



**Mobile Application Development
(SOFE 4640U)**

**Assignment 3: App Development Using Flutter - Food
Ordering App**

Name	Student ID
Rohan Radadiya	100704614

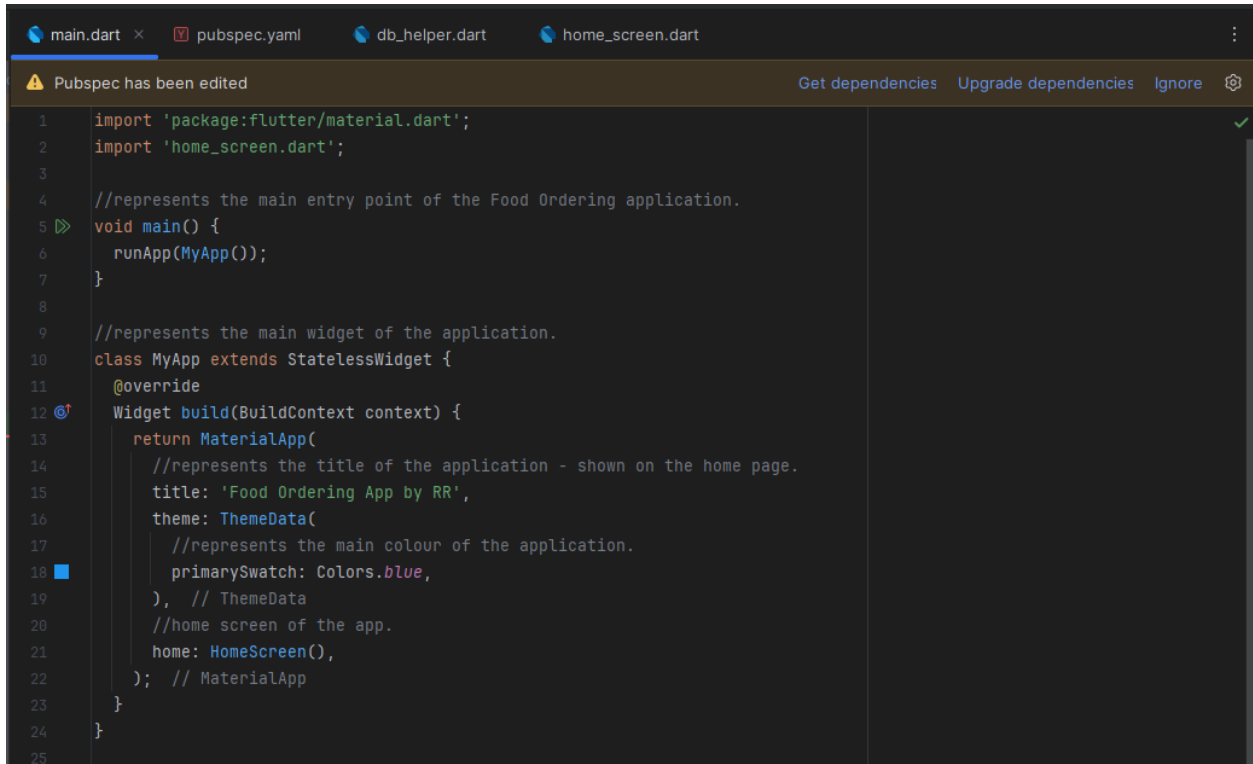
Due Date: November 27, 2024

GitHub Link:

<https://github.com/rohanradadiya/MOBILE-APPLICATIONS-ASSIGNMENT-3>

Code Snippets and Explanations:

File: main.dart:



```
1 import 'package:flutter/material.dart';
2 import 'home_screen.dart';
3
4 //represents the main entry point of the Food Ordering application.
5 void main() {
6   runApp(MyApp());
7 }
8
9 //represents the main widget of the application.
10 class MyApp extends StatelessWidget {
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       //represents the title of the application - shown on the home page.
15       title: 'Food Ordering App by RR',
16       theme: ThemeData(
17         //represents the main colour of the application.
18         primarySwatch: Colors.blue,
19       ), // ThemeData
20       //home screen of the app.
21       home: HomeScreen(),
22     ); // MaterialApp
23   }
24 }
25
```

The screenshot above represents the “main.dart” file. It contains the main() function, which is the application's main entry point. It calls “run app” to start the application. The MyApp class is the application's main root widget, which sets up the application with a title and home screen.

