Food For All
Software Design Description

Revision 1.0

20 December 2019

Issued & Authored by
CS 364, Section 02
Semester: Fall 2019

Table of Contents

List of Figures

5.2 Context

5.3 Composition

5.4 Logical

5.5 Dependency

5.6 Information

5.7 Patterns

5.8 Interface

5.8 Interface (continued)

5.9 Structure

5.10 Interaction

5.11 State Dynamics

5.12 Algorithm

5.13 Resources

1 Revision History

Revision	Release Date	Description
0.1	16 November 2019	Initial Revision
0.2	30 November 2019	Version 2
0.3	7 December 2019	Version 3
1.0RC1	20 December 2019	Final Version Release Candidate 1
1.0	20 December 2019	Final Version

2 Introduction

2.1 Purpose

The purpose of this document is to provide a software design description of Food For All, an application platform where a number of people can collaboratively manage recipes, menu planning, and shopping lists.

2.2 Scope

The Food For All software design description defines viewpoints and views of functionality exposed to end users (for recipes, cookbooks, menus, shopping lists, and group management), to administrative users (for moderation of duplicate and shared content), and interactions with external interfaces that make the other functionality possible (for user authentication, calendar publication, recipe ingredient parsing, the application technology stack, and application hosting services).

2.3 Summary

Food For All will provide a web application based interface, designed to be accessible via traditional as well as mobile web browsers, to several tools:

1. A recipe manager that allows a number of people to collaboratively create, customize, and share recipes with one another.
2. A menu planner that allows a number of people to participate in the creation of menus (ie, recipe selection) for a user selectable period of time (ie, daily or weekly).
3. A shopping list manager that will support automatic generation of a list of ingredients necessary to prepare an individual recipe or menu of multiple recipes.
4. User and group management to control who has access to specific instances of recipes, menu planning, and shopping lists.

3 References

IEEE, "IEEE Standard for Information Technology — Systems Design — Software Design Descriptions", July 2009. [Online] Available: http://ieeexplore.ieee.org/servlet/opac?punumber=5167253

CS 364, Section 02, “Food For All Software Requirements Specification”, October 26, 2019. [Online] Available: https://docs.google.com/document/d/1nJTxNCfZNIpFUpiT6TwVlHZ6EWA_n_fx1MZ2x1eZ08k/

"AngularJS Directive with Our First AngularJS Example", Softwaretestinghelp.com, 2019. [Online]. Available: https://www.softwaretestinghelp.com/angularjs-directive-example/. [Accessed: 07- Dec- 2019].

"Single Page Application Architecture in AngularJS - Tech Funda", *Techfunda.com*, 2019. [Online]. Available: http://techfunda.com/howto/966/single-page-application-architecture. [Accessed: 11- Dec- 2019].

"Web Sites - NearlyFreeSpeech.net", *NearlyFreeSpeech.NET*, 2019. [Online]. Available: https://www.nearlyfreespeech.net/services/hosting. [Accessed: 11-Dec-2019].

"MySQL - NearlyFreeSpeech.net", *NearlyFreeSpeech.NET*, 2019. [Online]. Available: https://www.nearlyfreespeech.net/services/mysql. [Accessed: 11-Dec-2019].

"Language Versions ("2019Q4" Realm)", *NearlyFreeSpeech.NET*, 2019. [Online]. Available: https://2019q4.nfshost.com/. [Accessed: 11-Dec-2019].

"What is JavaScript? - Learn web development | MDN", *Mozilla.org*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Accessed: 11-Dec-2019].

"Learning HTML: Guides and tutorials - Learn web development | MDN", *Mozilla.org*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/HTML. [Accessed: 11-Dec-2019].

"Learn to style HTML using CSS - Learn web development | MDN", *Mozilla.org*, 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/CSS. [Accessed: 11-Dec-2019].

"Single-page application vs. multiple-page application", *Medium.com*, 2019. [Online]. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [Accessed: 11-Dec-2019].

"AngularJS — Superheroic JavaScript MVW Framework", *AngularJS.org*, 2019. [Online]. Available: https://angularjs.org/. [Accessed: 11-Dec-2019].

"Easily add sign-in to your Web app with FirebaseUI | Firebase", *Google.com*, 2019. [Online]. Available: https://firebase.google.com/docs/auth/web/firebaseui. [Accessed: 11-Dec-2019].

"Single Page Application Architecture in AngularJS - Tech Funda", Techfunda.com, 2019. [Online]. Available: http://techfunda.com/howto/966/single-page-application-architecture. [Accessed: 11- Dec- 2019].

