Mobile Application Developments - SOFE4640U

Assignment 3

CRN: 44434

Name: Alden O’Cain

Student ID: 100558599

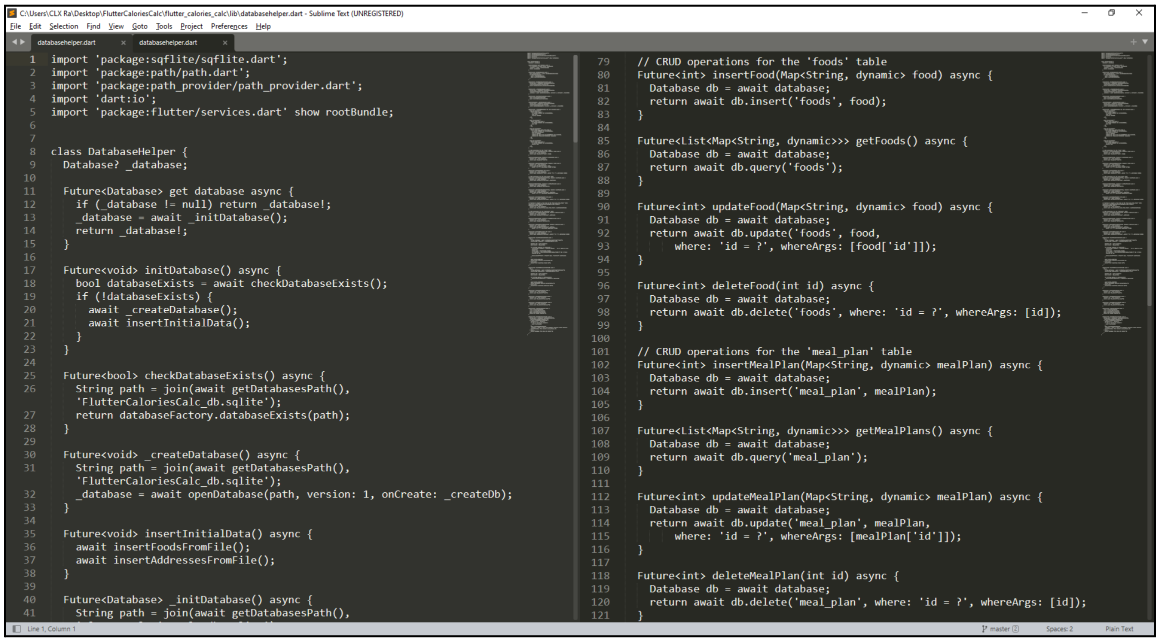
GitHub Repository

Introduction

This assignment uses Flutter and Dart programming languages in building the FlutterCaloriesCalculator mobile app in Android. The app allows users to manage daily calorie intake by selecting a target calorie count, and food items to create a meal plan, ensuring the daily caloric limit is not exceeded. The app also provides features for querying address-related data stored in the database and incorporates functionalities for adding, deleting, and updating entries.

