Mobile Application Development (SOFE 4640U)

Assignment 3: App Development Using Flutter - Food Ordering App

Name	**Student ID**
Rohan Radadiya	100704614

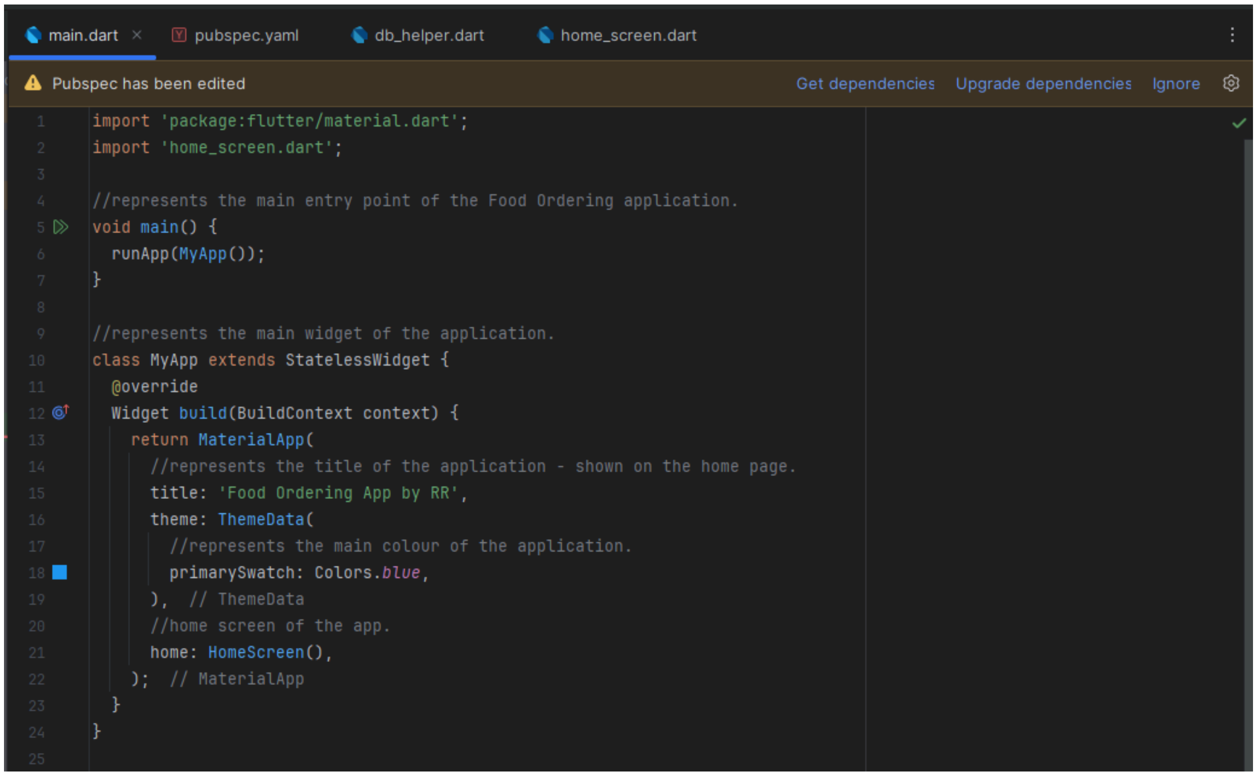
Due Date: November 27, 2024

GitHub Link:

https://github.com/rohanradadiya/MOBILE-APPLICATION-ASSIGNMENT-3

Code Snippets and Explanations:

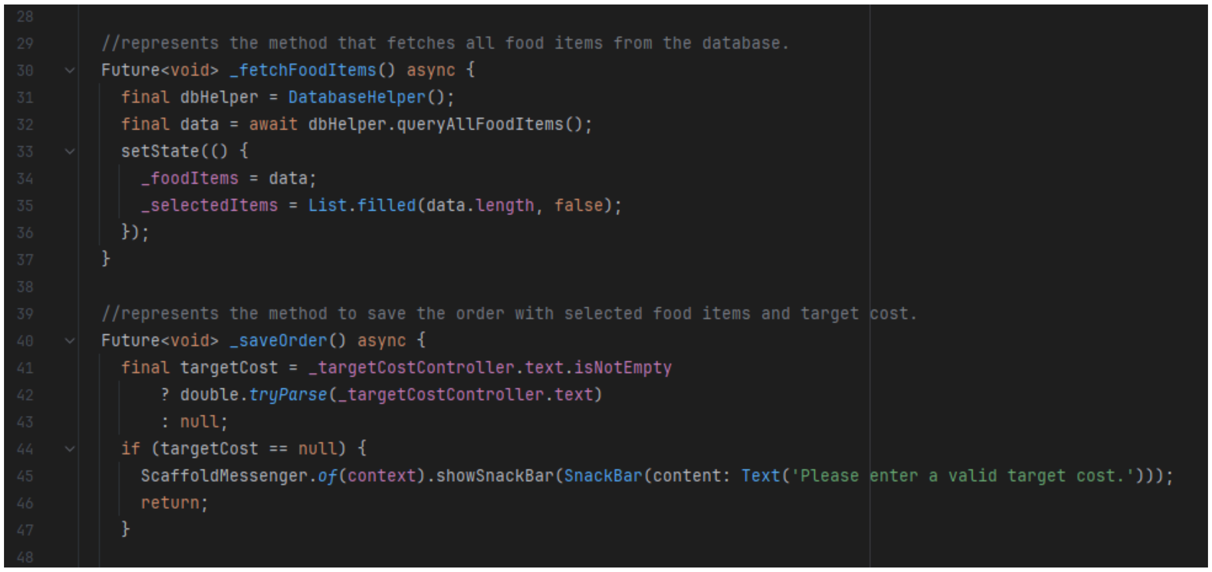
File: main.dart:



```dart
import 'package:flutter/material.dart';
import 'home_screen.dart';

//represents the main entry point of the Food Ordering application.
void main() {
  runApp(MyApp());
}

//represents the main widget of the application.
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      //represents the title of the application - shown on the home page.
      title: 'Food Ordering App by RR',
      theme: ThemeData(
        //represents the main colour of the application.
        primarySwatch: Colors.blue,
      ), // ThemeData
      //home screen of the app.
      home: HomeScreen(),
    ); // MaterialApp
  }
}
```