4 Glossary

Client - The user-side device or browser that makes requests on the user's behalf to the application server. Also acts as an extension to view-layer code, displaying pages and data to user in proper format.

Controller - Server-side code that receives requests and input from client, processes data as needed, passes it through service layer as needed, and returns data to client.

Model - The layer of our application which contains all of the various classes and objects required to store data and process it within our application. Instances of these models are called, created, and passed between the service layer and controller.

Sanitization - To pass user input through a function that checks that input is given in the expected format and does not contain input that circumvents database or server code, enabling hackers to corrupt or steal data or control application servers. Modifies data to allow it to be compatible for the database and application, when passing between the two. Sanitization occurs within the service layer.

Service Layer - An extra layer for performing functions outside what the controller would normally do such as sanitization, database querying, etc. Controller is typically a "router" of data and requests. Service layer can handle calculations and other processing so routing is not interrupted. The only layer with direct access to database. Controller calls service methods and requests. The database returns query results to controller via the service layer.

View - Client-side code that controls how data and HTML are rendered to the end-user. This code is run in the user's browser and it receives data and pages sent (or "served") by the Controller to be rendered to the user.

5 Viewpoints & Views

5.1 Stakeholders & Design Concerns

5.1.1 Identified Stakeholders

- **Project Sponsor**
  - Joshua Isom
- **Project Manager**
  - Scott Robison
- **Team Leads**
  - Janzen Ferrin
  - Caleb Hensley
  - Nicholas Hendricks
  - Sara Reece
  - Jordon Thompson
  - Adam Tipton
- **Individual Contributors**
  - Todd Anderson
  - Thomas Rollin Burr
  - Koby Sanchez Campbell
  - Whitney Chase
  - Peyton Dunnaway
  - Ben Dzado
  - Jose Jaime Gamero
  - Bryan Heder
  - Austin Kelly
  - Shawn Lilly
  - Scott Malin
  - Jonathan Mancia
  - Tanner Meade
  - Dylan Miessner
  - Paul Owens
  - John Reiley
  - Nathan Rowley
  - Stephen Sharp
  - Jonathan Stutz
  - Tanner Wilson

5.1.2 Identified Design Concerns

Design concerns are expressed in the “Food For All Software Requirements Specification, Revision 1.0” as published 26 October 2019. It can be found online as a Google Doc.

Textual Link: https://docs.google.com/document/d/1nJTxNCfZNIpFUpiT6TwVlHZ6EWA_n_fx1MZ2x1eZ08k/edit?usp=sharing

5.2 Context

5.2.1 Design Concerns

The context viewpoint is described in IEEE Std 1016-2009 on page 14, section 5.2.

The purpose of the Context viewpoint is to identify a design subject's offered services, its actors (users and other interacting stakeholders), to establish the system boundary and to effectively delineate the design subject's scope of use and operation.

Drawing a boundary separating a design subject from its environment, determining a set of services to be provided, and the information flows between design subject and its environment, is typically a key design decision. That makes this viewpoint applicable to most design efforts.

When the system is portrayed as a black box, with internal decisions hidden, the Context view is often a starting point of design, showing what is to be designed functionally as the only available information about the design subject: a name and an associated set of externally identifiable services. Requirements analysis identifies these services with the specification of quality of service attributes, henceforth invoking many non-functional requirements. Frequently incomplete, a Context view is begun in requirements analysis. Work to complete this view continues during design.

5.2.2 Design Elements

Design elements for the Context viewpoint include use case actors, triggers, primary scenarios, alternate scenarios, pre-conditions, and post-conditions. These design elements are elaborated in the use cases provided hereinbelow.

