Smart Grocery Project Final Report

Prepared by
Hrishikesh Badve, Sudhanshu Basuroy, Utsav Sharma, Mridvika Suresh
***for use in* CS 442**
at the
University of Illinois Chicago

Spring 2024

Table of Contents

List of Figures

List of Tables

I Project Description

1 Project Overview

The project is about a platform that is designed to assist home cooks by suggesting recipes based on the ingredients they currently have. It houses a vast collection of recipes, allowing users to explore and discover new dishes. The key feature of this platform is its ability to recommend recipes, which not only helps users utilize their ingredients effectively but also introduces them to a variety of dishes they can prepare at home.

2 Project Domain

The project falls under the non-technical domain of food technology and the technical domain of mobile application development. With the broad topic being helping the application's users discover new recipes to make, it goes beyond that in how users find new recipes with added features to help them manage their grocery inventories.

3 Relationship to Other Documents

This document refers to the Smart Grocery Project Report [1] as this project implemented in part the ideas proposed in the report prepared by Group 25 in the CS 440 Software Engineering class of Fall 2022. The ideas implemented include browsing through recipes, users adding and sharing their own recipes, and filtering recipes by dietary preferences.

4 Naming Conventions and Definitions

4a Definitions of Key Terms

Recipe - A set of instructions for preparing a specific dish, including a list of ingredients and steps for combining and cooking them.

API - Application Programming Interface. A communication interface to exchange information between an application and a database

API Recipe – Recipe fetched from a publicly available API

User Recipe - Recipe created by users of the application for other users to view

Ingredient – A component used in the preparation of a recipe

Instruction - A direction given to guide the creation of a dish from a recipe

Shopping List – A list of ingredients in a recipe that a user intends to buy

Favorite – A recipe saved by a user for later viewing

Filter – A mechanism by which the list of recipes can be modified to show only those meeting certain criteria

4b UML and Other Notation Used in This Document

This document follows the Smart Grocery Project Report as described by group 25 in [1]

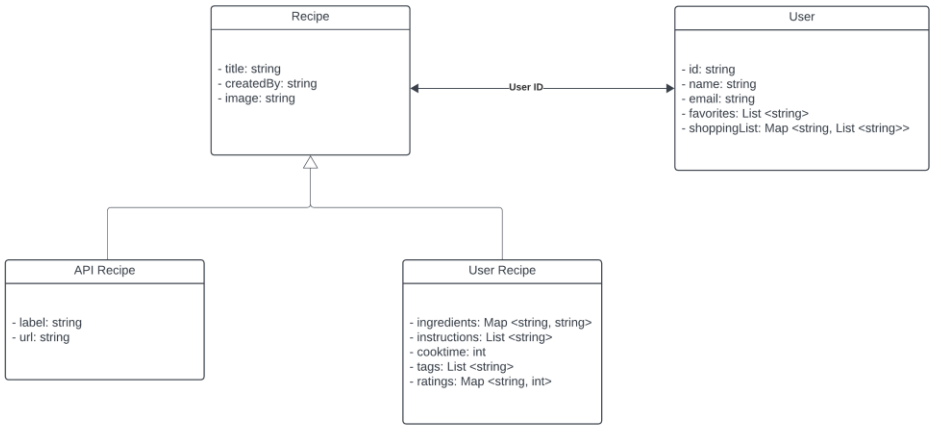

Figure 1 UML Class Diagram

4c Data Dictionary for Any Included Models

- **Recipe** = Recipe ID + Recipe Image URL + Ingredients: {Ingredient 1 + Ingredient 2 + ... Ingredient N} + Instructions: {Instruction 1 + Instruction 2 + ... Instruction N} + Author User ID + Recipe Rating + Recipe Tags {Tag 1 + Tag 2 + ... Tag N}
- **Ingredient** = Substance + Quantity
- **Instruction** = Direction to follow
- **Recipe Rating** = 1.0 to 5.0
- **Recipe Tag** = Category to filter recipe by

II Project Deliverables

This application allows a user to search for recipes based on ingredients available in their pantry. Users can browse through recipes on the application, using filters to find specific recipes, as well as rate the recipes (that ends up contributing to overall recipe ratings) and favorite recipes they like. These favorite recipes are visible on their profile in the application. Furthermore, users are allowed to add recipes of their own, which will be moderated before they are made public. The application is connected to an API that pulls recipes from different websites, giving the users plenty of choices for a specific recipe, which they can search through using any keyword when they access the API. Finally, the user can add a specific recipe to their shopping cart. Users can search for grocery stores nearby if they would like to shop for them.

1 First Release

Release 1 Date: Feb 23, 2024

The Smart Grocery application allows users to view recipes based on the ingredients they have in their pantry at home. The first release demonstrates two primary features of the application. describes the “Recipe Recommendation” features that allows users to pick 5 ingredients from which the application will recommend recipes to the users containing the 5 ingredients they’ve selected. Additionally, the “Recipe Exploration” feature allows users to view recipes on the applications.