Implementation Steps

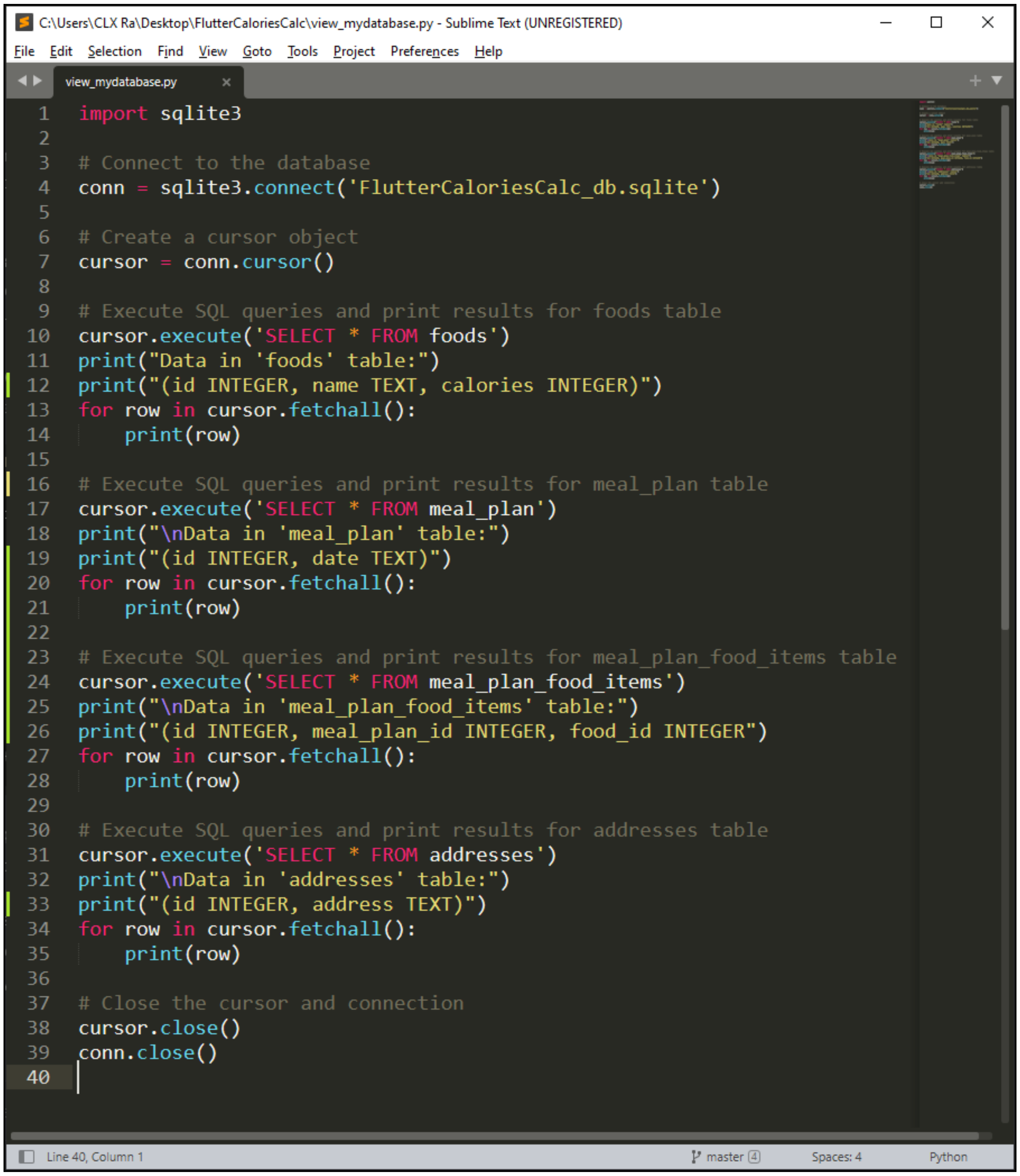
1. **Project Initialization:** Initialized a new Flutter project - *flutter create flutter_calories_calc*
2. **Database Operations Abstraction:** Created a 'databasehelper.dart' class to abstract and manage database operations using an SQLite database, ensuring functionalities such as CRUD (Create, Read, Update, Delete) operations were implemented effectively.


C:\Users\CLX Ra\Desktop\FlutterCaloriesCalc\flutter_calories_calc\lib\databasehelper.dart - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
databasehelper.dart
databasehelper.dart
Line 1, Column 1
master 2
Spaces: 2
Plain Text

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
import 'package:path_provider/path_provider.dart';
import 'dart:io';
import 'package:flutter/services.dart' show rootBundle;


class DatabaseHelper {
  Database? _database;

  Future<Database> get database async {
    if (_database != null) return _database!;
    _database = await _initDatabase();
    return _database!;
  }

  Future<void> initDatabase() async {
    bool databaseExists = await checkDatabaseExists();
    if (!databaseExists) {
      await _createDatabase();
      await insertInitialData();
    }
  }

  Future<bool> checkDatabaseExists() async {
    String path = join(await getDatabasesPath(),
    'FlutterCaloriesCalc_db.sqlite');
    return databaseFactory.databaseExists(path);
  }

  Future<void> _createDatabase() async {
    String path = join(await getDatabasesPath(),
    'FlutterCaloriesCalc_db.sqlite');
    _database = await openDatabase(path, version: 1, onCreate: _createDb);
  }

  Future<void> insertInitialData() async {
    await insertFoodsFromFile();
    await insertAddressesFromFile();
  }

  Future<Database> _initDatabase() async {
    String path = join(await getDatabasesPath(),
```