5.2.2.1 System Diagrams

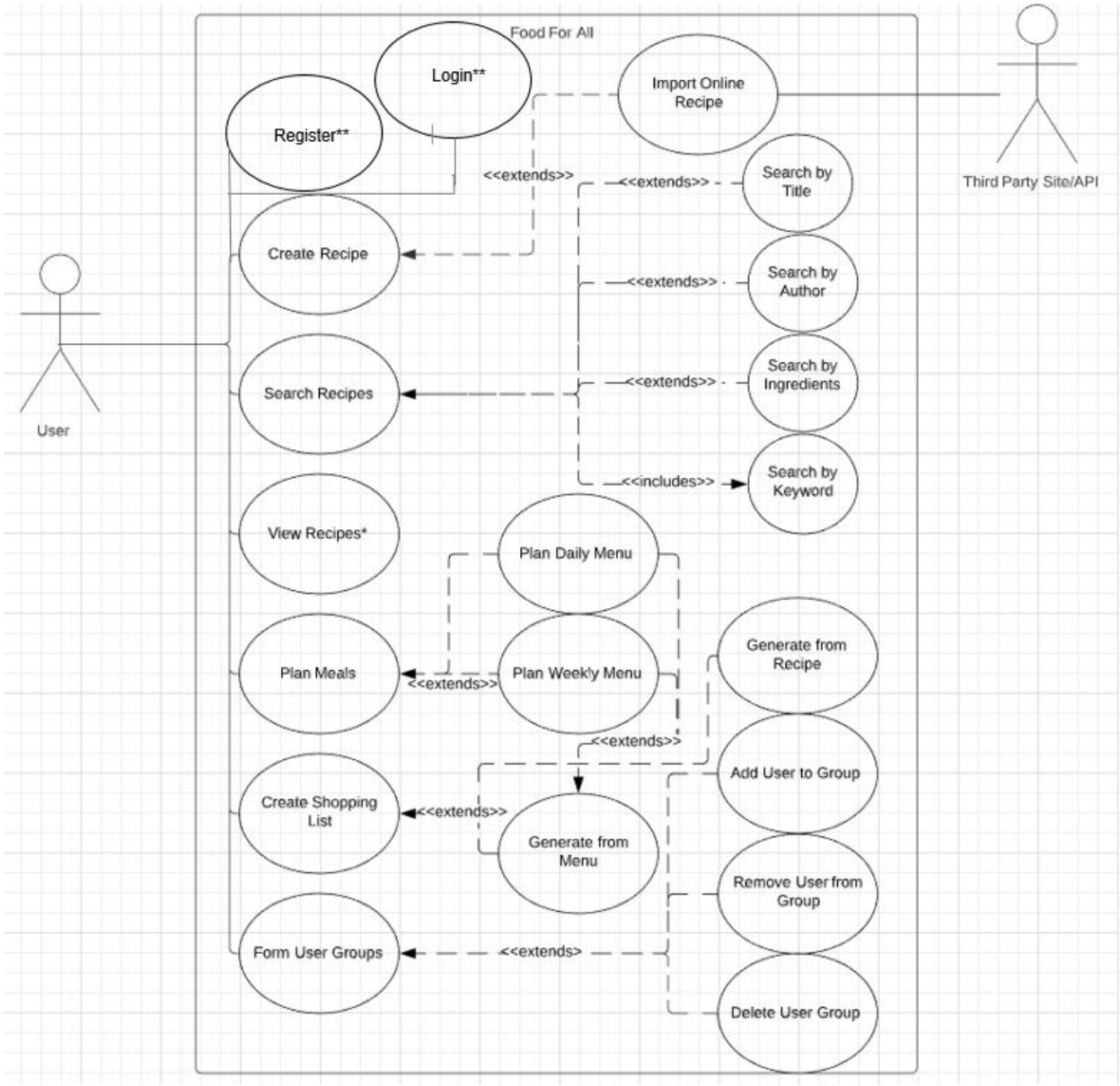


Figure 5.2.2.1.1 Use Case Diagram - System Diagram
**View Recipes extended in Figure 5.2.2.2.*
***Register and Login extended in Figure 5.2.2.3.*