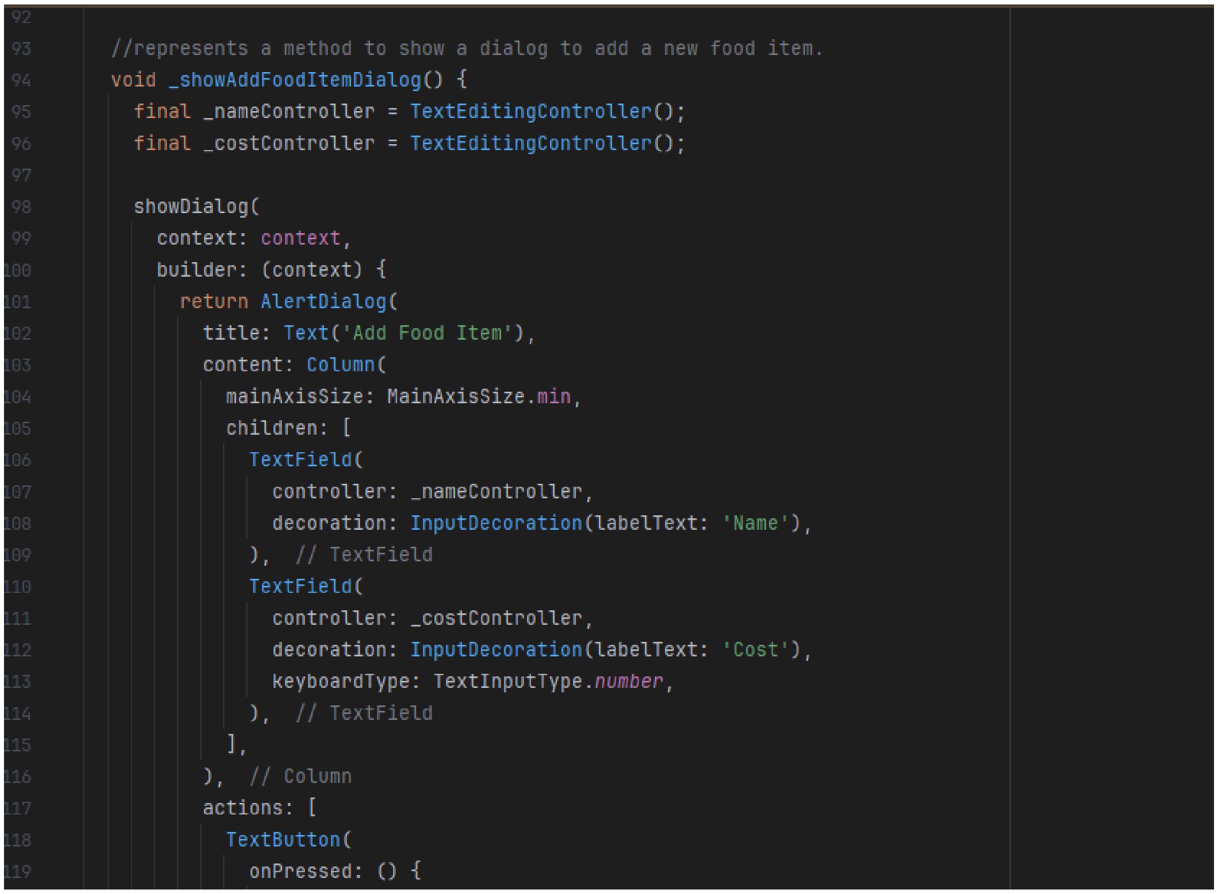
The screenshot above represents the “main.dart” file. It contains the main() function, which is the application's main entry point. It calls “run app” to start the application. The MyApp class is the application's main root widget, which sets up the application with a title and home screen.