```dart
  // CRUD operations for the 'foods' table
  Future<int> insertFood(Map<String, dynamic> food) async {
    Database db = await database;
    return await db.insert('foods', food);
  }

  Future<List<Map<String, dynamic>>> getFoods() async {
    Database db = await database;
    return await db.query('foods');
  }

  Future<int> updateFood(Map<String, dynamic> food) async {
    Database db = await database;
    return await db.update('foods', food,
        where: 'id = ?', whereArgs: [food['id']]);
  }

  Future<int> deleteFood(int id) async {
    Database db = await database;
    return await db.delete('foods', where: 'id = ?', whereArgs: [id]);
  }

  // CRUD operations for the 'meal_plan' table
  Future<int> insertMealPlan(Map<String, dynamic> mealPlan) async {
    Database db = await database;
    return await db.insert('meal_plan', mealPlan);
  }

  Future<List<Map<String, dynamic>>> getMealPlans() async {
    Database db = await database;
    return await db.query('meal_plan');
  }

  Future<int> updateMealPlan(Map<String, dynamic> mealPlan) async {
    Database db = await database;
    return await db.update('meal_plan', mealPlan,
        where: 'id = ?', whereArgs: [mealPlan['id']]);
  }

  Future<int> deleteMealPlan(int id) async {
    Database db = await database;
    return await db.delete('meal_plan', where: 'id = ?', whereArgs: [id]);
  }
```

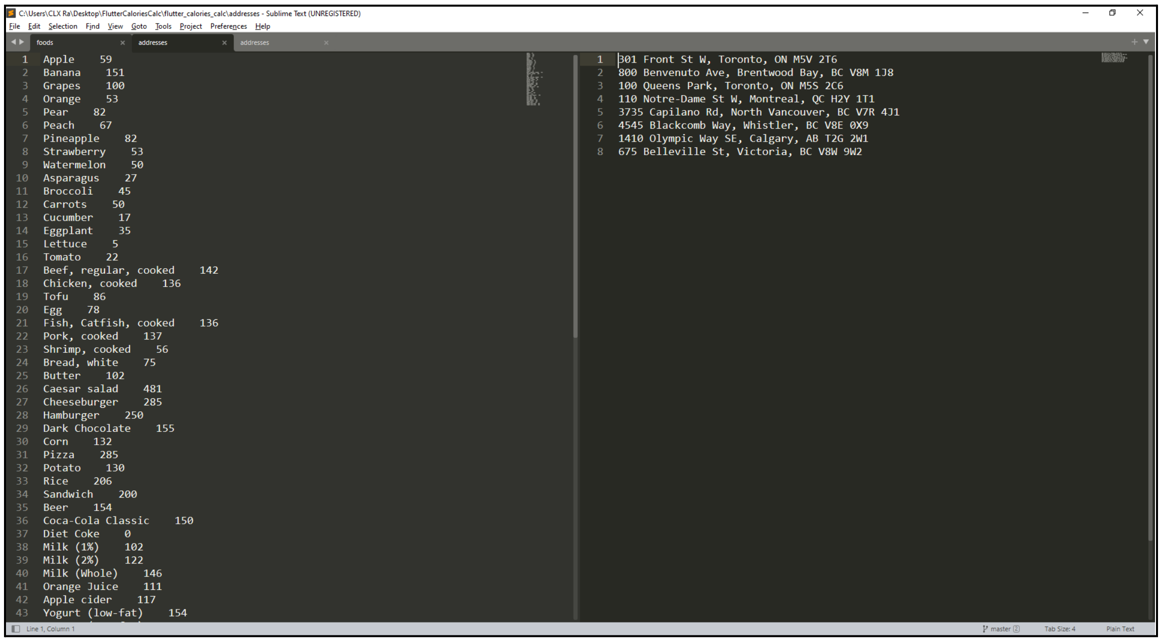
3. **Database Testing:** Tested the database operations by performing simple operations and checking the effects on the database. The database was exported from the emulated device and explored using a Python script to confirm the internal state of the database after running tests.



C:\Users\CLX Ra\Desktop\FlutterCaloriesCalc\view_mydatabase.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
view_mydatabase.py
Line 40, Column 1
master 4
Spaces: 4
Python