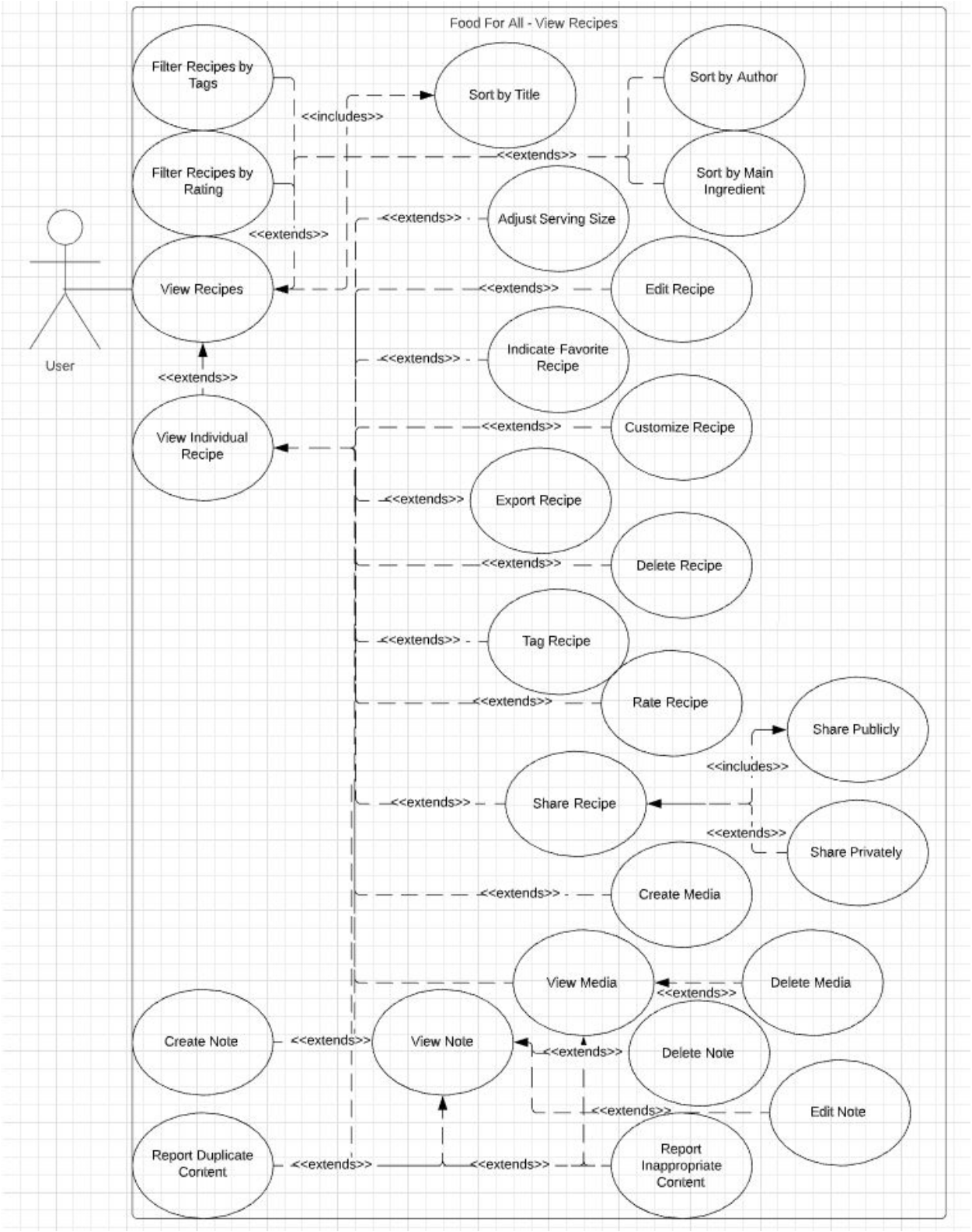


Figure 5.2.2.1.2 Use Case Diagram - View Recipes Diagram Continued

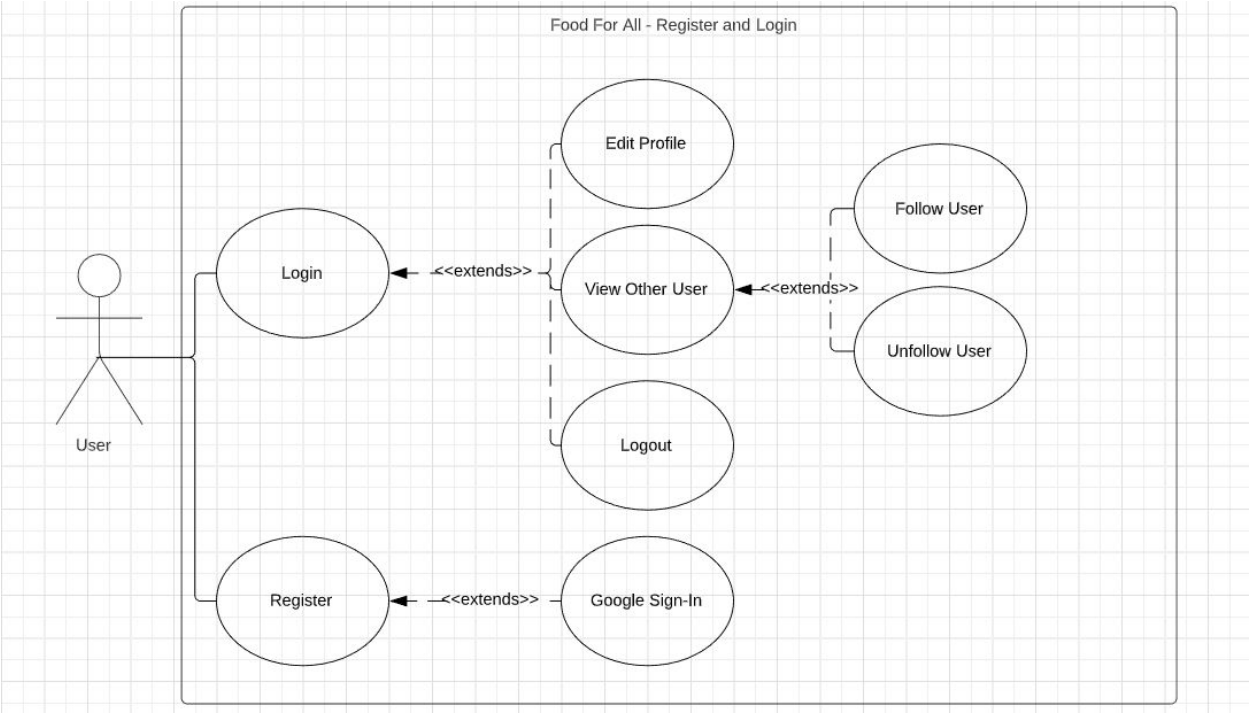


Figure 5.2.2.1.3 Use Case Diagram - Register and Login Continued

5.2.2.2 Create Recipe

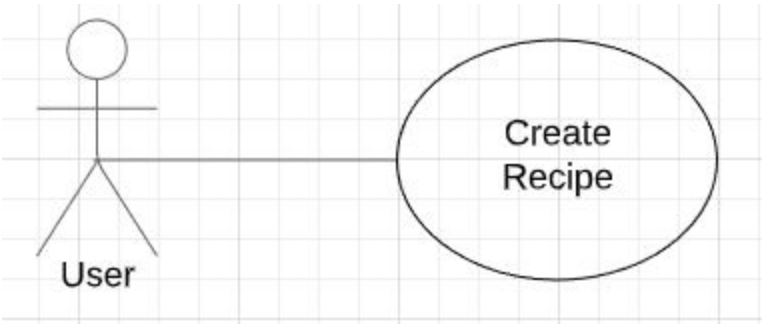


Figure 5.2.2.2.1 Use Case Diagram - Create Recipe

Use Case Number	1
Use Case	Create Recipe
Summary	User can create a new recipe in the application.
Actor	User
Trigger	“Create Recipe” button (See 5.8.2.1.7)
Primary Scenario	After being logged in, the user needs to add a new recipe to FFA via the main screen.
Alternate Scenario	When viewing their list of current recipes, the user would like to add a new recipe.
Exception Scenario	None
Pre-Conditions	Logged in and connected to the application.
Post-Conditions	New recipe is added to the database.
Assumptions	None
SRS Requirement	3.2.3

5.2.2.3 Import Online Recipes