<u>File: home_screen.dart:</u>



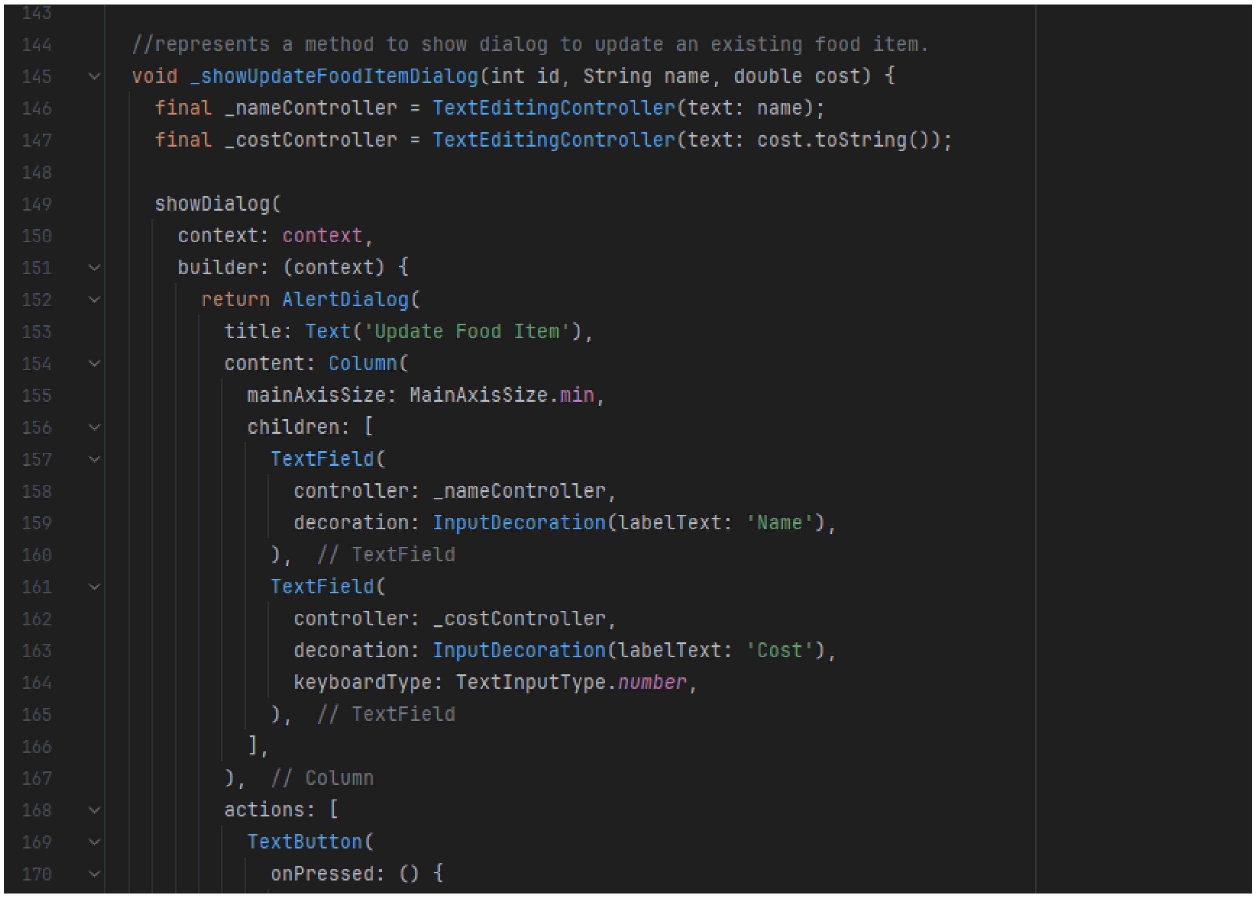
```
//represents the method that fetches all food items from the database.
Future<void> _fetchFoodItems() async {
  final dbHelper = DatabaseHelper();
  final data = await dbHelper.queryAllFoodItems();
  setState(() {
    _foodItems = data;
    _selectedItems = List.filled(data.length, false);
  });
}

//represents the method to save the order with selected food items and target cost.
Future<void> _saveOrder() async {
  final targetCost = _targetCostController.text.isNotEmpty
      ? double.tryParse(_targetCostController.text)
      : null;
  if (targetCost == null) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Please enter a valid target cost.')));
    return;
  }

```

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.



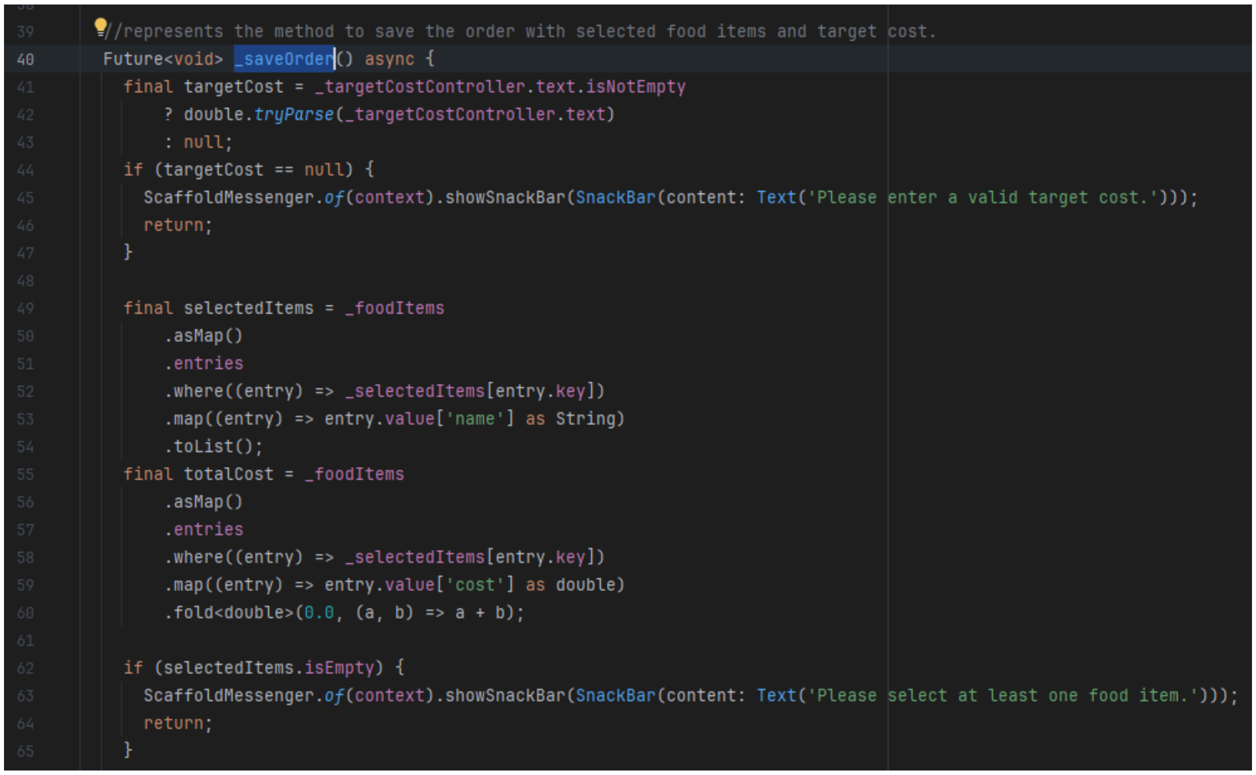
```

//represents a method to show a dialog to add a new food item.
void _showAddFoodItemDialog() {
  final _nameController = TextEditingController();
  final _costController = TextEditingController();

  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text('Add Food Item'),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            TextField(
              controller: _nameController,
              decoration: InputDecoration(labelText: 'Name'),
            ),  // TextField
            TextField(
              controller: _costController,
              decoration: InputDecoration(labelText: 'Cost'),
              keyboardType: TextInputType.number,
            ),  // TextField
          ],
        ),  // Column
        actions: [
          TextButton(
            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.