```python
import sqlite3

# Connect to the database
conn = sqlite3.connect('FlutterCaloriesCalc_db.sqlite')

# Create a cursor object
cursor = conn.cursor()

# Execute SQL queries and print results for foods table
cursor.execute('SELECT * FROM foods')
print("Data in 'foods' table:")
print("(id INTEGER, name TEXT, calories INTEGER)")
for row in cursor.fetchall():
    print(row)

# Execute SQL queries and print results for meal_plan table
cursor.execute('SELECT * FROM meal_plan')
print("\nData in 'meal_plan' table:")
print("(id INTEGER, date TEXT)")
for row in cursor.fetchall():
    print(row)

# Execute SQL queries and print results for meal_plan_food_items table
cursor.execute('SELECT * FROM meal_plan_food_items')
print("\nData in 'meal_plan_food_items' table:")
print("(id INTEGER, meal_plan_id INTEGER, food_id INTEGER")
for row in cursor.fetchall():
    print(row)

# Execute SQL queries and print results for addresses table
cursor.execute('SELECT * FROM addresses')
print("\nData in 'addresses' table:")
print("(id INTEGER, address TEXT)")
for row in cursor.fetchall():
    print(row)

# Close the cursor and connection
cursor.close()
conn.close()
```

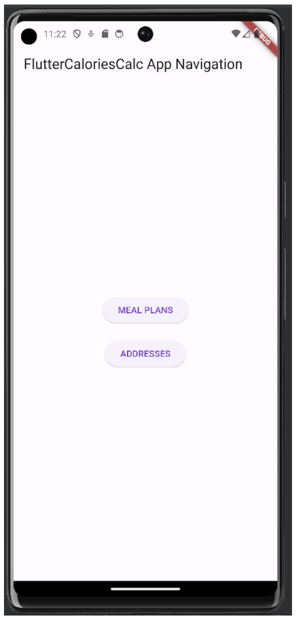
4. **Data Loading from Files:** Wrote functions to load address and food pairs (names and calorie counts) from text files located in the project's root directory, allowing for initial data integration into the app when the databases are created.


C:\Users\CLX Ra\Desktop\FlutterCaloriesCalc\flutter_calories_calc\addresses - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
foods
addresses
addresses
Line 1, Column 1
master
Tab Size: 4
Plain Text

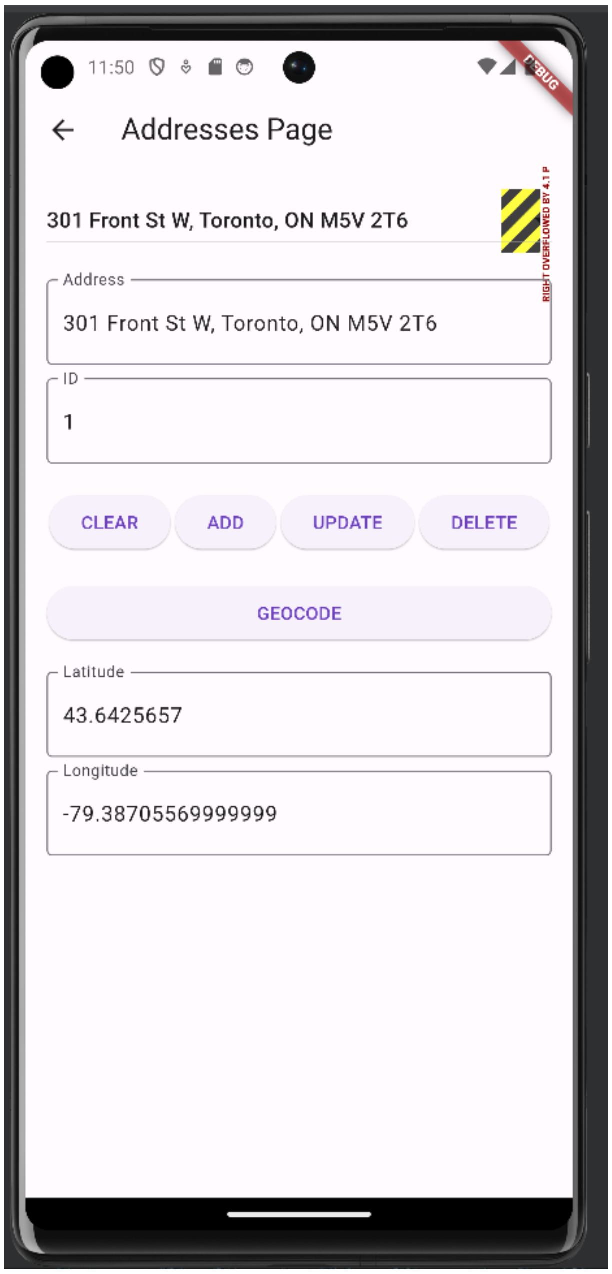
```
Apple    59
Banana    151
Grapes    100
Orange    53
Pear    82
Peach    67
Pineapple    82
Strawberry    53
Watermelon    50
Asparagus    27
Broccoli    45
Carrots    50
Cucumber    17
Eggplant    35
Lettuce    5
Tomato    22
Beef, regular, cooked    142
Chicken, cooked    136
Tofu    86
Egg    78
Fish, Catfish, cooked    136
Pork, cooked    137
Shrimp, cooked    56
Bread, white    75
Butter    102
Caesar salad    481
Cheeseburger    285
Hamburger    250
Dark Chocolate    155
Corn    132
Pizza    285
Potato    130
Rice    206
Sandwich    200
Beer    154
Coca-Cola Classic    150
Diet Coke    0
Milk (1%)    102
Milk (2%)    122
Milk (Whole)    146
Orange Juice    111
Apple cider    117
Yogurt (low-fat)    154
```