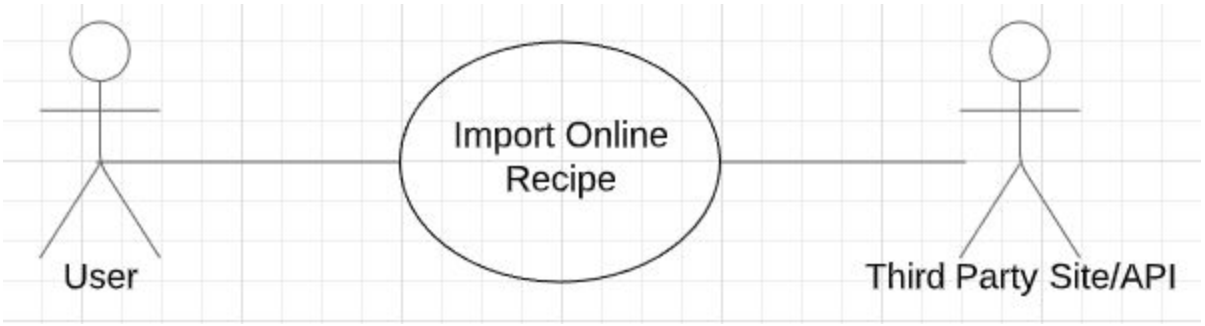


Figure 5.2.2.3.1 Use Case Diagram - Import Online Recipe

Use Case Number	2
Use Case	Import Online Recipes
Summary	User can create a new recipe via importing it from a third party site.
Actor	User and Third Party Site/API
Trigger	“Import Recipe” button on the Create Recipe screen. (See 5.8.2.1.7)
Primary Scenario	User is creating a new recipe and decides to import the recipe which already exists on another site. They provide the URL for the recipes location. FFA then parses the information into the recipe window for the users confirmation.
Alternate Scenario	None
Exceptional Scenario	None
Pre-Conditions	User is in the create recipe window, and has the URL for the recipe to be imported.
Post-Conditions	Recipe information parsed and added to create recipe screen for the users approval.
Assumptions	The URL provided is correctly linked to an existing recipe.
SRS Requirement	3.2.3.2.1