```
//represents a method to show dialog to update an existing food item.
void _showUpdateFoodItemDialog(int id, String name, double cost) {
  final _nameController = TextEditingController(text: name);
  final _costController = TextEditingController(text: cost.toString());

  showDialog(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text('Update Food Item'),
        content: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            TextField(
              controller: _nameController,
              decoration: InputDecoration(labelText: 'Name'),
            ),  // TextField
            TextField(
              controller: _costController,
              decoration: InputDecoration(labelText: 'Cost'),
              keyboardType: TextInputType.number,
            ),  // TextField
          ],
        ),  // Column
        actions: [
          TextButton(
            onPressed: () {
```

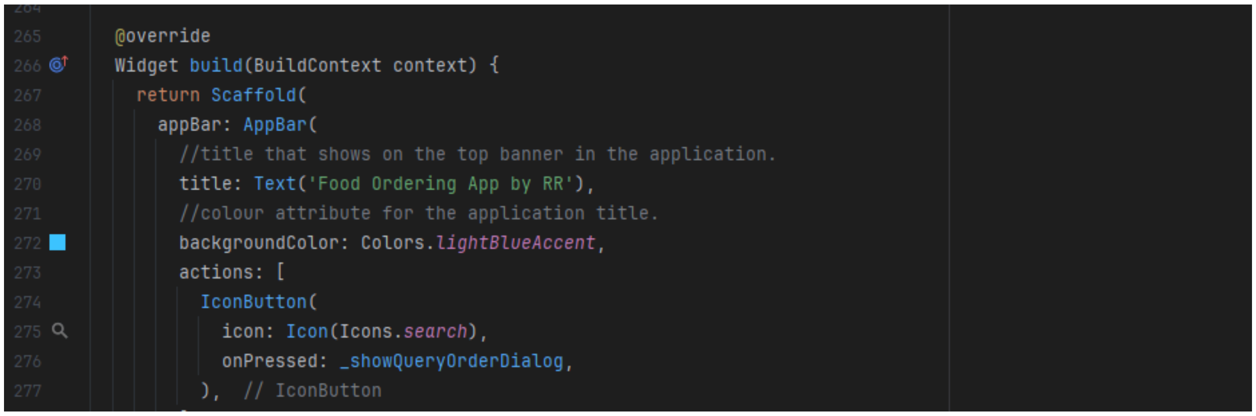
The _showUpdatedFoodItemDialog() method is created to show a dialog for updating an existing food item. The dialog is already filled with the name and cost of the food item. In addition, when the "Update" button is clicked, the food item is then updated in the database, and the list of food items is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



```
//represents the method to save the order with selected food items and target cost.
Future<void> _saveOrder() async {
  final targetCost = _targetCostController.text.isNotEmpty
      ? double.tryParse(_targetCostController.text)
      : null;
  if (targetCost == null) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Please enter a valid target cost.')));
    return;
  }

  final selectedItems = _foodItems
      .asMap()
      .entries
      .where((entry) => _selectedItems[entry.key])
      .map((entry) => entry.value['name'] as String)
      .toList();
  final totalCost = _foodItems
      .asMap()
      .entries
      .where((entry) => _selectedItems[entry.key])
      .map((entry) => entry.value['cost'] as double)
      .fold<double>(0.0, (a, b) => a + b);

  if (selectedItems.isEmpty) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Please select at least one food item.')));
    return;
  }
```

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.

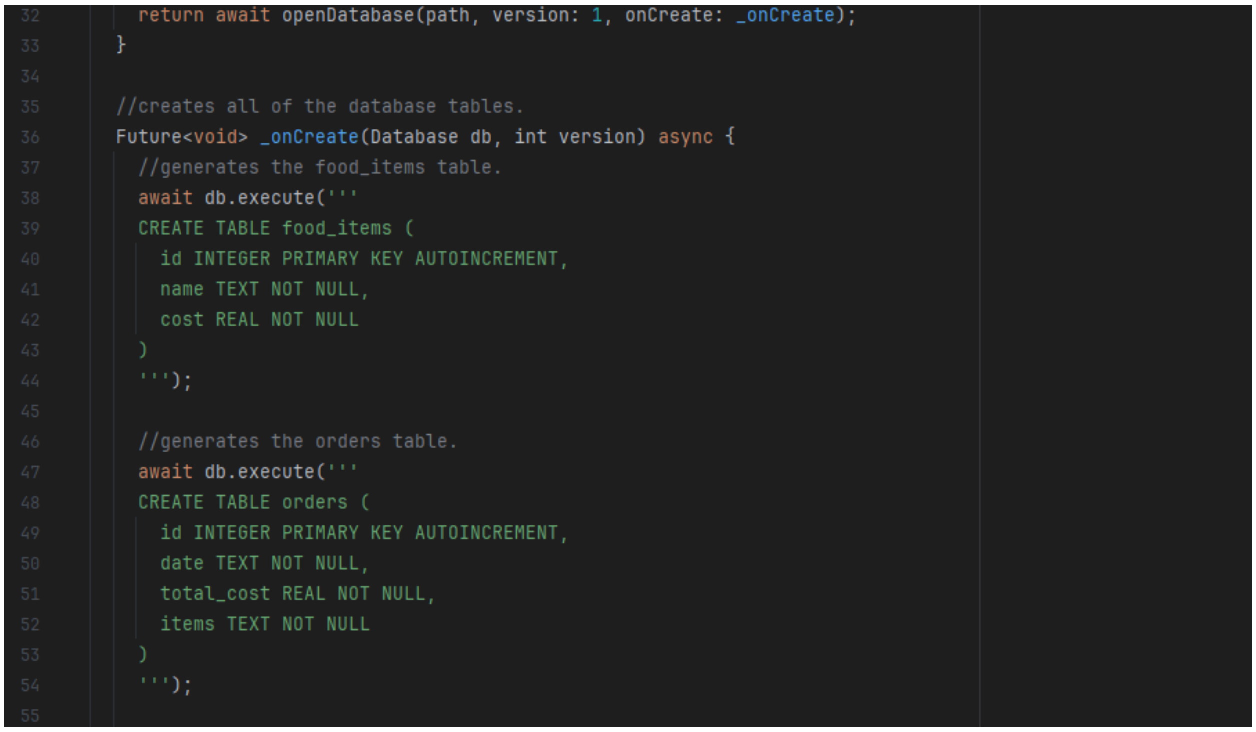
If the total cost of the food items selected tends to exceed the target cost, then an error message is displayed to the user that the cost of the food exceeds the target cost. After these steps, the method helps to either update an existing order or insert a new order into the database.



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      //title that shows on the top banner in the application.
      title: Text('Food Ordering App by RR'),
      //colour attribute for the application title.
      backgroundColor: Colors.lightBlueAccent,
      actions: [
        IconButton(
          icon: Icon(Icons.search),
          onPressed: _showQueryOrderDialog,
        ), // IconButton
```