```
301 Front St W, Toronto, ON M5V 2T6
800 Benvenuto Ave, Brentwood Bay, BC V8M 1J8
100 Queens Park, Toronto, ON M5S 2C6
110 Notre-Dame St W, Montreal, QC H2Y 1T1
3735 Capilano Rd, North Vancouver, BC V7R 4J1
4545 Blackcomb Way, Whistler, BC V8E 0X9
1410 Olympic Way SE, Calgary, AB T2G 2W1
675 Belleville St, Victoria, BC V8W 9W2
```

5. **UI Development for Address Handling:** Modified the default home page to navigate to a new page dedicated to handling reverse geocoding. Implemented functionalities using the 'addresses.dart' class, enabling users to select addresses from a dropdown menu populated with database-loaded addresses from files.



6. **CRUD Operations on Database:** Added buttons for performing CRUD operations on the database based on the values entered in text fields.



301 Front St W, Toronto, ON M5V 2T6

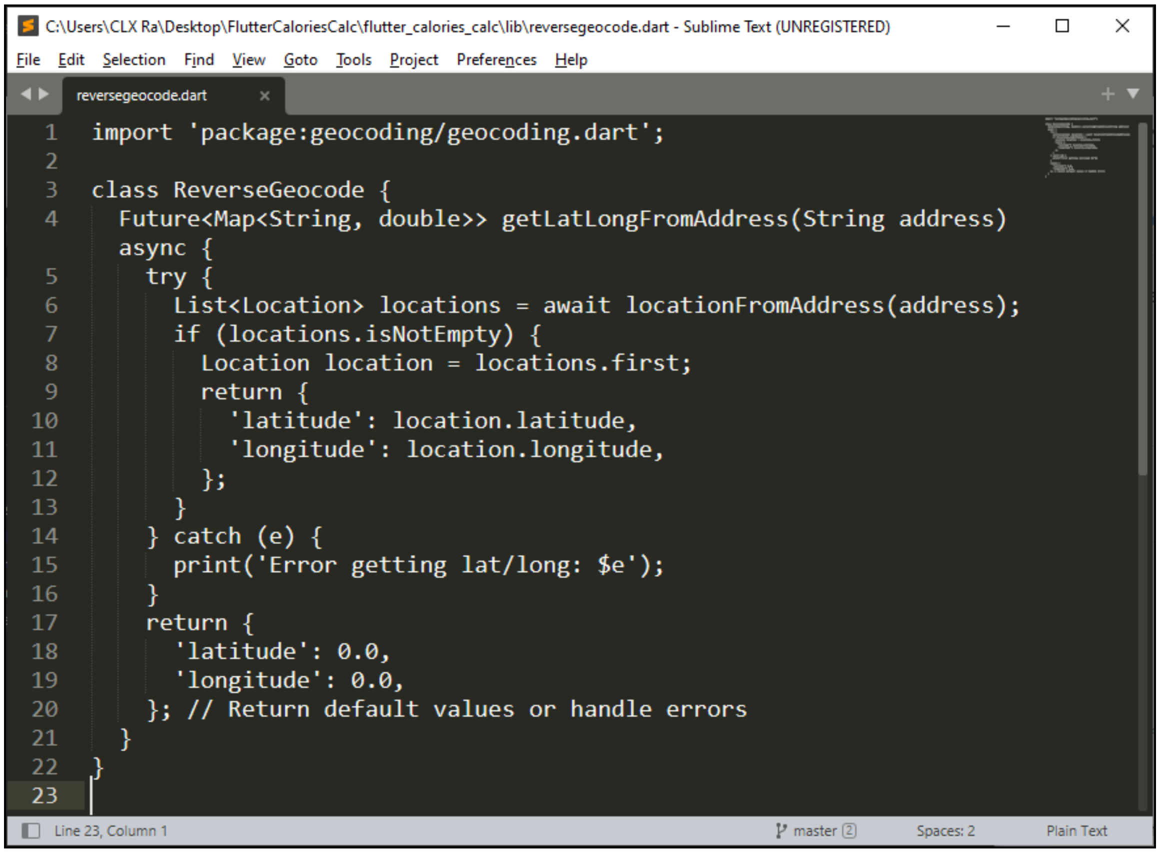
Address
301 Front St W, Toronto, ON M5V 2T6

ID
1

Latitude
43.6425657

Longitude
-79.38705569999999

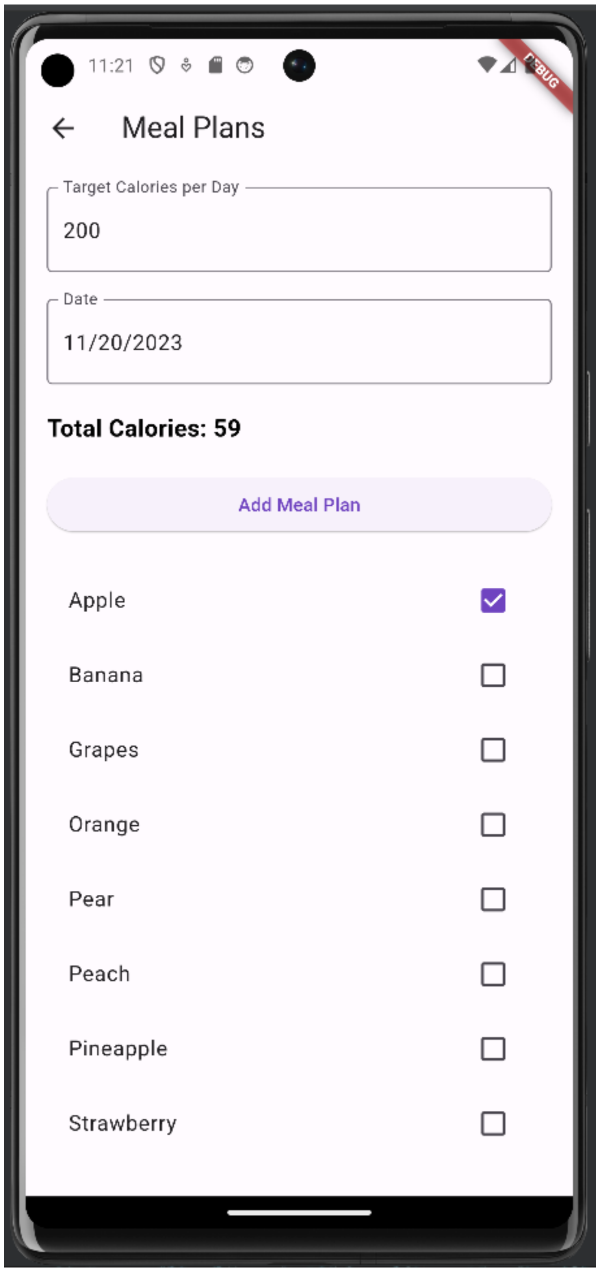
7. **Reverse Geocoding Functionality:** Created the 'reversegeocode.dart' class to handle reverse geocoding functionalities. Tested the accuracy using the campus address for Ontario Tech U (‘2000 Simcoe St N, Oshawa, ON L1G 0C5, Canada’), and validated the returned latitude and longitude using Google Maps.


C:\Users\CLX Ra\Desktop\FlutterCaloriesCalc\flutter_calories_calc\lib\reversegeocode.dart - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
reversegeocode.dart

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

class ReverseGeocode {
  Future<Map<String, double>> getLatLongFromAddress(String address)
  async {
    try {
      List<Location> locations = await locationFromAddress(address);
      if (locations.isNotEmpty) {
        Location location = locations.first;
        return {
          'latitude': location.latitude,
          'longitude': location.longitude,
        };
      }
    } catch (e) {
      print('Error getting lat/long: $e');
    }
    return {
      'latitude': 0.0,
      'longitude': 0.0,
    }; // Return default values or handle errors
  }
}
```