5.2.2.4 Search Recipes

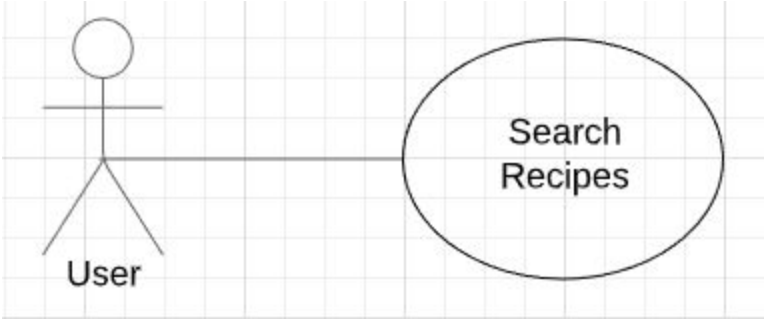


Figure 5.2.2.4.1 Use Case Diagram - Search Recipes

Use Case Number	3
Use Case	Search Recipes
Summary	The user can search for recipes via a search bar provided.
Actor	User
Trigger	Enter submitted after search bar is filled in or search icon selected.
Primary Scenario	User needs to locate a recipe they would like to view, they search a keyword to find it.
Alternate Scenario	The user is unsure if a recipe exists so they attempt to search for it.
Exceptional Scenario	None
Pre-Conditions	User is in the application and inputs a keyword to search.
Post-Conditions	Recipes that match the search are displayed in a list format. If nothing matches, a “No Results Found” page will be displayed.
Assumptions	Supported text is input.
SRS Requirement	3.2.10

5.2.2.5 Search by Title

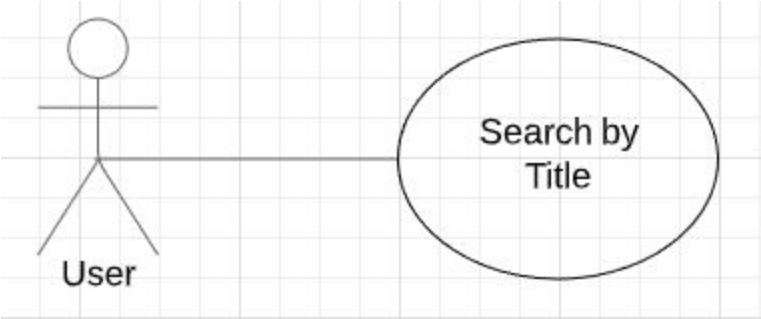


Figure 5.2.2.5.1 Use Case Diagram - Search by Title

Use Case Number	4
Use Case	Search By Title
Summary	The user can search for recipes by title.
Actor	User
Trigger	Search bar is filled and by title is selected.
Primary Scenario	User needs to locate a recipe by title.
Alternate Scenario	The user is unsure if a recipe exists with a certain title
Exceptional Scenario	None
Pre-Conditions	User is in the application and input a title in the search bar, “Search By Title” is selected.
Post-Conditions	Recipes that match the searched title are displayed in a list format. If nothing matches, a “No Results Found” page will be displayed.
Assumptions	Supported text is input.
SRS Requirement	3.2.10.2.1

5.2.2.6 Search by Author

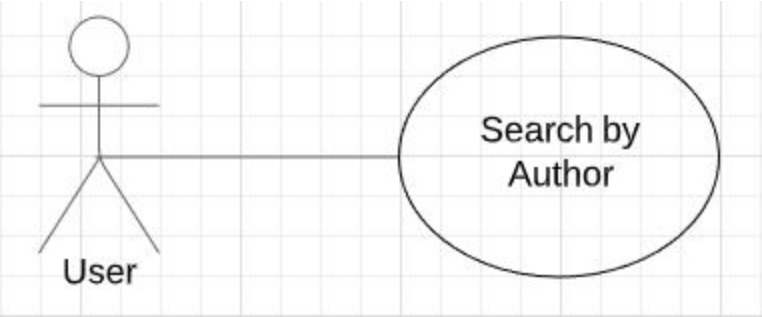


Figure 5.2.2.6.1 Use Case Diagram - Search by Author

Use Case Number	5
Use Case	Search by Author
Summary	The user can search for recipes by title.
Actor	User
Trigger	Search bar is filled and by author is selected.
Primary Scenario	User needs to locate a recipe by author name.
Alternate Scenario	The user is unsure if a recipe exists by a certain author name.
Exceptional Scenario	None
Pre-Conditions	User is in the application and input an author name in the search bar, “Search By Author” is selected.
Post-Conditions	Recipes that match the searched author name are displayed in a list format. If nothing matches, a “No Results Found” page will be displayed.
Assumptions	Supported text is input.
SRS Requirement	3.2.10.2.2