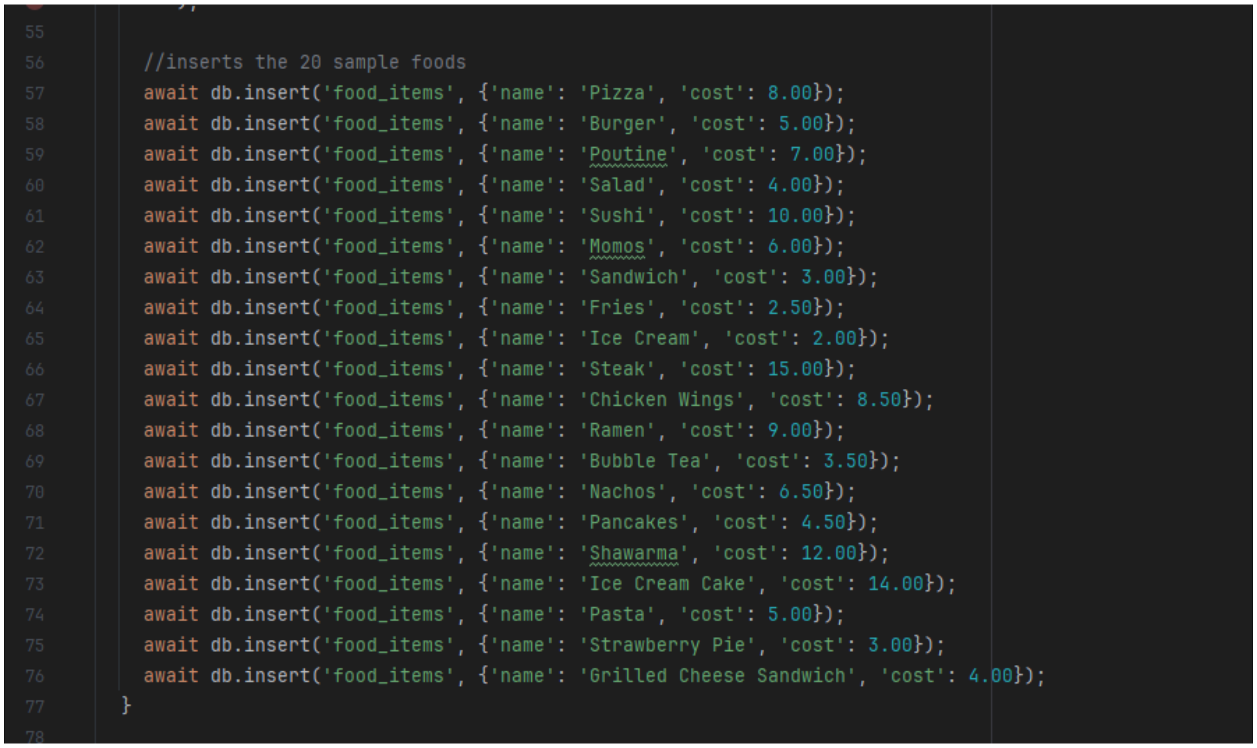
The snippet above represents the text for the title of the application, including color attributes in this case “lightBlueAccent”.

File: db_helper.dart:



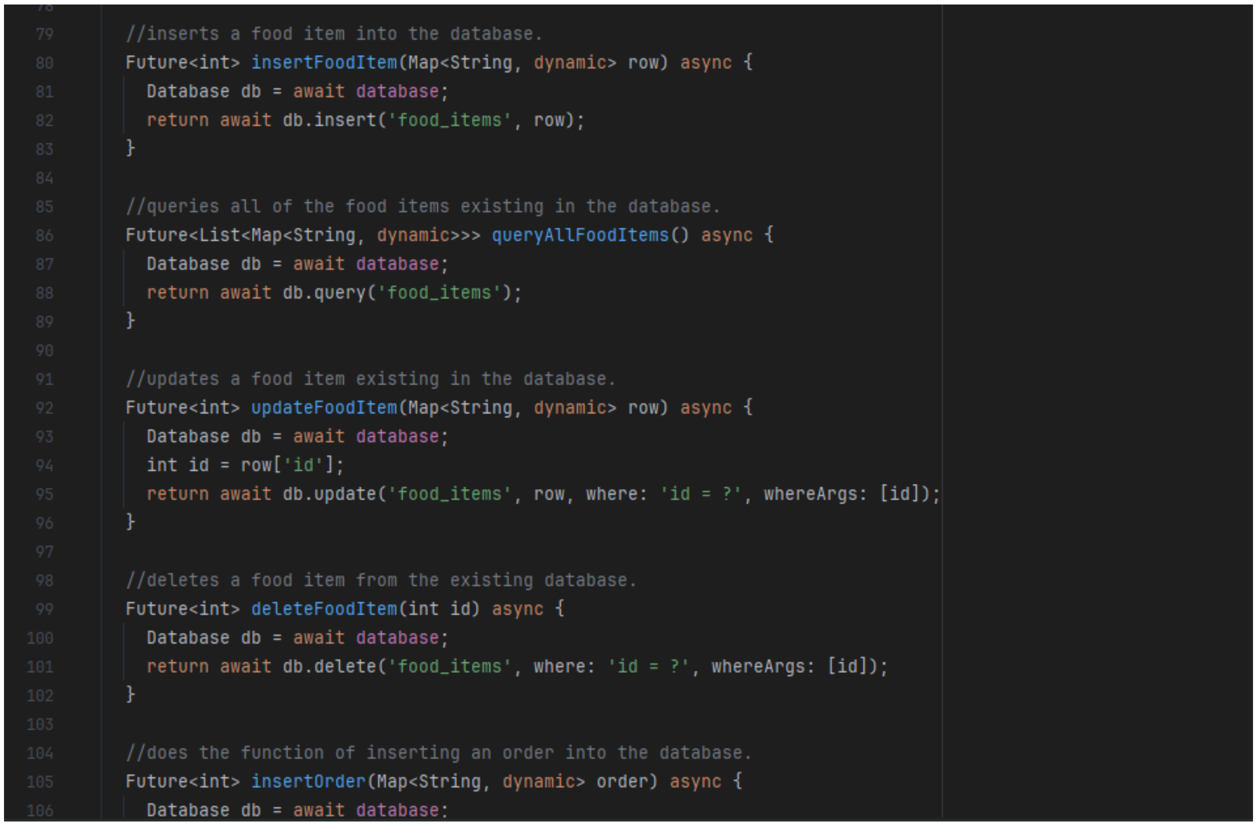
```dart
    return await openDatabase(path, version: 1, onCreate: _onCreate);
  }

  //creates all of the database tables.
  Future<void> _onCreate(Database db, int version) async {
    //generates the food_items table.
    await db.execute('''
    CREATE TABLE food_items (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      name TEXT NOT NULL,
      cost REAL NOT NULL
    )
    ''');

    //generates the orders table.
    await db.execute('''
    CREATE TABLE orders (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      date TEXT NOT NULL,
      total_cost REAL NOT NULL,
      items TEXT NOT NULL
    )
    ''');

```

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



```dart

    //inserts the 20 sample foods
    await db.insert('food_items', {'name': 'Pizza', 'cost': 8.00});
    await db.insert('food_items', {'name': 'Burger', 'cost': 5.00});
    await db.insert('food_items', {'name': 'Poutine', 'cost': 7.00});
    await db.insert('food_items', {'name': 'Salad', 'cost': 4.00});
    await db.insert('food_items', {'name': 'Sushi', 'cost': 10.00});
    await db.insert('food_items', {'name': 'Momos', 'cost': 6.00});
    await db.insert('food_items', {'name': 'Sandwich', 'cost': 3.00});
    await db.insert('food_items', {'name': 'Fries', 'cost': 2.50});
    await db.insert('food_items', {'name': 'Ice Cream', 'cost': 2.00});
    await db.insert('food_items', {'name': 'Steak', 'cost': 15.00});
    await db.insert('food_items', {'name': 'Chicken Wings', 'cost': 8.50});
    await db.insert('food_items', {'name': 'Ramen', 'cost': 9.00});
    await db.insert('food_items', {'name': 'Bubble Tea', 'cost': 3.50});
    await db.insert('food_items', {'name': 'Nachos', 'cost': 6.50});
    await db.insert('food_items', {'name': 'Pancakes', 'cost': 4.50});
    await db.insert('food_items', {'name': 'Shawarma', 'cost': 12.00});
    await db.insert('food_items', {'name': 'Ice Cream Cake', 'cost': 14.00});
    await db.insert('food_items', {'name': 'Pasta', 'cost': 5.00});
    await db.insert('food_items', {'name': 'Strawberry Pie', 'cost': 3.00});
    await db.insert('food_items', {'name': 'Grilled Cheese Sandwich', 'cost': 4.00});
  }