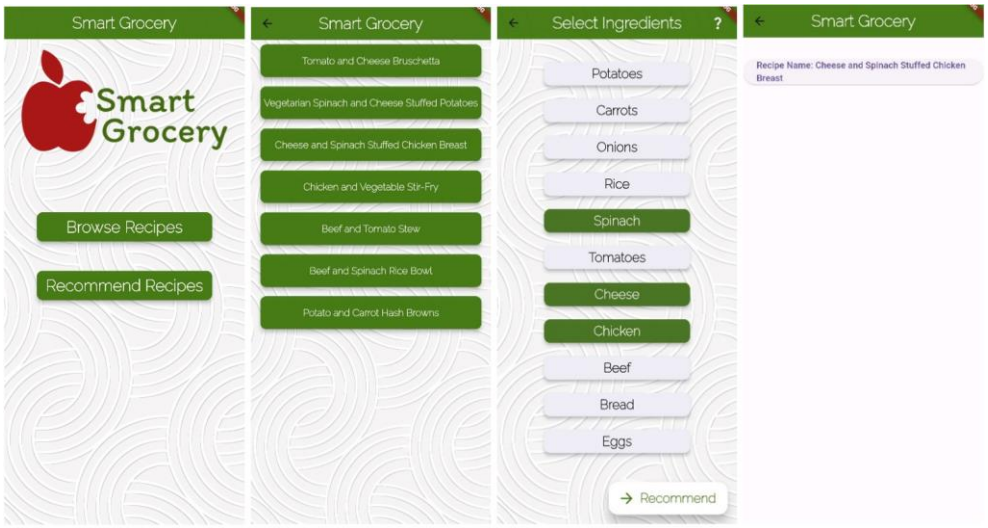

Figure 2: Release 1 Screenshots

2 Second Release

Release 2 Date: Mar 29, 2024

This release enhances the recipe recommendation and exploration systems further, now allowing users to choose from a much larger database of ingredients. There is also a wider range of recipes being stored, with an updated algorithm so the users can view recipes with partial ingredient matches, rather than exact matches. Additionally, users can favorite recipes they like to view later. A login functionality was implemented to save their favorite recipe information.

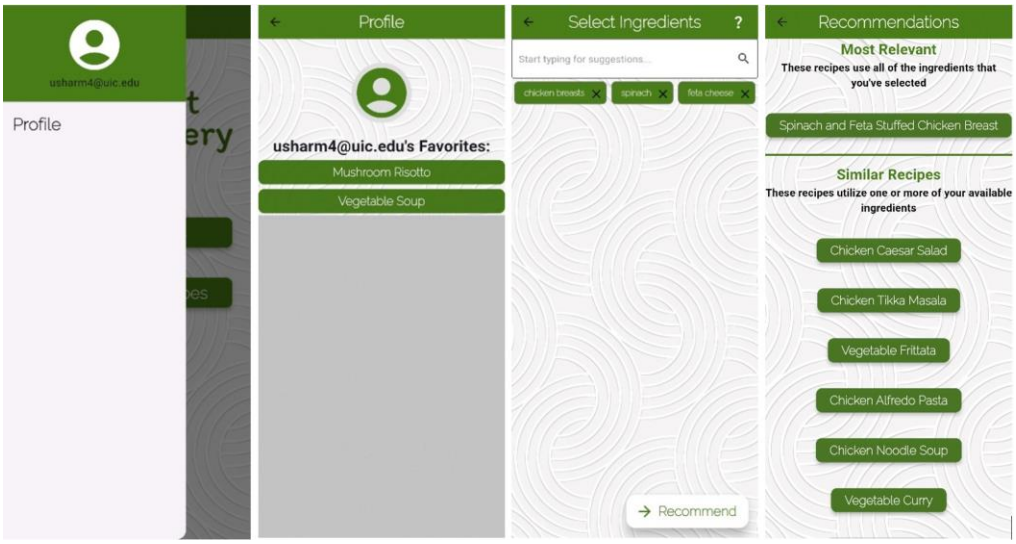

Figure 3: Release 2 Screenshots

3 Third Release

Release 3 Date: Apr 24, 2024

The structure of the application was changed to allow users to navigate the application through bottom bar navigation allowing them to access primary features of the application. In this release, a user can add a recipe to the existing recipes in the application and they can view it in their user profile under "My Recipes". The application was made to be more user friendly for this release. The user can rate recipes, and filter according to dietary preferences. The users can also add items to a shopping list and find grocery stores nearby if they would like to purchase those ingredients.

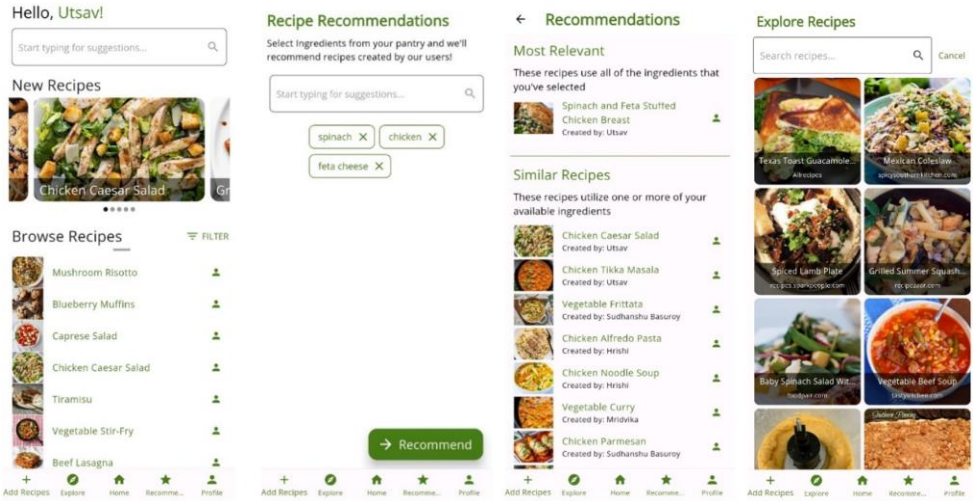

Figure 4: Release 3 Screenshots

4 Comparison with Original Project Design Document

Our prototype differs a bit from the original document. The original document requires the application to connect with grocery stores nearby to recommend recipes based on the availability of the products. We were not able to implement that functionality as there are no specific easily accessible APIs that allowed us to check for grocery availability from stores nearby. We instead allowed the users to enter grocery ingredients as they see fit to get recipe recommendations. Furthermore, the original project design document had a barcode scanner feature and had functionality that required using ML. The barcode scanner would require specific libraries as well as a database of product information for the scanner to detect. Our application can find grocery stores nearby as well as recommend recipes for users and have users view other user recipes on the application.

