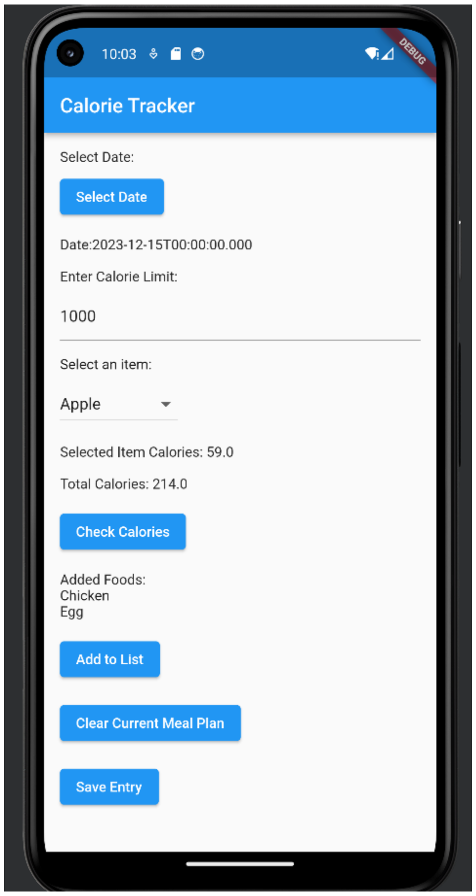
Create a FlutterCaloriesCalculator app for the following instructions:

Github: https://github.com/sarahwedge/assignment3

1. Create a database and store at least 20 preferred food items and calories pairs

To accomplish this, I created a text file with 20 food and calorie pairs each separated by a new line. Then I created a database helper class called FoodDatabaseHandler that creates a database storing each food with a corresponding id and its calorie amount. This class reads from the text file and populates the database with the food information. In my main class, I created the interface for users to enter in the date, select a food item, check the calorie count, add foods to a meal plan, and save them to the database. The main class communicates with the FoodDatabaseHandler class to retrieve a list of all the food items stored in the database. This list of food objects is then used to populate the dropdown menu.



Select Date:

Date:2023-12-15T00:00:00.000

Enter Calorie Limit:

1000

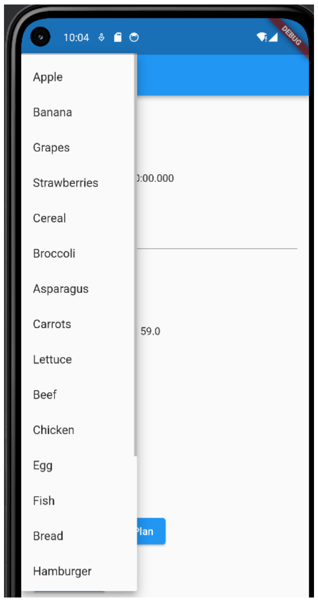
Select an item:

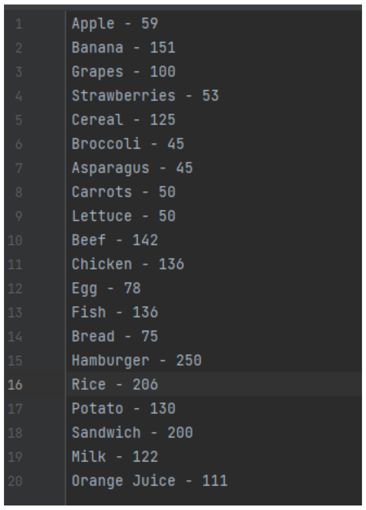
Apple

Selected Item Calories: 59.0

Total Calories: 214.0

Added Foods:
Chicken
Egg

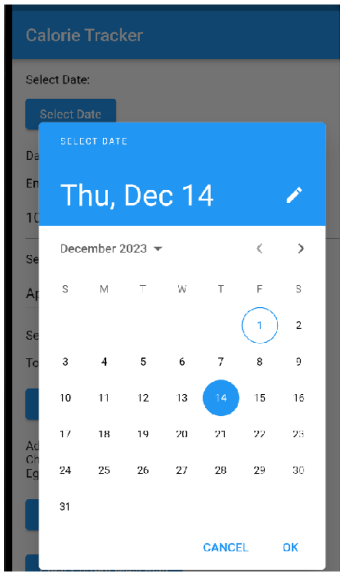


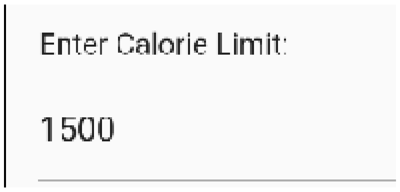


```
Apple - 59
Banana - 151
Grapes - 100
Strawberries - 53
Cereal - 125
Broccoli - 45
Asparagus - 45
Carrots - 50
Lettuce - 50
Beef - 142
Chicken - 136
Egg - 78
Fish - 136
Bread - 75
Hamburger - 250
Rice - 206
Potato - 130
Sandwich - 200
Milk - 122
Orange Juice - 111
```

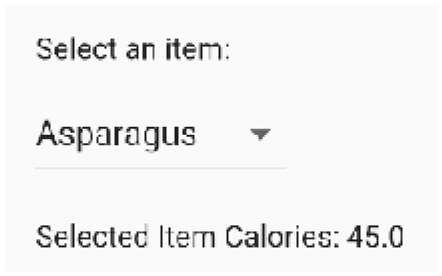
2. Users can select a “target calories per day”, “date”, and select the food item from the list to not exceed the target calories.

Users can select a date by pressing the “Select Date” button and enter their target calories using a TextFormField. Each food item selected on the dropdown will be shown with its calorie amount upon being selected. The user can add the food to their meal plan by clicking the “Add to List” button. The total sum of calories for all foods in the meal plan will be shown to the user under “Total Calories”. The user can also check if the current contents of their meal plan are over the target calorie limit by pressing the “Check Calories” button. Users can clear the contents of their meal plan or save the meal plan to the database by selecting the “Save Entry” button.



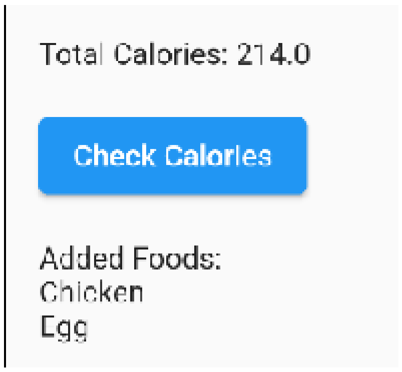

Enter Calorie Limit:

1500

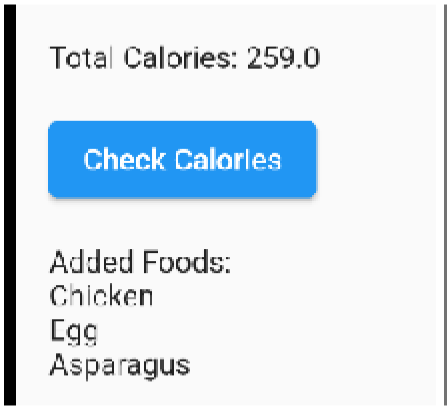

Select an item:

Asparagus

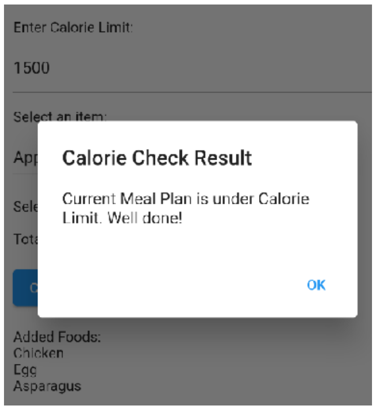
Selected Item Calories: 45.0


Total Calories: 214.0

Added Foods:
Chicken
Egg


Total Calories: 259.0

Added Foods:
Chicken
Egg
Asparagus


Enter Calorie Limit:
1500
Select an item:

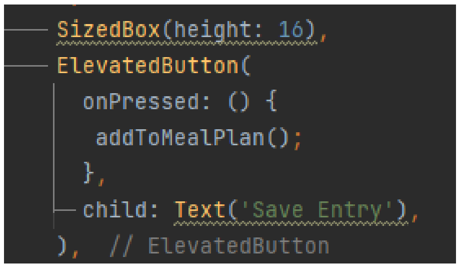
Calorie Check Result

Current Meal Plan is under Calorie Limit. Well done!