File: home_screen.dart:

```

28
29 //represents the method that fetches all food items from the database.
30 Future<void> _fetchFoodItems() async {
31   final dbHelper = DatabaseHelper();
32   final data = await dbHelper.queryAllFoodItems();
33   setState(() {
34     _foodItems = data;
35     _selectedItems = List.filled(data.length, false);
36   });
37 }
38
39 //represents the method to save the order with selected food items and target cost.
40 Future<void> _saveOrder() async {
41   final targetCost = _targetCostController.text.isNotEmpty
42     ? double.tryParse(_targetCostController.text)
43     : null;
44   if (targetCost == null) {
45     ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Please enter a valid target cost.')));
46     return;
47   }
48

```

The snippet above is of the `_fetchFoodItems()` method. It fetches all of the food items from the database with the use of the `DatabaseHelper` class. This also updates the `_foodItems` and `_selectedItems` lists.

```

92
93 //represents a method to show a dialog to add a new food item.
94 void _showAddFoodItemDialog() {
95   final _nameController = TextEditingController();
96   final _costController = TextEditingController();
97
98   showDialog(
99     context: context,
100     builder: (context) {
101       return AlertDialog(
102         title: Text('Add Food Item'),
103         content: Column(
104           mainAxisAlignment: MainAxisAlignment.min,
105           children: [
106             TextField(
107               controller: _nameController,
108               decoration: InputDecoration(labelText: 'Name'),
109             ), // TextField
110             TextField(
111               controller: _costController,
112               decoration: InputDecoration(labelText: 'Cost'),
113               keyboardType: TextInputType.number,
114             ), // TextField
115           ],
116         ), // Column
117         actions: [
118           TextButton(
119             onPressed: () {

```

The snippet above represents the `_showAddFoodItemDialog()` method, which shows a dialog for adding a new food item. The dialog also contains text fields in order to display the food item's name and associated cost. Also, when the “Add” button is clicked, a new food item is then inserted into the database afterward where the list of food items is finally refreshed.

```

143
144 //represents a method to show dialog to update an existing food item.
145 void _showUpdateFoodItemDialog(int id, String name, double cost) {
146     final _nameController = TextEditingController(text: name);
147     final _costController = TextEditingController(text: cost.toString());
148
149     showDialog(
150         context: context,
151         builder: (context) {
152             return AlertDialog(
153                 title: Text('Update Food Item'),
154                 content: Column(
155                     mainAxisAlignment: MainAxisAlignment.min,
156                     children: [
157                         TextField(
158                             controller: _nameController,
159                             decoration: InputDecoration(labelText: 'Name'),
160                         ), // TextField
161                         TextField(
162                             controller: _costController,
163                             decoration: InputDecoration(labelText: 'Cost'),
164                             keyboardType: TextInputType.number,
165                         ), // TextField
166                     ],
167                 ), // Column
168                 actions: [
169                     TextButton(
170                         onPressed: () {

```

The `_showUpdatedFoodItemDialog()` method is created to show a dialog for updating an existing food item. The dialog is already filled with the name and cost of the food item. In addition, when the “Update” button is clicked, the food item is then updated in the database, and the list of food items is refreshed.

```

39 //represents the method to save the order with selected food items and target cost.
40 Future<void> _saveOrder() async {
41     final targetCost = _targetCostController.text.isNotEmpty
42       ? double.tryParse(_targetCostController.text)
43       : null;
44     if (targetCost == null) {
45       ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Please enter a valid target cost.')));
46       return;
47     }
48
49     final selectedItems = _foodItems
50       .asMap()
51       .entries
52       .where((entry) => _selectedItems[entry.key])
53       .map((entry) => entry.value['name'] as String)
54       .toList();
55     final totalCost = _foodItems
56       .asMap()
57       .entries
58       .where((entry) => _selectedItems[entry.key])
59       .map((entry) => entry.value['cost'] as double)
60       .fold<double>(0.0, (a, b) => a + b);
61
62     if (selectedItems.isEmpty) {
63       ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Please select at least one food item.')));
64       return;
65     }

