
        builder: (context, provider, child) {
          return ListView.builder(
            itemCount: provider.students.length,
            itemBuilder: (context, index) {
              final student = provider.students[index];
              return ListTile(
                title: Text('${student.firstName} ${student.lastName}'),
                subtitle: Text('Age: ${student.age} - Major: ${student.major}'),
                onTap: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) => EditStudentScreen(student: student),
                    ),
                  );
                },
              );
            },
          );
        },
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => EditStudentScreen(),
            ),
          );
        },
        child: Icon(Icons.add),
        tooltip: 'Add Student',
      ),
    );
  }
}
```

- Uses `Consumer<StudentProvider>` to listen to student list updates
- Implements a `ListView.builder` to render each student as a list item
- Defines a floating action button to navigate to the `EditStudentScreen` for adding a new student

Step 6: Test without local storage

Initially, test the application without local storage to observe that students are not retained after a restart.

- **Task**: Add a few students using the app, and press `r` in the terminal to hot reload the application. Your screen will reset, and you will lose all the students previously entered! Let's fix this by storing students in local storage.

Add Local Storage

Add local storage to the project

Add localstorage to the Flutter project with the following command:

```
flutter pub add localstorage
```

Initialize local storage in the main file

Import the package in `main.dart` file.

```
import 'package:localstorage/localstorage.dart';
```

Initialize local storage as follows:

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await initLocalStorage();
  runApp(MyApp(localStorage: localStorage));
}
class MyApp extends StatelessWidget {
  final LocalStorage localStorage;
  const MyApp({Key? key, required this.localStorage}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    // Wrap the MaterialApp with the ChangeNotifierProvider
    return ChangeNotifierProvider<StudentProvider>(
      create: (_) => StudentProvider(storage: localStorage),
      child: MaterialApp(
        title: 'Student Management App',
        home: HomeScreen(), // Assuming HomeScreen leads to EditStudentScreen
      ),
    );
  }
}
```

Change the provider to work with local storage instead of memory

Add a function to load students from local storage:

```
void _loadStudentsFromStorage() async {
  var data = storage.getItem('students');
  if (data != null) {
    _students = List<Student>.from((jsonDecode(data) as List)
        .map((item) => Student.fromMap(item as Map<String, dynamic>)));
    notifyListeners();
  }
}
```

Create a method to save students in local storage:

```
void _saveToStorage() {
  storage.setItem('students',
      jsonEncode(_students.map((student) => student.toMap()).toList()));
}
```

You can use these two methods in the other methods to use the storage:

```
void addStudent(Student student) {
    _students.add(student);
    _saveToStorage();
    notifyListeners();
  }
  void deleteStudent(int id) {
    _students.removeWhere((element) => element.id == id);
    _saveToStorage();
    notifyListeners();
  }
  void updateStudent(Student student) {
    var index = _students.indexWhere((element) => element.id == student.id);
    if (index != -1) {
      _students[index] = student;
      _saveToStorage();
      notifyListeners();
    }
```

```
}
```

Complete code

Here is the complete code if you get stuck along the way:

1. lib/student_model.dart

▶ Click here for code

1. lib/student_provider.dart

▶ Click here for code

3. lib/edit_student_screen.dart

▶ Click here for code

5. lib/home_screen.dart

▶ Click here for code

6. lib/main.dart

▶ Click here for code

Conclusion and next steps

Here are some other things you can try:

- Add a way to update existing students. Changing a student in the app should make the same change to the local storage array.
- Add other user preferences to the storage. For example, the user should be able sort the student by name or by age. This preference can be stored in the local storage.

Congratulations on completing the initial part of this lab!

Author(s)

Skills Network