III Techniques and Methodologies

Pair programming was used in developing this application. By using pair programming, we were able to identify code smells in each other's code as well as working at a fast pace. We used CI/CD to continuously synchronize our codebase, this was done using GitHub.

Our application was developed on Android Studio using Flutter. Flutter was selected due to the nature of the language of having reusable widget components throughout the application development cycle. We used Firebase as our backend, this was used authentication, storage of recipes and image storage. We used Jira as our CASE tool, in specific, we made use of Kanban boards to keep track of the tasks that needed to get done and moved it from to do, to in progress to done. We also used the Edamam API to retrieve a large selection of recipes. This API was selected because the free tier of this API had quotas that we could use for the prototype. Postman was used to test the API and its functionality.

IV Testing

1 Items to be Tested

Registration module, login module, browse recipe module, recommend recipe module, favorite recipe module, recipe rating module, shopping list module, add recipe module, explore recipe/API module.

2 Test Specifications

ID#1 – Registration Functionality

Description: The system must allow the user to successfully register a new account.

Items covered by this test: Registration Module

Requirements addressed by this test: Access Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Test ID#2 depends on the success of this test.

Test Procedures: User registers a new account in the application.

Input Specification: Valid name, valid email id in the correct email id format, valid password (Having more than 5 characters), re-confirmed password must be an exact match of password entered.

Output Specifications: Registration successful screen pops up, with text stating 'Registration Successful. Thank you for registering!'

Pass/Fail Criteria: Pass if registration successful screen pops up, else fail.

ID#2 – Successful Login

Description: Test whether the user can successfully log into a valid account.

Items covered by this test: Login Module

Requirements addressed by this test: Access Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#1. Tests with ID# 4,5,6,7,8,9,10,11,12,13,14 depend on the success of this test.

Test Procedures: User enters valid credentials and tries to log in.

Input Specification: Valid email id in the correct email id format, valid password (Matching the password set for this email id).

Output Specifications: Login is successful, home page shows up.

Pass/Fail Criteria: Pass if home page pops up, else fail.

ID#3 – Invalid Login

Description: Test whether the application tells the user if his/her email or password is incorrect.

Items covered by this test: Login Module

Requirements addressed by this test: Access Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: N/A.

Test Procedures: User enters invalid details and tries to log in.

Input Specification: Invalid email id in any format, invalid password (Not matching the password set for this email id).

Output Specifications: Login is unsuccessful, a pop-up message mentioning that something is incorrect shows up.

Pass/Fail Criteria: Pass if user is notified that the login credentials are incorrect, else fail.

ID#4 – Browse Recipe Filter

Description: Users can filter using tags and see different sets of recipes in the browse recipe list.

Items covered by this test: Browse Recipe Module

Requirements addressed by this test: Filter Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User adds filters in the browse recipe section.

Input Specification: Click on any filter category (E.g. Dessert), click on the 'ok' button.

Output Specifications: All recipes with the tags specified by the filter are now shown in the recipe list (E.g. For dessert, the recipes that show up are blueberry muffins, tiramisu, and chocolate chip cookies).

Pass/Fail Criteria: Pass if the required recipes show up in the list, else fail.

ID#5 – Recommend Recipe Functionality

Description: User gets recipe recommendations based on their ingredient input.

Items covered by this test: Recommend Recipe Module

Requirements addressed by this test: Functional Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User goes to the recommend recipe page, enters ingredients and gets recipes recommended based on the ingredients input.

Input Specification: Click on the search bar, input any ingredients (E.g. Chicken Breasts), click on the recommend button.

Output Specifications: All recipes which have an exact match of the ingredients input show up under 'Most Relevant' and recipes with similar ingredients will show up below 'Similar Recipes'.

Pass/Fail Criteria: Pass if the required recipes show up in both lists, else fail.

ID#6 – Favorite Button Functionality

Description: User should be able to favorite any recipe.

Items covered by this test: Favorite Recipe Module

Requirements addressed by this test: User Engagement Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User goes to a recipe's description page and adds it to his/her favorite list.

Input Specification: Click on the favorite button in any recipe description page, click on user profile page, click on the 'My Favorite' button.

Output Specifications: On clicking the heart shaped favorite button, the button color turns red, and a text prompt is shown to the user stating that 'recipe name added to favorites!'. On navigating to 'my favorite' for the user, the recipe name must now show up in the list.

Pass/Fail Criteria: Pass if the heart button turns red and the recipe name shows up in the favorite list, else fail.

ID#7 – Ratings Functionality

Description: User should be able to rate any recipe.

Items covered by this test: Recipe Rating Module

Requirements addressed by this test: User Engagement Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User goes to a recipe's description page and gives his/her rating.

Input Specification: Click on the stars in any recipe description page, click on the back button, and click on the recipe again to get back into the recipe description.

Output Specifications: On clicking the stars, the stars turn gold specifying what the rating given is. A pop up comes up stating that the rating has been given. After going back out and clicking on the recipe again, the rating shown next to the recipe name is updated.

Pass/Fail Criteria: Pass if the stars turn gold, text shows up indicating the rating has been given and the recipe rating gets updated on the recipe page, else fail.

ID#8 – Add Items to Shopping List

Description: User should be able to add items to the shopping list.

Items covered by this test: Shopping List Module

Requirements addressed by this test: Functional Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2. Test ID#9 Depends on the success of this test.

Test Procedures: User goes to 'Add to Shopping List' page and adds required ingredients to the shopping list.

Input Specification: Click on the 'Add to Shopping List' button in any recipe description page, click on the check box for any ingredients and click on 'Add to Shopping List'. Navigate to the user profile and click on the 'View Shopping List' button.

Output Specifications: On clicking the required ingredients and clicking on 'Add to Shopping List', a text prompt must show up stating that 'Item/Items added to shopping list successfully'. On navigating to the view shopping list page in the user profile, the recipe and ingredients which were added should show up.