8. **Enhancement of Address Page:** Expanded the 'addresses.dart' class to include text fields for latitude and longitude and implemented a button calling the geocoding methods in the 'reversegeocode.dart' class. Verified the consistency of geocoding results when a new address was selected. See previous image in #6 ‘CRUD Operations on Database’.

9. **Meal Planning Page Development:** Designed a new page for meal plans featuring text fields for setting daily calories, date selection with auto-population, and a display box for current calories. Added a checkbox list for food items from the database and developed logic to update the total calories displayed when users interacted with the checkbox list. Incorporated text color changes to indicate exceeding the set calorie limit.

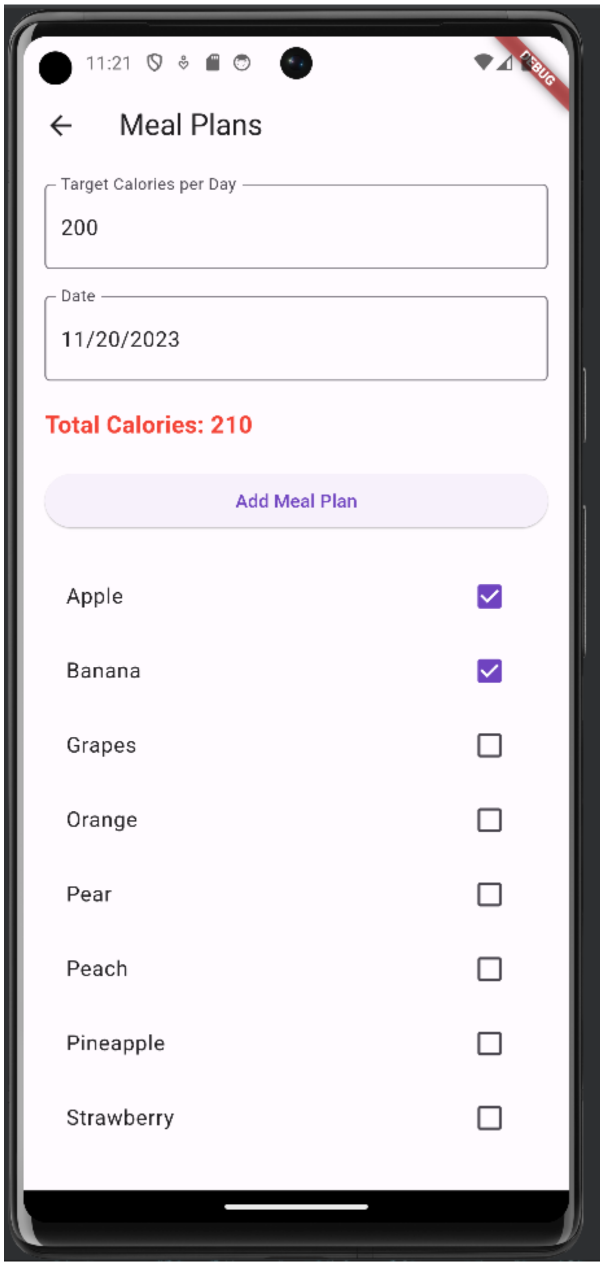


Target Calories per Day: 200

Date: 11/20/2023

Total Calories: 59

- Apple ✓
- Banana ☐
- Grapes ☐
- Orange ☐
- Pear ☐
- Peach ☐
- Pineapple ☐
- Strawberry ☐