5.2.2.7 Search by Ingredient(s)

Figure 5.2.2.7.1 Use Case Diagram - Search by Ingredient(s)

Use Case Number	6
Use Case	Search by Ingredient(s)
Summary	The user can search for recipes by ingredients.
Actor	User
Trigger	Search bar is filled and by ingredient(s) is selected.
Primary Scenario	User needs to locate a recipe by ingredient(s).
Alternate Scenario	The user is unsure if a recipe exists with certain ingredients.
Exception Scenario	None
Pre-Conditions	User is in the application and input an ingredient or multiple in the search bar, “Search By Ingredient(s)” is selected.
Post-Conditions	Recipes that match the searched ingredient(s) are displayed in a list format. If nothing matches, a “No Results Found” page will be displayed.
Assumptions	Supported text is input.
SRS Requirement	3.2.10.2.3

5.2.2.8 Search by Keyword

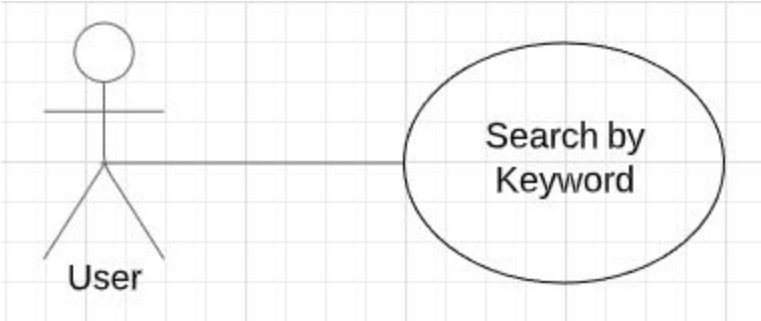


Figure 5.2.2.8.1 Use Case Diagram - Search by Keyword

Use Case Number	7
Use Case	Search by Keyword
Summary	The user can search for recipes by keyword.
Actor	User
Trigger	Search bar is filled search is selected.
Primary Scenario	User needs to locate a recipe by a keyword that can be stored in any part of the recipe.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is in the application and input a keyword in the search bar.
Post-Conditions	Recipes that match the searched keyword are displayed in a list format. If nothing matches, a “No Results Found” page will be displayed.
Assumptions	Supported text is input.
SRS Requirement	3.2.10.2.4

5.2.2.9 View List of Recipes

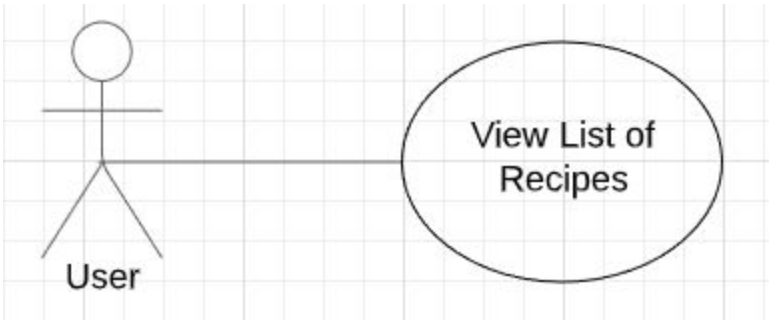


Figure 5.2.2.9.1 Use Case Diagram - View List of Recipes

Use Case Number	8
Use Case	View List of Recipes
Summary	User can view a list of all recipes they have added or searched for.
Actor	User
Trigger	Search display a list, or the user selects “Recipes” and all are displayed.
Primary Scenario	User needs to view recipes, they select the Recipes” button from the main navigation.
Alternate Scenario	User runs a search to view recipes.
Exception Scenario	None
Pre-Conditions	A query is run against the database to get a list of recipes to be displayed.
Post-Conditions	Recipes are displayed that match the query. Default ordering is alphabetically by recipe title.
Assumptions	Recipes exist in the system.
SRS Requirement	3.2.10.1