Pass/Fail Criteria: Pass if the text prompt shows up and if the correct recipe and ingredients that were added show up in the shopping list, else fail.

ID#9 – Cross Out Items in the Shopping List

Description: User should be able to cross out items in the shopping list.

Items covered by this test: Shopping List Module

Requirements addressed by this test: Functional Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2 and Test ID#8.

Test Procedures: User goes to the shopping list page on their profile and checks out items.

Input Specification: Click on the 'View Shopping List' button in the user profile, click on any recipe, any ingredient and check the box next to it.

Output Specifications: On checking the box next to ingredients, the ingredient name must be striked out.

Pass/Fail Criteria: Pass if the ingredient name is successfully striked out after checking the box next to it, else fail.

ID#10 – Add Recipe Functionality

Description: User should be able to add recipes to the database.

Items covered by this test: Add Recipe Module

Requirements addressed by this test: Functional Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User goes to the add recipe page and enters required details to add a recipe.

Input Specification: Click on the 'Add Recipe' button in the bottom navigation bar, enter recipe title, cook time, add ingredients, instructions, tags, and an image for the recipe and finally click on the submit recipe button.

Output Specifications: On adding all required values and submitting recipe, a pop up will show up stating that the recipe is added and will be reviewed soon. When the user navigates to the 'My Recipes' page on their profile, they can view the newly added recipe. The Approved status will be shown as 'Pending'.

Pass/Fail Criteria: Pass if the recipe is successfully added and can be viewed in the 'My Recipes' page with the status as 'Pending', else fail.

ID#11 – Explore Recipe Functionality

Description: User should be able to view and interact with the explore recipes page.

Items covered by this test: Explore Recipe/API Module

Requirements addressed by this test: Functional Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2. Test ID#12 depends on the success of this test.

Test Procedures: User goes to the explore recipe page and clicks on a recipe.

Input Specification: Click on the 'Explore Recipe' button in the bottom navigation bar, click on any recipe.

Output Specifications: On clicking the 'Explore Recipe' button, the user will be navigated to the explore recipe page where he/she can view all the recipes. On clicking any recipe, the web view of that recipe opens.

Pass/Fail Criteria: Pass if the explore recipe page opens and the web view of the required recipe opens upon clicking it, else fail.

ID#12 – Explore Recipe Favorite Button Functionality

Description: User should be able to favorite any recipe, opened in the web view through the explore recipe page.

Items covered by this test: Explore Recipe/API Module

Requirements addressed by this test: User Engagement Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2 and Test ID#11.

Test Procedures: User goes to the explore recipe page, clicks on a recipe, then clicks on the favorite button in the recipe web view.

Input Specification: Click on the 'Explore Recipe' button in the bottom navigation bar, click on any recipe, click on the 'Favorite' button in web view of the recipe.

Output Specifications: On clicking the 'Explore Recipe' button, the user will be navigated to the explore recipe page where he/she can view all the recipes. On clicking any recipe,

the web view of that recipe opens. On clicking the ‘Favorite’ button, the heart turns red. On navigating to ‘my favorite’ for the user, the recipe name must now show up in the list.

Pass/Fail Criteria: Pass if the heart button turns red and the recipe name shows up in the favorite list, else fail.

ID#13 – Recipe Search Bar

Description: User should be able to search for any recipes

Items covered by this test: Browse Recipe Module

Requirements addressed by this test: Functionality Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User clicks on search bar, types in names and searches for recipes.

Input Specification: Click on the search bar in the home page, type in any recipe name, click on the recipe in the suggestions that pop up.

Output Specifications: On clicking on the recipe name in the suggestions, the recipe description page for that specific recipe comes up.

Pass/Fail Criteria: Pass if the recipe description page for the specific recipe comes up, else fail.

ID#14 – Nearby Stores Button

Description: User should be able to look at nearby grocery stores by clicking the ‘Find Grocery Stores Nearby’ button.

Items covered by this test: Shopping List Module

Requirements addressed by this test: Functionality Requirements.

Environmental needs: No specific hardware or software requirements.

Intercase Dependencies: Depends on success of Test ID#2.

Test Procedures: User clicks on ‘view shopping list’ button, then clicks on the ‘find grocery stores nearby’ button to open the map.

Input Specification: Click on the ‘view shopping list’ button, then click on the ‘find grocery stores nearby’ button.

Output Specifications: On clicking the ‘find grocery stores nearby’ button, the map opens, showing grocery stores nearby.

Pass/Fail Criteria: Pass if the map comes up, else fail.

3 Test Results

ID#1 – Registration Functionality

Date(s) of Execution: 04/17/2024

Staff conducting tests: Hrishikesh

Expected Results: User should be able to successfully register a new account, reflected in the database.

Actual Results: User is able to successfully register a new account, and it is reflected in the database.

Test Status: Pass.

ID#2 – Successful Login

Date(s) of Execution: 04/17/2024

Staff conducting tests: Hrishikesh

Expected Results: User should be able to successfully login with valid credentials.

Actual Results: User is able to successfully login and home page comes up.

Test Status: Pass. Hrishikesh was able to log in using the newly added credentials.

ID#3 – Invalid Login

Date(s) of Execution: 04/17/2024

Staff conducting tests: Hrishikesh

Expected Results: User should be notified that the login credentials are incorrect.

Actual Results: User is not able to log in using invalid credentials, but there is no message given to the user suggesting that the credentials are incorrect.

Test Status: Fail. Even though the user is unable to log in with invalid credentials, there is no notification given to the user about the same.

ID#4 – Browse Recipe Filter

Date(s) of Execution: 04/18/2024

Staff conducting tests: Mridvika

Expected Results: User should be able to browse through recipes after filtering them using tags.

Actual Results: User is able to filter recipes using tags and browse the results.

Test Status: Pass.

ID#5 – Recommend Recipe Functionality

Date(s) of Execution: 04/18/2024

Staff conducting tests: Sudhanshu

Expected Results: User should be able to get recipe recommendations based on the ingredients they input.