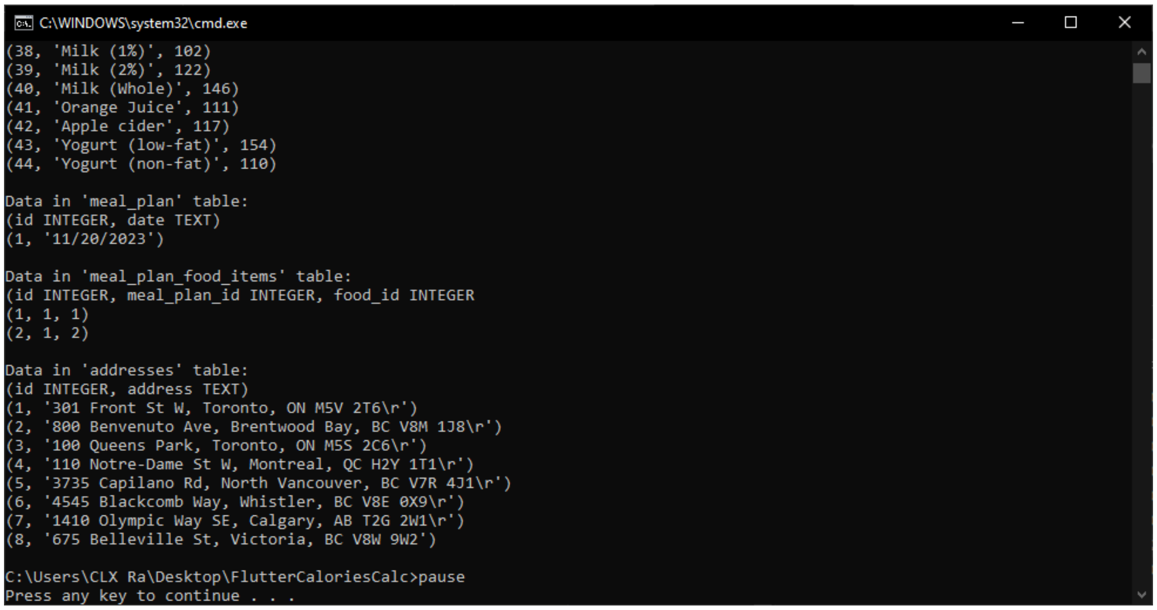
Target Calories per Day: 200

Date: 11/20/2023

Total Calories: 210

- Apple ✓
- Banana ✓
- Grapes ☐
- Orange ☐
- Pear ☐
- Peach ☐
- Pineapple ☐
- Strawberry ☐

10. **Saving Meal Plans to Database:** Implemented a button and functionality to save meal plans into the database based on the selected date as well as food items from the checkbox list, finally confirmed the accurate storage of data in the database.



```
(38, 'Milk (1%)', 102)
(39, 'Milk (2%)', 122)
(40, 'Milk (Whole)', 146)
(41, 'Orange Juice', 111)
(42, 'Apple cider', 117)
(43, 'Yogurt (low-fat)', 154)
(44, 'Yogurt (non-fat)', 110)

Data in 'meal_plan' table:
(id INTEGER, date TEXT)
(1, '11/20/2023')

Data in 'meal_plan_food_items' table:
(id INTEGER, meal_plan_id INTEGER, food_id INTEGER
(1, 1, 1)
(2, 1, 2)

Data in 'addresses' table:
(id INTEGER, address TEXT)
(1, '301 Front St W, Toronto, ON M5V 2T6\r')
(2, '800 Benvenuto Ave, Brentwood Bay, BC V8M 1J8\r')
(3, '100 Queens Park, Toronto, ON M5S 2C6\r')
(4, '110 Notre-Dame St W, Montreal, QC H2Y 1T1\r')
(5, '3735 Capilano Rd, North Vancouver, BC V7R 4J1\r')
(6, '4545 Blackcomb Way, Whistler, BC V8E 0X9\r')
(7, '1410 Olympic Way SE, Calgary, AB T2G 2W1\r')
(8, '675 Belleville St, Victoria, BC V8W 9W2')

C:\Users\CLX Ra\Desktop\FlutterCaloriesCalc>pause
Press any key to continue . . .
```

Conclusion

The FlutterCaloriesCalculator app uses Flutter and Dart to manage food items, set daily calorie limits, and incorporates database-related features. The user interface allows for selecting food items and creating meal plans. Additionally there is a reverse geocoding feature that allows users to enter an address and find the corresponding latitude and longitude and manage a directory of addresses that are stored in a SQLite database.