```

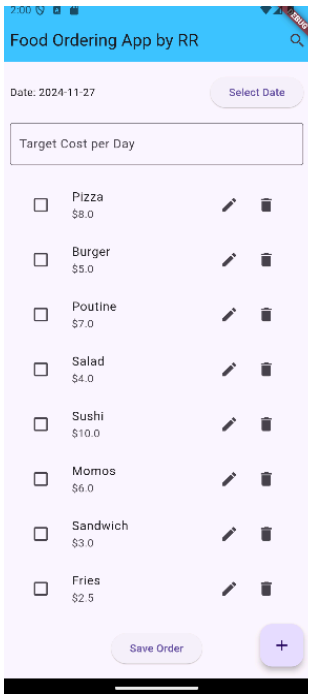
Showing the hardcoding 20 sample food items into the database. Additional food items can be added right from the application.



```
  //inserts a food item into the database.
  Future<int> insertFoodItem(Map<String, dynamic> row) async {
    Database db = await database;
    return await db.insert('food_items', row);
  }

  //queries all of the food items existing in the database.
  Future<List<Map<String, dynamic>>> queryAllFoodItems() async {
    Database db = await database;
    return await db.query('food_items');
  }

  //updates a food item existing in the database.
  Future<int> updateFoodItem(Map<String, dynamic> row) async {
    Database db = await database;
    int id = row['id'];
    return await db.update('food_items', row, where: 'id = ?', whereArgs: [id]);
  }

  //deletes a food item from the existing database.
  Future<int> deleteFoodItem(int id) async {
    Database db = await database;
    return await db.delete('food_items', where: 'id = ?', whereArgs: [id]);
  }

  //does the function of inserting an order into the database.
  Future<int> insertOrder(Map<String, dynamic> order) async {
    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:

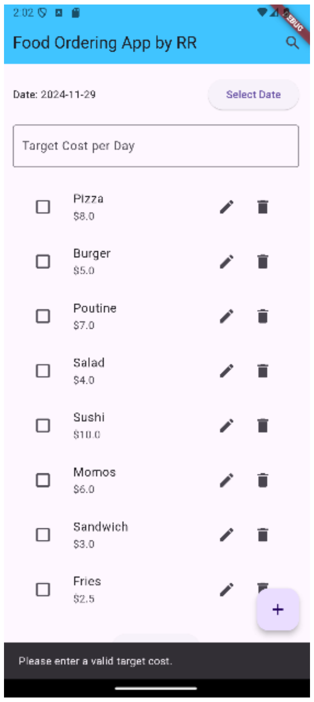

Food Ordering App by RR

Date: 2024-11-27

Target Cost per Day

Pizza	$8.0
Burger	$5.0
Poutine	$7.0
Salad	$4.0
Sushi	$10.0
Momos	$6.0
Sandwich	$3.0
Fries	$2.5

Home screen of the application.


Food Ordering App by RR

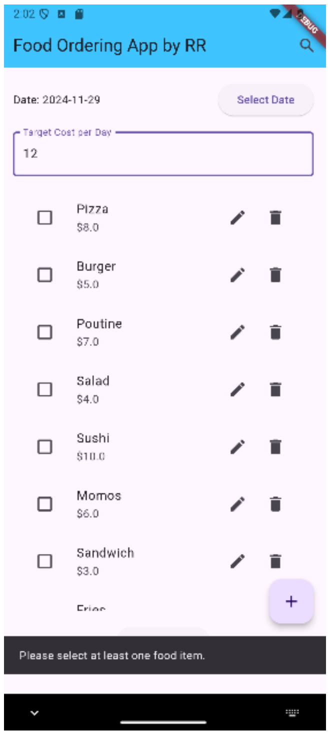
Date: 2024-11-29

Target Cost per Day

Pizza	$8.0
Burger	$5.0
Poutine	$7.0
Salad	$4.0
Sushi	$10.0
Momos	$6.0
Sandwich	$3.0
Fries	$2.5

Please enter a valid target cost.

Error message because target cost was left blank.



2:02

Food Ordering App by RR

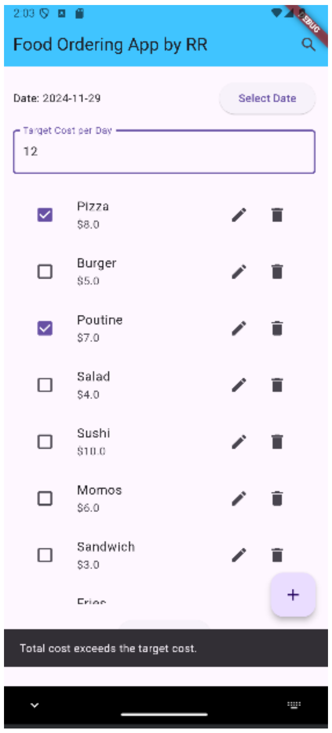
Date: 2024-11-29 | Select Date

Target Cost per Day: 12

☐	Pizza	$8.0
☐	Burger	$5.0
☐	Poutine	$7.0
☐	Salad	$4.0
☐	Sushi	$10.0
☐	Momos	$6.0
☐	Sandwich	$3.0

Please select at least one food item.

Error message because no food items were selected.



2:03

Food Ordering App by RR

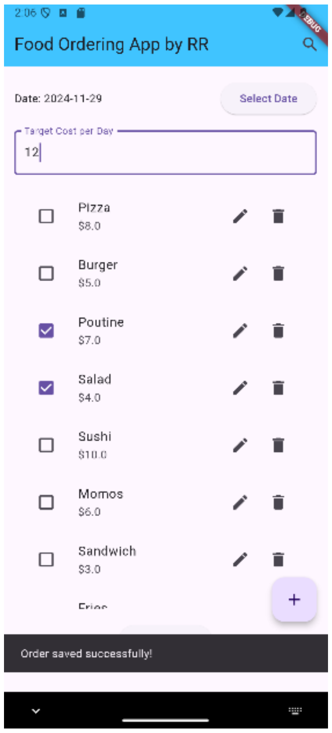
Date: 2024-11-29 | Select Date

Target Cost per Day: 12

✓	Pizza	$8.0
☐	Burger	$5.0
✓	Poutine	$7.0