Actual Results: User is able to get specific recipe recommendations after inputting ingredients.

Test Status: Pass.

ID#6 – Favorite Button Functionality

Date(s) of Execution: 04/17/2024

Staff conducting tests: Hrishikesh

Expected Results: User should be able to favorite any recipe and add it to their favorites list, along with it being reflected in the database.

Actual Results: User is able to favorite any recipe and add it to their favorites list, and the result is reflected in the database.

Test Status: Pass.

ID#7 – Ratings Functionality

Date(s) of Execution: 04/19/2024

Staff conducting tests: Utsav

Expected Results: User should be able to rate any recipe, receive a pop-up stating that they have done so, have the ratings reflected in the UI as well as the database.

Actual Results: User is able to rate any recipe, the ratings are reflected in the UI as well as the database, but there is no text pop-up indicating that the rating was successful.

Test Status: Fail. Although the functioning of the ratings feature is correct, there is no text pop-up which indicates to the user that the rating was successful, and this was a passing criterion for this test.

ID#8 – Add Items to Shopping List

Date(s) of Execution: 04/17/2024

Staff conducting tests: Hrishikesh

Expected Results: User should be able to add items to their shopping list, receive a text prompt stating that the addition of items was successful, and finally be able to view the added items in their shopping list page.

Actual Results: User is able to add items to the shopping list, get a text prompt stating that they have added items, and view these added items in their shopping list page.

Test Status: Pass.

ID#9 – Cross Out Items in the Shopping List

Date(s) of Execution: 04/19/2024

Staff conducting tests: Utsav

Expected Results: User should be able to cross out items from the shopping list by clicking the check boxes next to them.

Actual Results: User is able to cross out items from the shopping list by clicking the check boxes next to them.

Test Status: Pass.

ID#10 – Add Recipe Functionality

Date(s) of Execution: 04/18/2024

Staff conducting tests: Mridvika

Expected Results: User should be able to add a new recipe and view it with status being pending.

Actual Results: User is able to add a new recipe and view it with the status being pending.

Test Status: Pass.

ID#11 – Explore Recipe Functionality

Date(s) of Execution: 04/17/2024

Staff conducting tests: Hrishikesh

Expected Results: User should be able to view and interact with the explore recipes page.

Actual Results: User is able to view the explore recipe page and open up the web view of the recipe after clicking on it.

Test Status: Pass.

ID#12 – Explore Recipe Favorite Button Functionality

Date(s) of Execution: 04/18/2024

Staff conducting tests: Sudhanshu

Expected Results: User should be able to favorite any recipe from the explore recipe page and add it to their favorites list, along with it being reflected in the database.

Actual Results: User is able to favorite any recipe from the explore recipe page and add it to their favorites list, along with it being reflected in the database.

Test Status: Pass.

ID#13 – Recipe Search Bar

Date(s) of Execution: 04/19/2024

Staff conducting tests: Utsav

Expected Results: User should be able to search for any recipes from the search bar.

Actual Results: User is able to search for any recipe using the search bar.

Test Status: Pass.

ID#14 – Nearby Stores Button

Date(s) of Execution: 04/19/2024

Staff conducting tests: Utsav

Expected Results: User should be able to look at nearby grocery stores by clicking the ‘Find Grocery Stores Nearby’ button.

Actual Results: On clicking the ‘find grocery stores nearby’ button, the map opens, showing grocery stores nearby.

Test Status: Pass.

4 Regression Testing

ID#2, ID#4, ID#5, ID#6 and ID#12 were tested multiple times.

V Inspection

1 Items to be Inspected

Hrishikesh’s Code:

```
class BaseWidget extends StatefulWidget {
  final Widget body;
  BaseWidget({required this.body});

  @override
  _BaseWidgetState createState() => _BaseWidgetState();
}

class _BaseWidgetState extends State<BaseWidget> {
  int _selectedIndex = 0;

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
      switch (index) {
        case 0:
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => AddRecipePage()),
          );
          break;
        case 1:
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => explorePageAPI()),
          );
          break; // <-- And this
```

```
        case 2:
          Navigator.pushAndRemoveUntil(
            context,
            MaterialPageRoute(builder: (context) => home(title: 'smartGrocery',)),
                (Route<dynamic> route) => false,
          );
          break;
        case 3:
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => IngredientsSelection()),
          );
          break;
        case 4:
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => ProfilePage()),
          );
          break;
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // Removed appBar
      body: widget.body,
      bottomNavigationBar: BottomNavigationBar(
        type: BottomNavigationBarType.fixed,
        unselectedLabelStyle: TextStyle(fontFamily: 'opensans'),
        selectedLabelStyle: TextStyle(fontFamily: 'opensans'),
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.add),
            label: 'Add Recipes',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.explore),
            label: 'Explore',
          ), // <-- And this
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: 'Home',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.star),
            label: 'Recommend',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.person),
            label: 'Profile',
          ),
        ],
        currentIndex: _selectedIndex,
        selectedItemColor: const Color(0xff4417810),
        unselectedItemColor: const Color(0xff4417810),
        onTap: _onItemTapped,
```

```
      ),
    );
  }
}
```