```

This method is the `_saveOrder()` method, which saves the order with the selected food items and a target cost. It also checks if the target cost is valid and if any food items are actually selected.

If the total cost of the food items selected tends to exceed the target cost, then an error message is displayed to the user that the cost of the food exceeds the target cost. After these steps, the method helps to either update an existing order or insert a new order into the database.

```

264
265 @override
266 Widget build(BuildContext context) {
267     return Scaffold(
268       appBar: AppBar(
269         //title that shows on the top banner in the application.
270         title: Text('Food Ordering App by RR'),
271         //colour attribute for the application title.
272         backgroundColor: Colors.lightBlueAccent,
273         actions: [
274           IconButton(
275             icon: Icon(Icons.search),
276             onPressed: _showQueryOrderDialog,
277           ), // IconButton

```

The snippet above represents the text for the title of the application, including color attributes in this case “lightBlueAccent”.

File: db_helper.dart:

```

32     return await openDatabase(path, version: 1, onCreate: _onCreate);
33   }
34
35   //creates all of the database tables.
36   Future<void> _onCreate(Database db, int version) async {
37     //generates the food_items table.
38     await db.execute('''
39     CREATE TABLE food_items (
40       id INTEGER PRIMARY KEY AUTOINCREMENT,
41       name TEXT NOT NULL,
42       cost REAL NOT NULL
43     )
44     ''');
45
46     //generates the orders table.
47     await db.execute('''
48     CREATE TABLE orders (
49       id INTEGER PRIMARY KEY AUTOINCREMENT,
50       date TEXT NOT NULL,
51       total_cost REAL NOT NULL,
52       items TEXT NOT NULL
53     )
54     ''');
55

```

The snippet above shows the creation of the “food_items” table and the “orders” table.

```

55
56   //inserts the 20 sample foods
57   await db.insert('food_items', {'name': 'Pizza', 'cost': 8.00});
58   await db.insert('food_items', {'name': 'Burger', 'cost': 5.00});
59   await db.insert('food_items', {'name': 'Poutine', 'cost': 7.00});
60   await db.insert('food_items', {'name': 'Salad', 'cost': 4.00});
61   await db.insert('food_items', {'name': 'Sushi', 'cost': 10.00});
62   await db.insert('food_items', {'name': 'Momos', 'cost': 6.00});
63   await db.insert('food_items', {'name': 'Sandwich', 'cost': 3.00});
64   await db.insert('food_items', {'name': 'Fries', 'cost': 2.50});
65   await db.insert('food_items', {'name': 'Ice Cream', 'cost': 2.00});
66   await db.insert('food_items', {'name': 'Steak', 'cost': 15.00});
67   await db.insert('food_items', {'name': 'Chicken Wings', 'cost': 8.50});
68   await db.insert('food_items', {'name': 'Ramen', 'cost': 9.00});
69   await db.insert('food_items', {'name': 'Bubble Tea', 'cost': 3.50});
70   await db.insert('food_items', {'name': 'Nachos', 'cost': 6.50});
71   await db.insert('food_items', {'name': 'Pancakes', 'cost': 4.50});
72   await db.insert('food_items', {'name': 'Shawarma', 'cost': 12.00});
73   await db.insert('food_items', {'name': 'Ice Cream Cake', 'cost': 14.00});
74   await db.insert('food_items', {'name': 'Pasta', 'cost': 5.00});
75   await db.insert('food_items', {'name': 'Strawberry Pie', 'cost': 3.00});
76   await db.insert('food_items', {'name': 'Grilled Cheese Sandwich', 'cost': 4.00});
77   }
78

```

Showing the hardcoding 20 sample food items into the database. Additional food items can be added right from the application.

