CMPS 312 Mobile Application Development –
Lab 6: Building Dynamic Lists in Flutter: Search, Filter, and State Management

Objective

In this lab, you will learn to display items in a scrollable list, and allows users to interact with this list through searching, filtering, and sorting functionalities. You will also learn how to use states to make your application responsive to user interactions.

By the end of this lab, you will be able to:

- **Structure a Flutter Project** by organizing folders and files for models, repositories, and UI components.
- **Read and Deserialize JSON Data** from the assets folder to display it in the application.
- **Build Interactive UIs** with widgets such as **ListView**, Dropdown Widgests , Scaffold , AppBar , Card, and TextField
- **Use basic state management in StatefulWidget** to handle app state and update the UI based on user interactions.
- **Implement Search, Filter , and Sort Functionalities** on a list of items using Flutter widgets and state management.

Overview

This lab consists of two parts:

1. **Part A – TipBuddy**: Last week, you designed the layout for the Tipping App using basic Flutter widgets. This week, you will integrate state management to handle user inputs and calculate tips dynamically based on user-entered values.
2. **Part B - Stadiums App**: This is a new project where you will build a Stadiums App from scratch. You will use state management with the Provider package to implement functionalities such as displaying a list of stadiums, searching by name or city, filtering by status, and sorting based on different criteria.

Part A – TipBuddy

In this part of the lab, you will build on the tipping app you designed last week. You will integrate states into your existing layout to create a dynamic application that calculates the tip amount based on user inputs. This will help you understand how to use states in your application and update the UI when the state/data of your app changes.

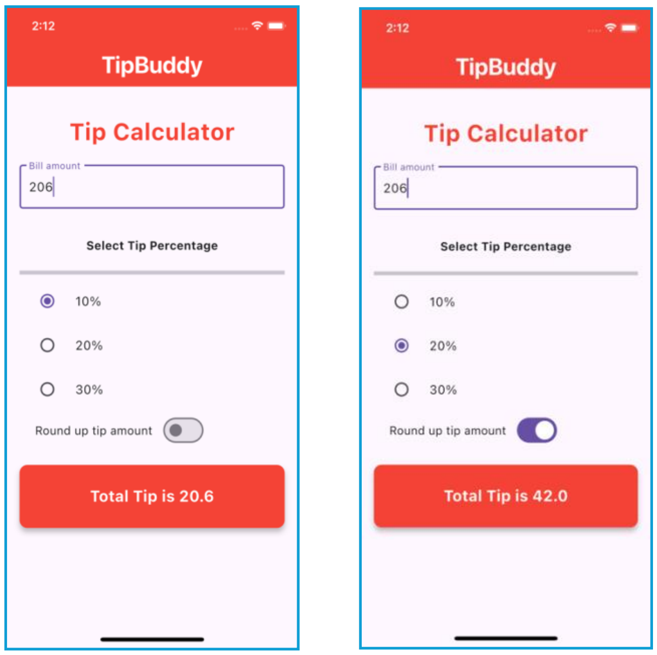
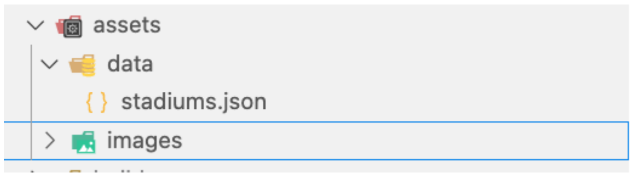


Figure 1 TipBuddy App

PART B – Stadium App

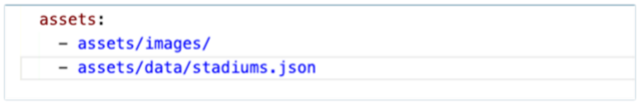
In this part of the lab, you will build a new Flutter application called StadiumsApp that displays a list of stadiums with interactive features such as searching, filtering, and sorting. You will implement state management using the Provider package to manage the state of the stadium data and keep the UI in sync with the app's state.

1. Create a new Flutter project named StadiumsApp.
2. Create an assets/data folder in your project directory and add the provided **stadiums.json** file.
3. Copy the provided images folder and paste it inside the assets folder



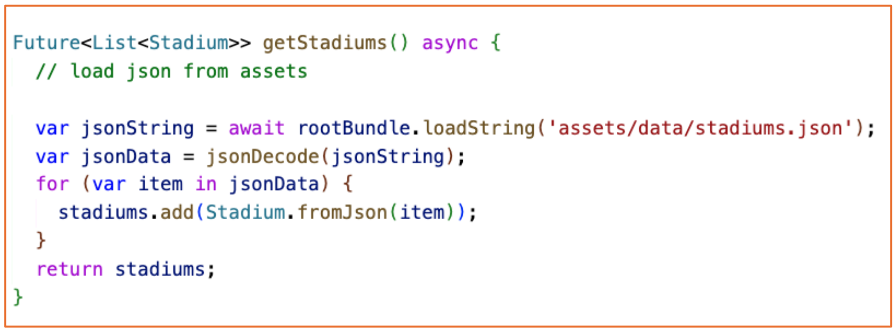
```
assets
  data
    stadiums.json
  images
```

4. Update the pubspec.yaml file to include the assets directory so that the JSON file can be accessed within the app. Do the same for the images.



```
assets:
  - assets/images/
  - assets/data/stadiums.json
```

5. Create a Stadium class inside a model folder.
    - Define fields that match the properties in the JSON file (e.g., *name, city, status, seatingCapacity, imageName*).
    - Implement a constructor and a fromJson method to convert JSON objects into Stadium instances.
6. Create a **StadiumRepo** class inside a repo folder.This class will handle loading and filtering stadium data from the stadiums.json file. Implement the following functionalities inside the class:
    - Loading Stadiums: Write a method getStadiums() that reads the stadium data from the JSON file and converts it into a list of Stadium objects.
    - Filtering Stadiums: Implement a filterStadiums(String query) method to filter the list of stadiums based on the name, city, or status using the where method.