Sudhanshu's Code:

```
Widget build(BuildContext context) {
  return FutureBuilder<List<Map<String, dynamic>>>(
    future: recipesFuture,
    builder: (BuildContext context, AsyncSnapshot<List<Map<String, dynamic>>> snapshot) {
      if (snapshot.hasError) {
        return const Text("Something went wrong");
      }

      if (snapshot.connectionState == ConnectionState.done) {
        print("Called getRecipes() from RecipeList");
        List<Map<String, dynamic>> recipes = snapshot.data!;
        List<Map<String, dynamic>> filteredRecipesData;
        if (widget.selectedCategories.isEmpty) {
          filteredRecipesData = recipes;
        } else {
          filteredRecipesData = recipes.where((recipe) {
            List<String> recipeTags = List<String>.from(recipe['tags']);
            return recipeTags.any((tag) => widget.selectedCategories.contains(tag));
          }).toList();
        }

        List<Widget> filteredRecipes = filteredRecipesData.map((recipe) {
          Map<String, dynamic> ingredientsMap = recipe['ingredients'];
          String ingredientsString = ingredientsMap.entries.map((e) =>
'${e.key[0].toUpperCase() + e.key.substring(1,e.key.length)}: ${e.value}').join(';');
          List<dynamic> instructionsList = recipe['instructions'];
          String instructionsString = instructionsList.join(';');
          List<String> tags = recipe['tags'] != null ? List<String>.from(recipe['tags'])
: [];
          return ListTile(
            leading: recipe['image'] != null
                ? Container(
              width: 55,
              height: 55,
              child: Image.network(
                recipe['image'],
                fit: BoxFit.cover,
              ),
            )
                : null,
            title: Text('${recipe['title']}',
                style: TextStyle(
                    fontFamily: 'opensans',
                    fontSize: 15.0,
                    fontWeight: FontWeight.normal,
                    color: const Color(0xff4417810))),
            trailing: Icon(Icons.person, color: const Color(0xff4417810), size: 20,),
```

```
            onTap: () {
              _navigateToRecipePage(
                recipe['title'],
                ingredientsString,
                instructionsString,
                recipe['id'],
                context,
              );
            },
          );
        }).toList();

        return Column(children: filteredRecipes);
      }

      return const Center(child: CircularProgressIndicator());
    },
  );
}
```

Mridvika's Code:

```
void _launchURL() async {
  const url =
'https://www.google.com/maps/search/?api=1&query=grocery+stores+near+me';
  if (await canLaunchUrlString(url)) {
    await launchUrlString(url);
  } else {
    throw 'Could not launch $url';
  }
}
```

```
floatingActionButton: FloatingActionButton.extended(
    onPressed: _launchURL,
    tooltip: 'Find Grocery Stores',
    backgroundColor: Color(0xff4417810).withOpacity(1), // Set the button color to green
    label: const Text(
    'Find Grocery Stores Nearby',
    style: TextStyle(
        fontFamily: 'opensans',
        fontSize: 15,
        fontWeight: FontWeight.normal,
        color: Colors.white,
    ),

  ), icon: const Icon(Icons.store, color: Colors.white, size: 20),
```

Utsav's Code:

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';
import 'package:flutter_inappwebview/flutter_inappwebview.dart';
import 'package:fluttertoast/fluttertoast.dart';
import 'package:shared_preferences/shared_preferences.dart';
/*
================================================================================
===
Draws the webview for the recipe
================================================================================
===
 */
class RecipeWebViewPage extends StatefulWidget {
  final String recipeUrl;
  final String recipe_id;

  RecipeWebViewPage({required this.recipeUrl, required this.recipe_id});

  @override
  _RecipeWebViewPageState createState() => _RecipeWebViewPageState();
}

class _RecipeWebViewPageState extends State<RecipeWebViewPage> {
  final FirebaseFirestore db = FirebaseFirestore.instance;
  bool isFavorite = false;

  @override
  void initState() {
    super.initState();
    checkFavoriteStatus();
  }

  Future<void> checkFavoriteStatus() async {
    final prefs = await SharedPreferences.getInstance();
    final uid = prefs.getString('uid');
    final doc = await db.collection('Users').doc(uid).get();
    final favorites = List<String>.from(doc.data()?['favorites'] ?? []);
    setState(() {
      isFavorite = favorites.contains(widget.recipe_id);
    });
  }

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
      onWillPop: () async {
        // You can do additional things here if needed
        return true; // return true if the route should be popped
      },
      child: Scaffold(
          appBar: AppBar(
            iconTheme: IconThemeData(color: Color(0xff4417810)),
  //title: Text('Recipe'),
  actions: [
    Row(
      children: [
```

```
          IconButton(
            icon: Icon(isFavorite ? Icons.favorite : Icons.favorite_border,
color: isFavorite ? Colors.red : Colors.green),
          onPressed: () async {
            final prefs = await SharedPreferences.getInstance();
            final uid = prefs.getString('uid');
            if (isFavorite) {
              db.collection('Users').doc(uid).update({"favorites":
FieldValue.arrayRemove([widget.recipe_id])});
              Fluttertoast.showToast(
                msg: 'Recipe removed from favorites!',
                toastLength: Toast.LENGTH_SHORT,
                gravity: ToastGravity.BOTTOM,
                backgroundColor: Colors.red,
                textColor: Colors.white,
              );
            } else {
              db.collection('Users').doc(uid).update({"favorites":
FieldValue.arrayUnion([widget.recipe_id])});
              Fluttertoast.showToast(
                msg: 'Recipe added to favorites!',
                toastLength: Toast.LENGTH_SHORT,
                gravity: ToastGravity.BOTTOM,
                backgroundColor: Colors.red,
                textColor: Colors.white,
              );
            }
            setState(() {
              isFavorite = !isFavorite;
            });
          },
        ),
          Text(
          //isFavorite ? 'Favorited' : 'Not Favorited',
          'Favorite',
          style: TextStyle(color: Colors.green, fontFamily: 'opensans',
fontWeight: FontWeight.bold, fontSize: 15.0),
        ),