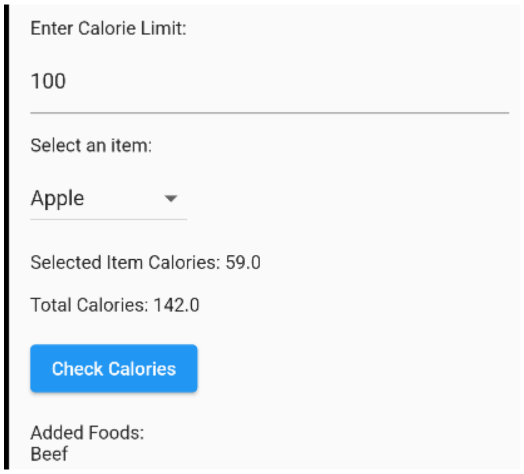
Added Foods:
Chicken
Egg
Asparagus

3. User then saves the selected food items (meal plan) into the database with a date

Pressing the “Save Entry” button will save each entry in the food list to the database.



```
SizedBox(height: 16),
ElevatedButton(
  onPressed: () {
    addToMealPlan();
  },
  child: Text('Save Entry'),
), // ElevatedButton
```



Enter Calorie Limit:

100

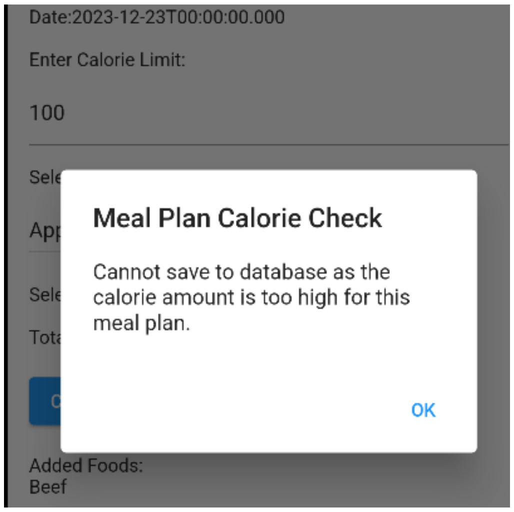
Select an item:

Apple

Selected Item Calories: 59.0

Total Calories: 142.0

Added Foods:
Beef



Date:2023-12-23T00:00:00.000

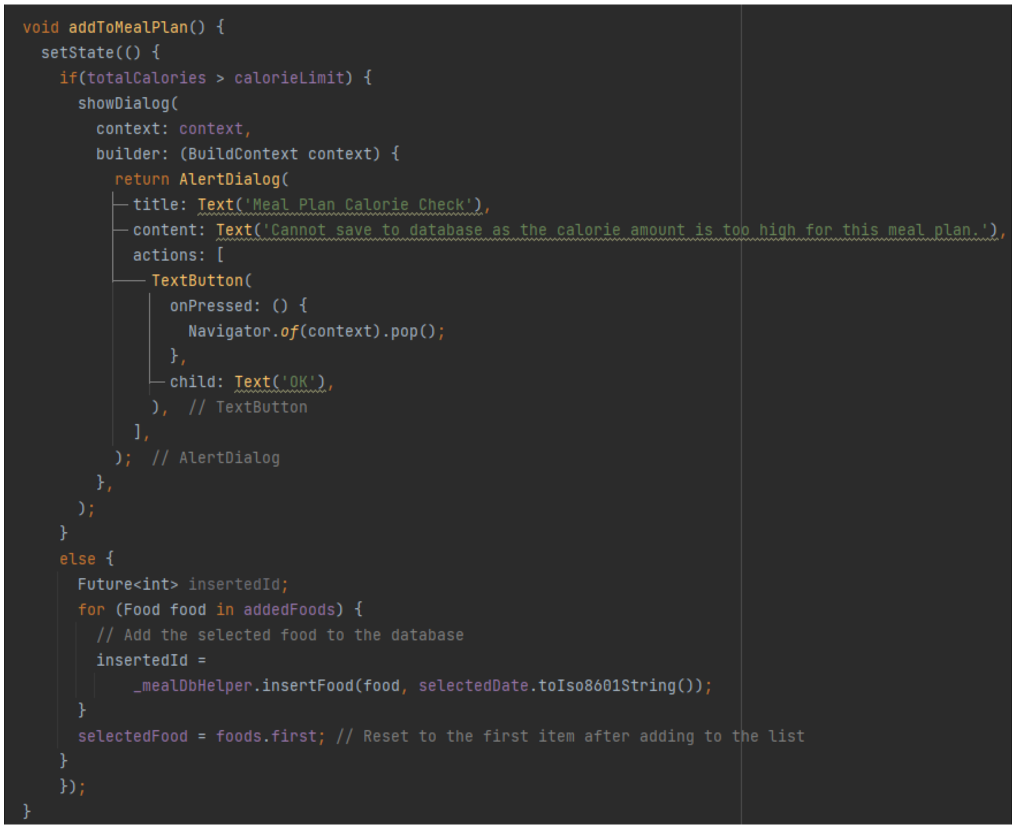
Enter Calorie Limit:

100

Meal Plan Calorie Check

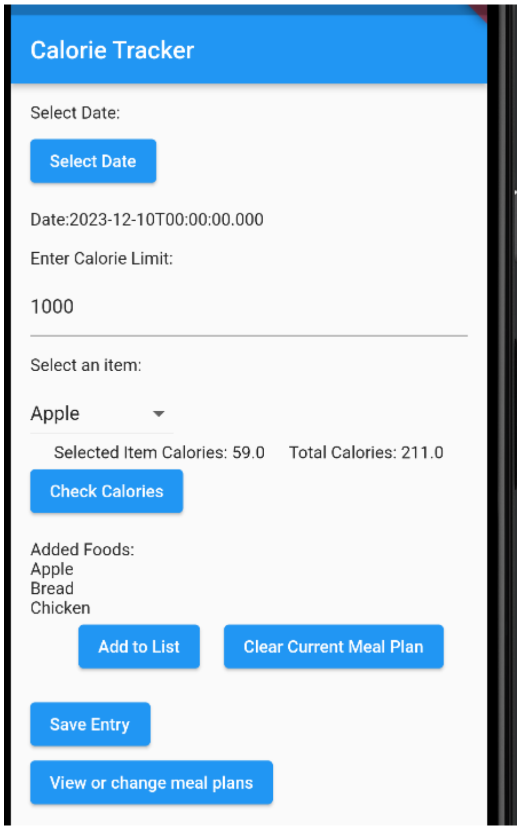
Cannot save to database as the calorie amount is too high for this meal plan.

Added Foods:
Beef



```
void addToMealPlan() {
  setState(() {
    if(totalCalories > calorieLimit) {
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: Text('Meal Plan Calorie Check'),
            content: Text('Cannot save to database as the calorie amount is too high for this meal plan.'),
            actions: [
              TextButton(
                onPressed: () {
                  Navigator.of(context).pop();
                },
                child: Text('OK'),
              ), // TextButton
            ],
          ); // AlertDialog
        },
      );
    }
    else {
      Future<int> insertedId;
      for (Food food in addedFoods) {
        // Add the selected food to the database
        insertedId =
            _mealDbHelper.insertFood(food, selectedDate.toIso8601String());
      }
      selectedFood = foods.first; // Reset to the first item after adding to the list
    }
  });
}
```

4. A query feature in the app to display meal plan for a date (if found in the database)



Calorie Tracker

Select Date:

Select Date

Date:2023-12-10T00:00:00.000

Enter Calorie Limit:

1000

Select an item:

Apple

Selected Item Calories: 59.0 Total Calories: 211.0

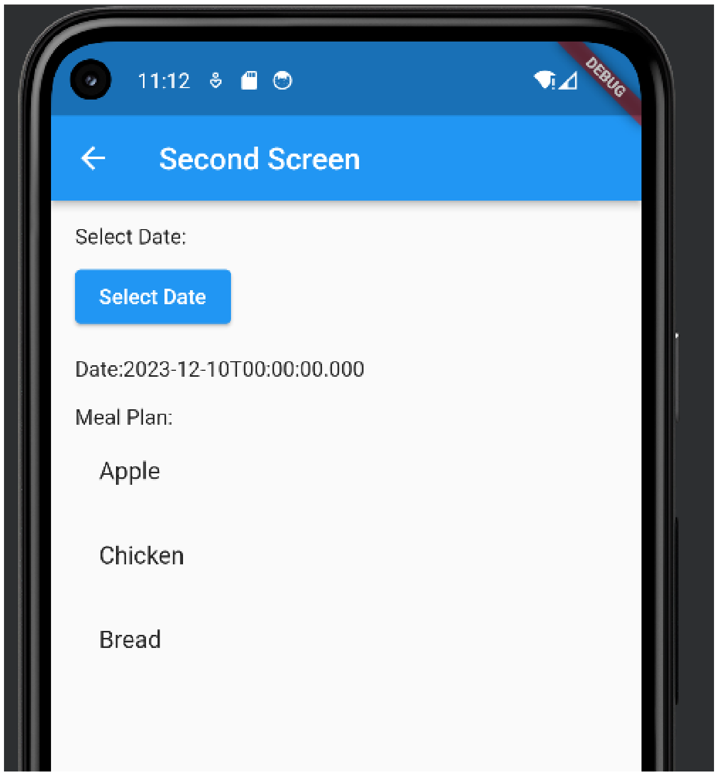
Check Calories

Added Foods:
Apple
Bread
Chicken

Add to List Clear Current Meal Plan

Save Entry

View or change meal plans



11:12

DEBUG

Second Screen

Select Date:

Select Date

Date:2023-12-10T00:00:00.000

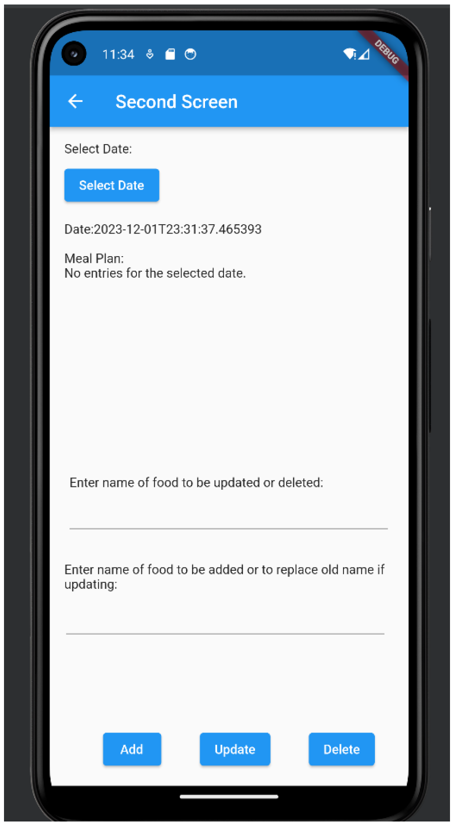
Meal Plan:

Apple

Chicken

Bread

5. An add, delete, and update feature in the app to add, delete, or update entries


11:34

DEBUG

Second Screen

Select Date:

Select Date

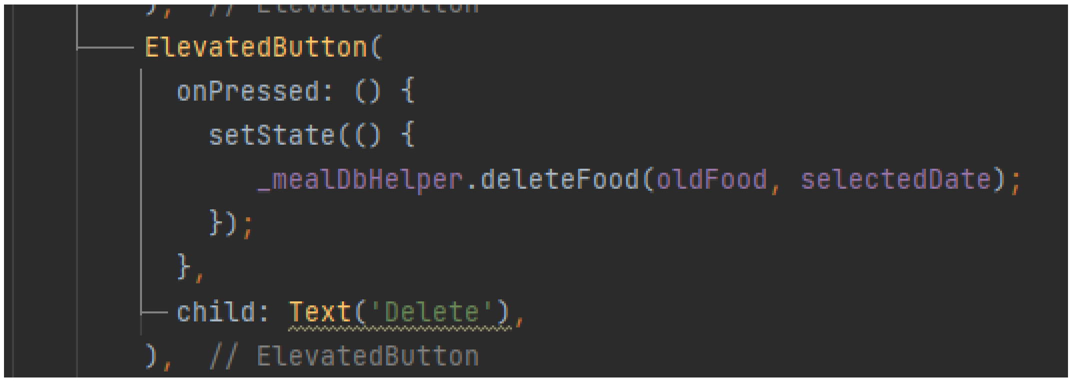
Date:2023-12-01T23:31:37.465393

Meal Plan:
No entries for the selected date.

Enter name of food to be updated or deleted:

Enter name of food to be added or to replace old name if updating:

Add Update Delete


```
ElevatedButton(
  onPressed: () {
    setState(() {
       _mealDbHelper.deleteFood(oldFood, selectedDate);
    });
  },
  child: Text('Delete'),
), // ElevatedButton
```