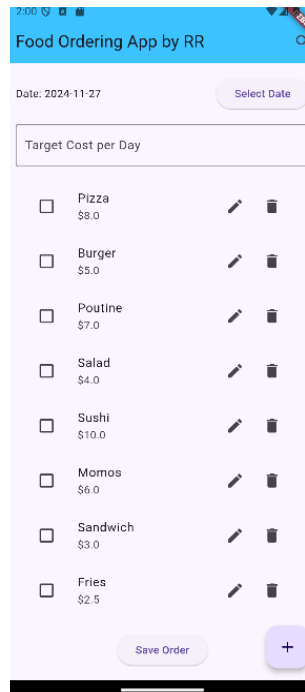
```

79 //inserts a food item into the database.
80 Future<int> insertFoodItem(Map<String, dynamic> row) async {
81   Database db = await database;
82   return await db.insert('food_items', row);
83 }
84
85 //queries all of the food items existing in the database.
86 Future<List<Map<String, dynamic>>> queryAllFoodItems() async {
87   Database db = await database;
88   return await db.query('food_items');
89 }
90
91 //updates a food item existing in the database.
92 Future<int> updateFoodItem(Map<String, dynamic> row) async {
93   Database db = await database;
94   int id = row['id'];
95   return await db.update('food_items', row, where: 'id = ?', whereArgs: [id]);
96 }
97
98 //deletes a food item from the existing database.
99 Future<int> deleteFoodItem(int id) async {
100   Database db = await database;
101   return await db.delete('food_items', where: 'id = ?', whereArgs: [id]);
102 }
103
104 //does the function of inserting an order into the database.
105 Future<int> insertOrder(Map<String, dynamic> order) async {
106   Database db = await database;

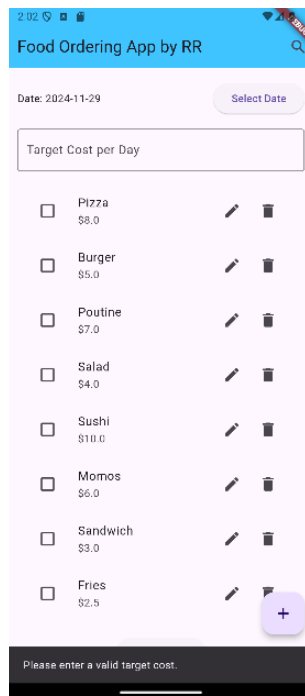
```

The snippet above portrays the codes that function to do the inserting of food items, query food items, update food items, etc. All of these work directly with the database.

Testing Screenshots of the Application:



Home screen of the application.



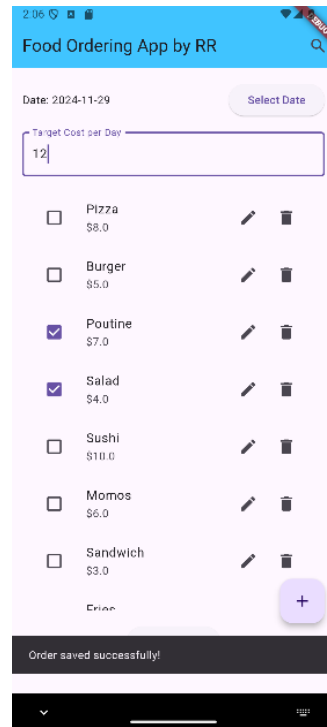
Error message because target cost was left blank.