      ],
    ),
  ],
),
          body: InAppWebView(
            initialUrlRequest: URLRequest(
              url: WebUri(widget.recipeUrl),
            ),
          )
      ),
    );
  }
}
```

2 Inspection Procedures

The code review checklist below is obtained from Michaela Greiler's Code Review Checklist [2]

Implementation

● Does this code change do what it is supposed to do?

● Can this solution be simplified?

● Does this change add unwanted compile-time or run-time dependencies?

● Was a framework, API, library, service used that should not be used?

● Was a framework, API, library, service not used that could improve the solution?

● Is the code at the right abstraction level?

● Is the code modular enough?

● Would you have solved the problem in a different way that is substantially better in terms of the code's maintainability, readability, performance, security?

● Does similar functionality already exist in the codebase? If so, why isn't this functionality reused?

● Are there any best practices, design patterns, or language-specific patterns that could substantially improve this code?

● Does this code follow Object-Oriented Analysis and Design Principles, like the Single Responsibility Principle, Open-close principle, Liskov Substitution Principle, Interface Segregation, Dependency Injection?

Logic Errors and Bugs

● Can you think of any use case in which the code does not behave as intended?

● Can you think of any inputs or external events that could break the code?

● Error Handling and Logging

● Is error handling done the correct way?

● Should any logging or debugging information be added or removed?

● Are error messages user-friendly?

● Are there enough log events and are they written in a way that allows for easy debugging?

Usability and Accessibility

● Is the proposed solution well designed from a usability perspective?

● Is the API well documented?

● Is the proposed solution (UI) accessible?

● Is the API/UI intuitive to use?

Ethics and Morality

● Does this change make use of user data in a way that might raise privacy concerns?

● Does the change exploit behavioral patterns or human weaknesses?

● Might the code, or what it enables, lead to mental and physical harm for (some) users?

● If the code adds or alters ways in which people interact with each other, are appropriate measures in place to prevent/limit/report harassment or abuse?

● Does this change lead to an exclusion of a certain group of people or users?

● Does this code change introduce any algorithm, AI or machine learning bias?

● Does this code change introduce any gender/racial/political/religious/ableist bias?

Testing and Testability

● Is the code testable?

● Does it have enough automated tests (unit/integration/system tests)?

● Do the existing tests reasonably cover the code change?

● Are there some test cases, input, or edge cases that should be tested in addition?

Dependencies

● If this change requires updates outside of the code, like updating the

documentation, configuration, readme files, was this done?

● Might this change have any ramifications for other parts of the system, backward compatibility?

Security and Data Privacy

● Does this code open the software for security vulnerabilities?

● Are authorization and authentication handled in the right way?

● Is sensitive data like user data, credit card information securely handled and stored? Is the right encryption used?

● Does this code change reveal some secret information (like keys, usernames, etc.)?

● If code deals with user input, does it address security vulnerabilities such as cross-site scripting, SQL injection, does it do input sanitization and validation?

● Is data retrieved from external APIs or libraries checked accordingly?

Performance

● Do you think this code change will impact system performance in a negative way?

● Do you see any potential to improve the performance of the code?

Readability

● Was the code easy to understand?

● Which parts were confusing to you and why?

● Can the readability of the code be improved by smaller methods?

● Can the readability of the code be improved by different function/method or variable names?

● Is the code located in the right file/folder/package?

● Do you think certain methods should be restructured to have a more intuitive control flow?

● Is the data flow understandable?

- Are there redundant comments?
- Could some comments convey the message better?
- Would more comments make the code more understandable?
- Could some comments be removed by making the code itself more readable?
- Is there any commented out code?

3 Inspection Results

What was inspected	Who did the inspection	Time and Date	Results of Inspection
Hrishikesh's Code	Mridvika Suresh	04/26/24 4:30pm	Code could be a little more modular. An extract method could be used to fix repetition in the switch statements. (Readability, Implementation)
	Utsav Sharma	04/26/24 5:15pm	The code is readable but could do with more comments to explain its function. There is also some scope for increasing the modularity of the code and decreasing duplication by defining the string constants in a list and creating a modular BottomNavigationBarItem () function that can be called for each item in the list. (Readability, Implementation).
	Sudhanshu Basuroy	04/26/24 4:38pm	The code is moderately readable, but some comments could be improved or removed.

			Errors during database updates are caught and logged. (Readability, Errors)
Utsav's Code	Mridvika Suresh	04/26/24 5:15pm	This code could use comments as it would help make the code more readable. Furthermore, error handling doesn't seem to be taken care of, which could be beneficial for the Firestore operations. (Logic Errors and Bugs, Readability)
	Hrishikesh Badve	04/26/24 4:00pm	The code is not well-documented. Adding comments explaining the purpose of each function and its parameters would improve readability and maintainability. (Usability and Accessibility, Readability)
	Sudhanshu Basuroy	04/26/24 4:53pm	The code handles adding/removing recipes from favorites and displays appropriate toast messages. The overall readability is moderate, and some comments could be improved or removed. (Implementation, Readability)
Sudhanshu's Code	Mridvika Suresh	04/26/24 4:45pm	While the code considers error handling through error messages, certain error messages are hardcoded. Some print statements can also be removed as it doesn't

			seem relevant to have in the code anymore. (Logic Errors and Bugs)
	Hrishikesh Badve	04/26/24 4:30pm	The code retrieves user data from a shared preference and a Firestore database. It's important to ensure this data is handled securely. (Security and Data Privacy)
	Utsav Sharma	04/26/24 4:55pm	The code requires some documentation in the form of comments to make its function apparent to a reviewer. It handles different cases of execution well, with sufficient error handling. The filteredRecipes function could be optimized for better performance on larger datasets. (Readability, Error Handling, Performance)
Mridvika's Code	Hrishikesh Badve	04/26/24 5:00pm	The performance of this code largely depends on the performance of the URL launching operations and the loading time of the web view. (Performance)
	Utsav Sharma	04/26/24 4:37pm	The code requires some documentation in the form of comments. It uses appropriate error handling with a throw statement, which is good. The use of const to maintain immutable values is a good practice that is followed. (Readability,

			Error Handing, Implementation)
	Sudhanshu Basuroy	04/26/24 5:22pm	The code handles launching the URL and throws an error if it cannot be launched. Overall, the implementation is straightforward, and the code follows a clear purpose. (Errors, Implementation)

Table 1: Inspection Results

VI Recommendations and Conclusions

2 of the 14 tests shown above failed. The following conclusions can be drawn:

- Ensure the addition of user feedback for all possible operations that the user may complete.
- In both cases, the functionalities are working from a technical perspective, but they are lacking in terms of user experience. Providing clear and immediate feedback to the user is a key aspect of user interface design and can greatly enhance the usability of our application.

From our inspection, it's important to use modular code structures to reduce repetition and improve maintainability in the future. This could involve creating reusable functions or components. Furthermore, implementing robust error handling mechanisms across all parts of the code, especially for operations involving databases and user data would be beneficial. Ensuring data is securely handled is also important. Lastly, well-documented code can greatly improve code quality.

VII Project Issues

1 Open Issues

API Key Security Issue: The API key for Edamam is currently hardcoded in the application, and thus is vulnerable to security issues. If the key were to be extracted from the application, any malicious actor would have the capability to access the API with our key.

Email Verification: While we ask users for their email addresses, we do not have the functionality to verify that email address. This makes our application vulnerable to fake accounts, since a malicious actor could create any number of users to flood the backend.

2 Waiting Room

Login Persistence: When a user logs into the application, their login state is not stored. Therefore, every time that a user closes the application and opens it again, they are presented with the login screen.

Share Recipes: We intended to implement a feature to let users share the recipes that they like with their friends through a messaging service/social media application of their choice. However, this feature moved to the backlog for release 3, and would be looked at if we had a future release.

Moderation backend: At the time of release 3, we have a functionality where users can create their own recipes to add to the application. These recipes are not shown on the application until a boolean variable is set to true. We intend to have a backend system of human moderators to review recipes, which has not been implemented yet.

Recipe Reviews: Currently, users can only rate recipes with stars on a scale of 1-5. In future releases, we plan to have the ability to let users review recipes in more detail with text fields.

Community Forums: We plan to include a community forum to let users interact with each other through chat boards. These chat boards would let users talk about recipes and general food related topics using public messages.

API Collaboration with food website/organization: We are currently using a free tier of the Edamam API. In the future, we would like to integrate an API from an existing food/recipe brand such as Epicurious, Tasty or Food Network.

Nutritional Information: While the Edamam API does give us nutritional information, we need to design a solution to let users choose their recipes based on this nutritional value.

Integration with Grocery Store/Delivery apps: Instacart, UberEats and even Jewel-Osco and Mariano's offer delivery services for groceries. We plan to integrate such a service into the application, so that users can get the items from their shopping list delivered to their homes at the click of a button.

3 Ideas for Solutions

One method to handle API keys within the application would be to utilize the Android Keystore system. According to the developer documentation on Android, "The Android Keystore system lets you store cryptographic keys in a container to make them more difficult to extract from the device. Once keys are in the keystore, you can use them for cryptographic operations, with the key material remaining non-exportable. Also, the keystore system lets you restrict when and how keys can be used, such as requiring user authentication for key use or restricting keys to use only in certain cryptographic modes." [3]

Another potential solution is utilizing a .env file and using the ENVied package for flutter that handles generating a .dart file for us to import and read the API key from. It also gives us the option to obfuscate the key, making it harder to decrypt the installation package and steal the API key. The choice of actual implementation will require more investigation to decide which approach works best with our current platforms and languages.

Email Verification can be handled with minimum programming, since Firebase does all the heavy lifting for us. By calling the sendSignInLinkToEmail() method, we can get Firebase to send a verification email with a link to click to the user, which would complete the authentication for us.

4 Project Retrospective

Using Flutter/Dart for building the application was a wise choice, since it allowed us to perform rapid prototyping to try implementing new features. It also allows us to quickly port the application over to iOS too, since it is a platform independent approach to mobile application development. Had we started creating the application using a native approach like Java/Kotlin for Android (or Swift for iOS), we would have had to spend considerable time and effort to create a port for the other platform.

Dividing the work into independent modules, then assigning those modules to individual team members each release also worked well for us. By analyzing the planned implementation of a feature to identify any dependencies, then assigning all of those to a single team member, we eliminated bottlenecks caused by having to wait around for others to work on their Jira tickets. This, however, led to some work imbalance, where some features that had more work required to set them up and program lead to those team members having to put in more hours. In the future, we would have to work towards finding the right balance for the work split, but since that is dependent on our effort estimates, we would have to make them more accurate first.

We should have also worked with an architecture such as Model-View-Controller to make updating the application easier for us, since our incremental design approach meant that we had a lot of reworking to do for the UI and the backend, causing a significant amount of dead code to float around.

VIII Glossary

API (Application Programming Interface): A set of protocols for building and interacting with software applications

API Recipe: A recipe obtained from a publicly available API

Favorite: A feature that allows users to save recipes they like for later reference

Filter: A feature that enables users to narrow down recipe choices based on specific criteria.

Flutter: An open-source UI software development kit used for building natively compiled applications.

Ingredient: A food item used in the preparation of a dish.

Instruction: Step-by-step directions for preparing a dish according to a recipe.

Recipe: A set of instructions for preparing a particular dish, including a list of ingredients and cooking steps

Recipe Rating: A numerical score from 1.0 to 5.0 assigned by users to rate the quality of a recipe

Recipe Tag: A label used to categorize recipes, making them easier to find and filter.

Shopping List: A list of ingredients needed for a recipe that a user plans to purchase.

UML (Unified Modeling Language): A standardized modeling language used to specify, visualize, construct, and document the artifacts of software systems.

User Recipe: A recipe created and shared by a user of the application

IX References / Bibliography

[1] A. N. G. N. &. M. P. Agnes Folga, "Smart Grocery Project Report," 2022.

[2] [Online]. Available: https://www.michaelagreiler.com/code-review-checklist-2/.

[3] [Online]. Available: https://developer.android.com/privacy-and-security/keystore.

X Index

No Index.