☐	Salad	$4.0
☐	Sushi	$10.0
☐	Momos	$6.0
☐	Sandwich	$3.0

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).


Food Ordering App by RR

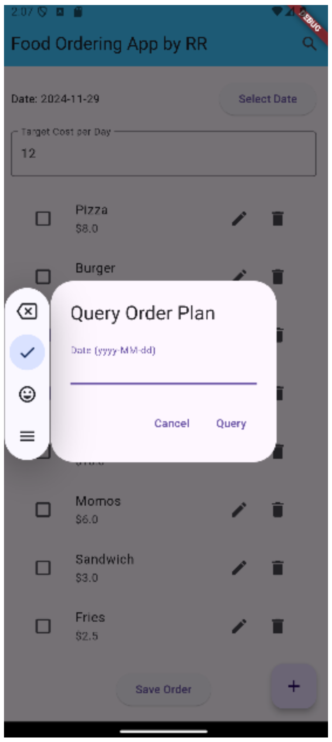
Date: 2024-11-29

Target Cost per Day: 12

☐	Pizza	$8.0
☐	Burger	$5.0
✓	Poutine	$7.0
✓	Salad	$4.0
☐	Sushi	$10.0
☐	Momos	$6.0
☐	Sandwich	$3.0
	Fries	

Order saved successfully!

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.


Food Ordering App by RR

Date: 2024-11-29

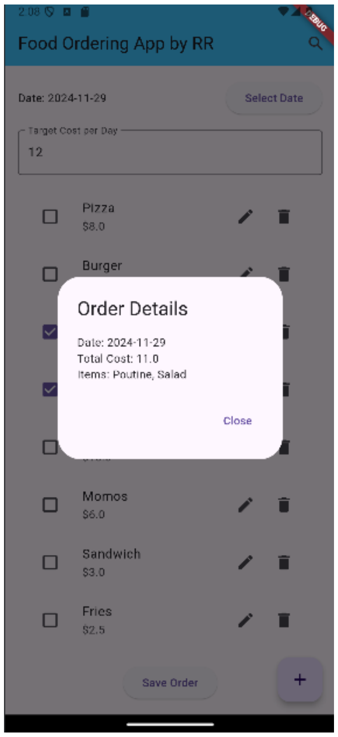
Target Cost per Day: 12

☐	Pizza	$8.0
☐	Burger	
☐	Momos	$6.0
☐	Sandwich	$3.0
☐	Fries	$2.5

Query Order Plan

Date (yyyy-MM-dd)

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

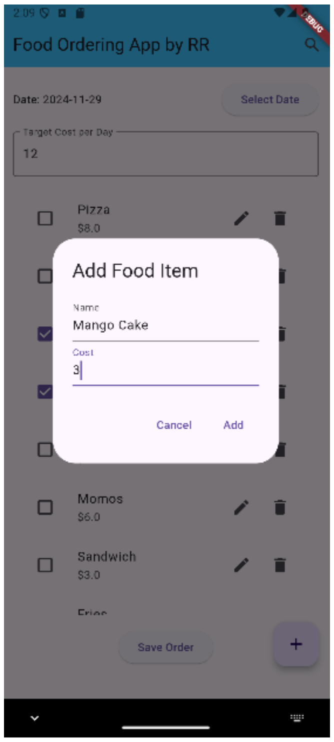


Date: 2024-11-29
12
$8.0
$6.0
$3.0
$2.5

Order Details

Date: 2024-11-29
Total Cost: 11.0
Items: Poutine, Salad

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

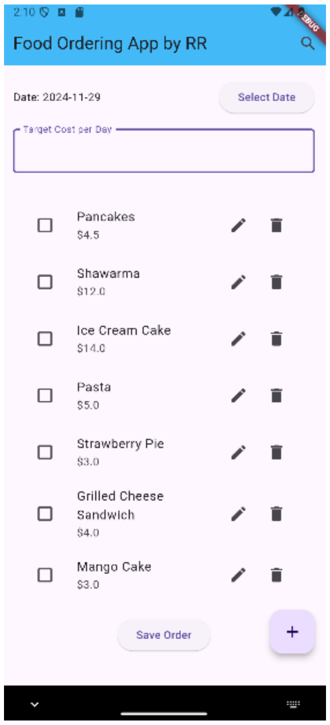


Date: 2024-11-29
12
$8.0
$6.0
$3.0

Add Food Item

Name
Mango Cake
Cost
3

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.



Food Ordering App by RR

Date: 2024-11-29

Target Cost per Day

Pancakes
$4.5

Shawarma
$12.0

Ice Cream Cake
$14.0

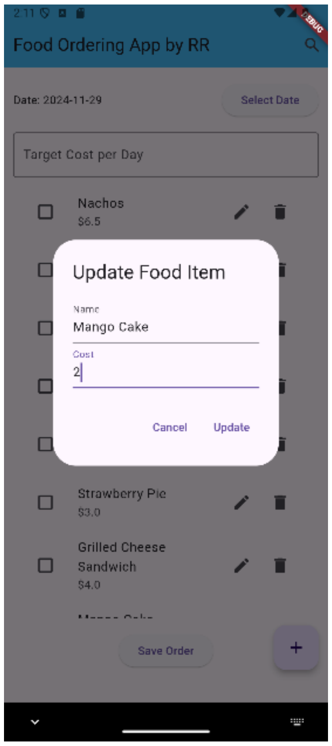
Pasta
$5.0

Strawberry Pie
$3.0

Grilled Cheese Sandwich
$4.0

Mango Cake
$3.0

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



Food Ordering App by RR

Date: 2024-11-29

Target Cost per Day

Nachos
$6.5

Update Food Item

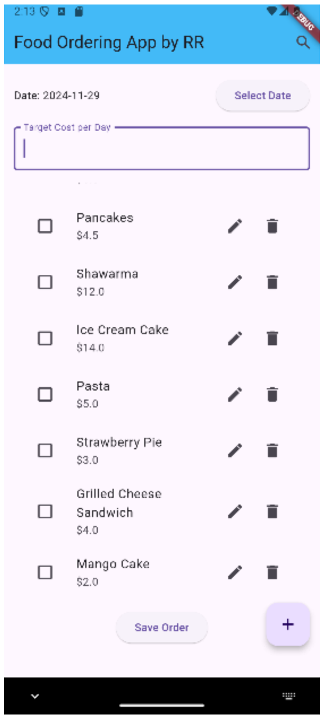
Name
Mango Cake

Cost
2

Strawberry Pie
$3.0

Grilled Cheese Sandwich
$4.0

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


2:13

Food Ordering App by RR

Date: 2024-11-29

Select Date

Target Cost per Day

Pancakes
$4.5

Shawarma
$12.0

Ice Cream Cake
$14.0

Pasta
$5.0

Strawberry Pie
$3.0

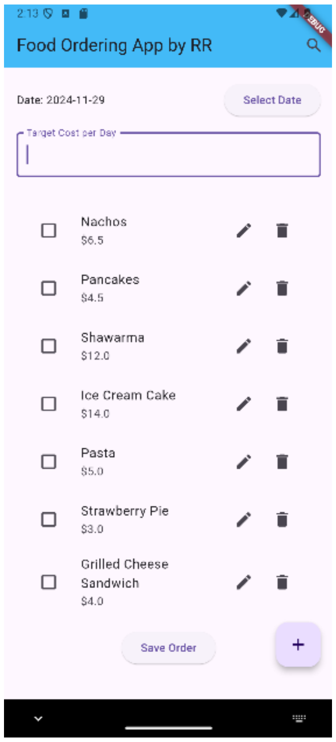
Grilled Cheese
Sandwich
$4.0

Mango Cake
$2.0

Save Order

+

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


2:13

Food Ordering App by RR

Date: 2024-11-29

Select Date

Target Cost per Day

Nachos
$6.5

Pancakes
$4.5

Shawarma
$12.0

Ice Cream Cake
$14.0

Pasta
$5.0

Strawberry Pie
$3.0

Grilled Cheese
Sandwich
$4.0

Save Order

+

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.