The screenshot shows the 'Food Ordering App by RR' interface. At the top, the date is set to '2024-11-29'. Below this, there is a 'Target Cost per Day' input field containing the value '12'. A list of food items is displayed, each with a checkbox, the item name, and its price: Pizza (\$8.0), Burger (\$5.0), Poutine (\$7.0), Salad (\$4.0), Sushi (\$10.0), Momos (\$6.0), and Sandwich (\$3.0). All checkboxes are currently unchecked. At the bottom of the list is a purple button with a white '+' sign. A dark grey error message bar at the bottom of the screen reads: 'Please select at least one food item.'

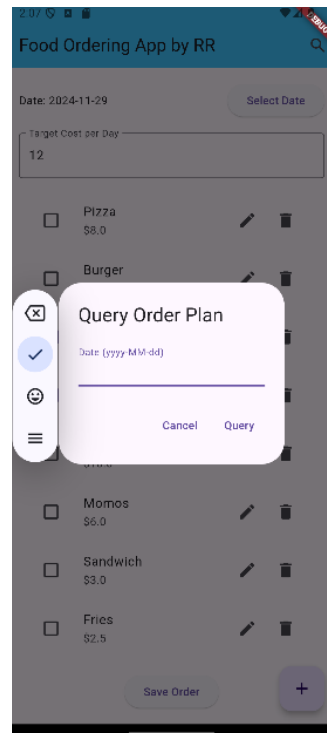
Error message because no food items were selected.

The screenshot shows the same 'Food Ordering App by RR' interface. The date is still '2024-11-29' and the 'Target Cost per Day' is '12'. In this view, the checkboxes for 'Pizza' and 'Poutine' are checked. The error message bar at the bottom now reads: 'Total cost exceeds the target cost.'

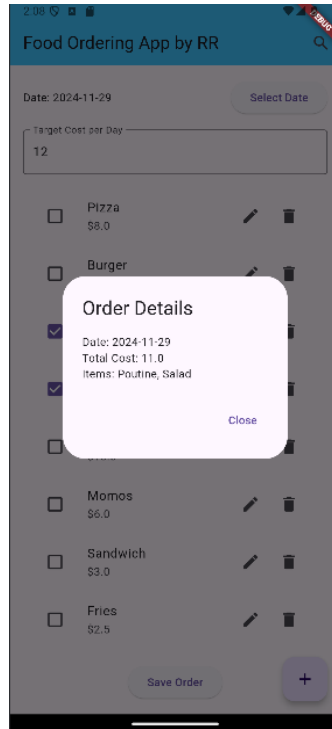
Error message because the total of Pizza (currently \$8.00) and Poutine (currently \$7.00) is greater than the “target cost per day” entered (\$12).



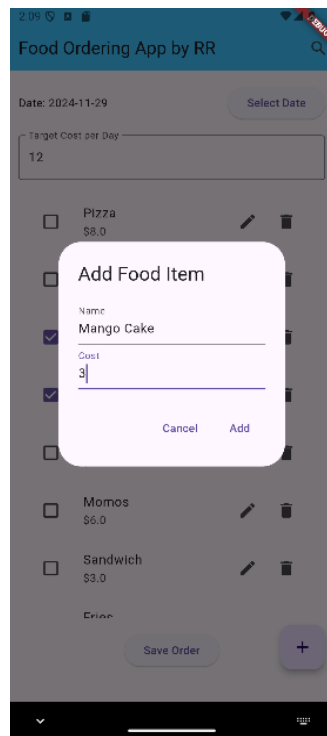
Order saved successfully for the date of 2024-11-29. The order contains Poutine (currently \$7.00) and Salad (currently \$4.00), totaling \$11.00.



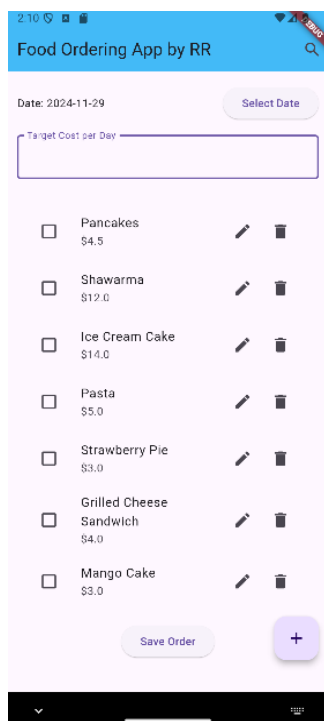
Querying the order recently created by clicking the search icon on the top right of the page and inputting the date 2024-11-29.