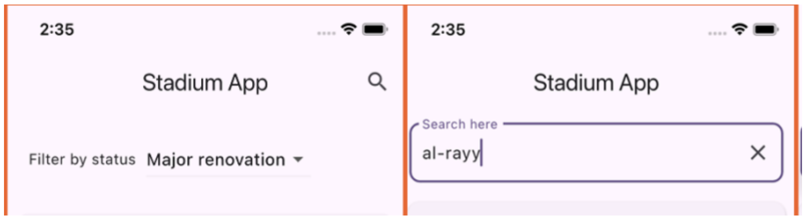
```dart
Future<List<Stadium>> getStadiums() async {
  // load json from assets

  var jsonString = await rootBundle.loadString('assets/data/stadiums.json');
  var jsonData = jsonDecode(jsonString);
  for (var item in jsonData) {
    stadiums.add(Stadium.fromJson(item));
  }
  return stadiums;
}
```

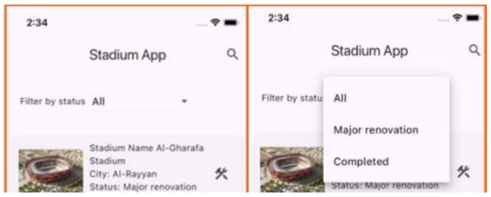
UI Implementation:

7. Create a scaffold widget that has an App Bar and a body as shown in figure 2.
8. In the body create a column that has a **ListView**: Use a `ListView.builder` to display the list of stadiums. Each stadium should be displayed inside a `Card` widget with the following details:
    - Stadium name
    - City
    - Status
    - Seating capacity
    - Stadium image (use placeholder images if needed)
9. Create **AppBar**: Add a search icon to the AppBar. When the user taps the icon, display a `TextField` for the search functionality. This textfield should be inside the body.
10. **Search Bar**: Implement a `TextField` widget that captures user input and filters the list of stadiums in real-time based on the entered name or city. The search bar should be inside

the column in the body and should only be displayed when the user clicks on the search icon. Also add a dismiss X icon that hides the search text field.



11. **Dropdown Filter**: Add a `DropdownButton` widget to filter stadiums based on their status. The available options should include "All", "Completed", and "Under Renovation".



The complete app is shown below in figure 2.

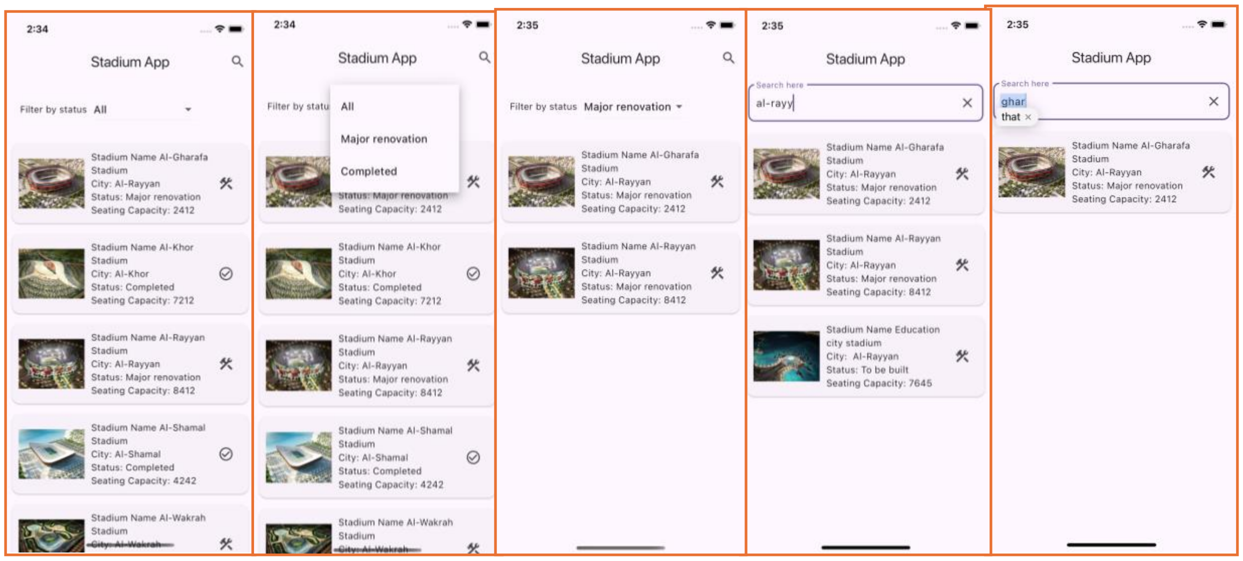


Figure 2 Stadium App Design and Features