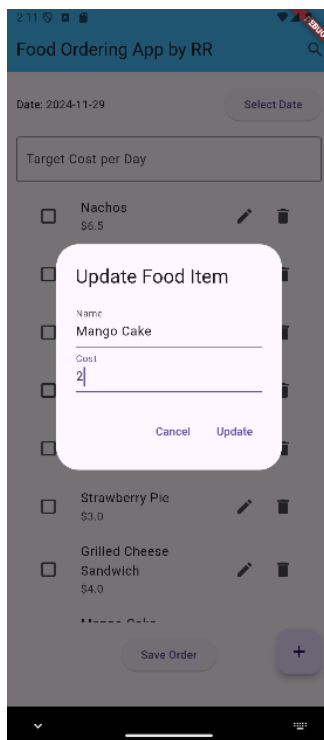
After making the query search, the correct order details are displayed.



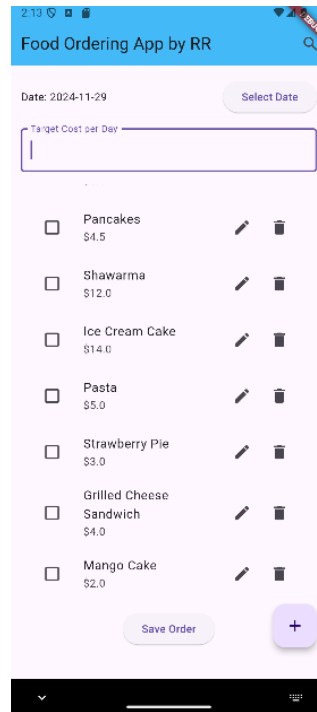
Can add a food item to the list by clicking on the “+” button on the bottom right and entering details like the one shown in the screenshot above.



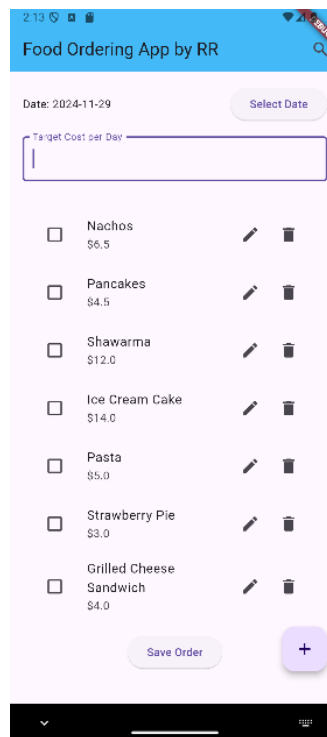
The newly-created food item is shown in the food list right away along with the cost.



Updating the cost of the newly-created food item to \$2 from the previous cost of \$3.



The previous cost was automatically updated to the new cost after the modification. Now Mango Cake has the updated cost of \$2.



The item can also be deleted by clicking on the trash icon. The previous image still contained the Mango Cake on the food item list, but after clicking the trash icon, it was deleted from the list (shown in the screenshot above).

Conclusion:

Overall, this assignment was very useful in being able to learn about mobile application development, but this time using Flutter. Setting up Flutter was one of the challenges at the beginning of the assignment, but with the help of some online resources and videos, the setup was finally done. Being able to build on the skills learned so far in the course and implementing it practically through this assignment is critical to test the knowledge of what has been learned so far and what can be improved. This “Food Ordering” was a great way to learn those skills and implement the adding/deleting/modifying and searching features on the app. All in all, the assignment was successful and all of the tasks were completed as expected with the use of the Android Studio platform and the Flutter/Dart plugins.