5.2.2.10 Filter Recipes by Tags

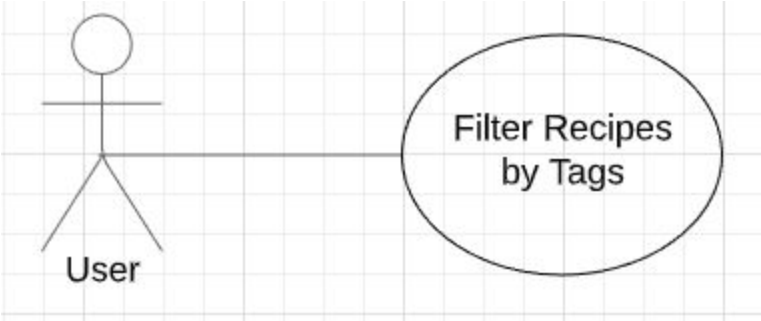


Figure 5.2.2.10.1 Use Case Diagram - Filter Recipes by Tags

Use Case Number	9
Use Case	Filter Recipes by Tags
Summary	User can filter list of recipes by tags.
Actor	User
Trigger	The user selects a tag or multiple tags when viewing a list.
Primary Scenario	The user selects the “Recipes” button from the main navigation, once viewing a list of recipes, the user filters the results by selecting available/existing tags.
Alternate Scenario	User runs a search to view recipes, once viewing a list of recipes, the user filters the results by selecting available/existing tags.
Exception Scenario	None
Pre-Conditions	A query is run against the database to get a list of recipes to be displayed. The results are then filtered based on tags.
Post-Conditions	Recipes are displayed that match the query. Results are then filtered again according to selected tags. Default ordering is alphabetically by recipe title.
Assumptions	Recipes exist in the system. Tags also exist and are applied to recipes.
SRS Requirement	3.2.10.2.6

5.2.2.11 Filter Recipes by Rating

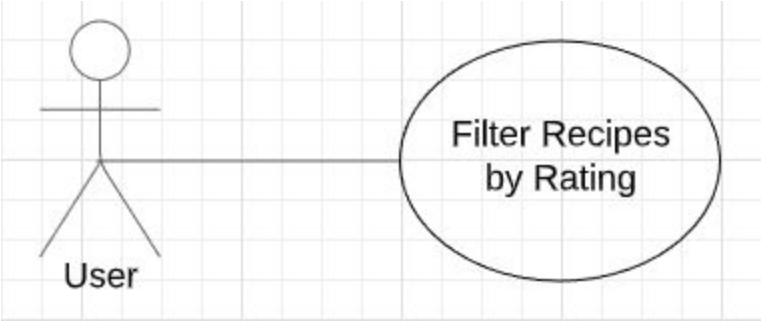


Figure 5.2.2.11.1 Use Case Diagram - Filter Recipes by Rating

Use Case Number	10
Use Case	Filter Recipes by Rating
Summary	User can filter list of recipes by rating.
Actor	User
Trigger	The user selects a rating when viewing a list.
Primary Scenario	The user selects the “Recipes” button from the main navigation, once viewing a list of recipes, the user filters the results by selecting a rating.
Alternate Scenario	User runs a search to view recipes, once viewing a list of recipes, the user filters the results by selecting a rating.
Exceptional Scenario	None
Pre-Conditions	A query is run against the database to get a list of recipes to be displayed. The results are then filtered based on the selected rating.
Post-Conditions	Recipes are displayed that match the query. Results are then filtered again according to the selected rating. Default ordering is alphabetically by recipe title.
Assumptions	Recipes exist in the system. Recipes have been rated.
SRS Requirement	3.2.7.1.3

5.2.2.12 Sort Recipes by Title

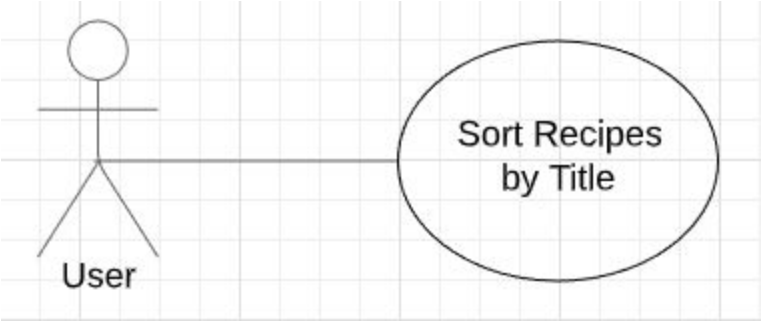


Figure 5.2.2.12.1 Use Case Diagram - Sort Recipes by Title

Use Case Number	11
Use Case	Sort Recipes by Title
Summary	User can sort list of recipes by Title both alphabetically and reverse alphabetically.
Actor	User
Trigger	User select the column header when viewing the list to have it sorted by Title.
Primary Scenario	User is viewing a list of recipes and select the Title column header.
Alternate Scenario	The user is already viewing the list after selecting Title column header, and they select it again.
Exception Scenario	None
Pre-Conditions	User is viewing a list of recipes.
Post-Conditions	First time Title column header is selected, the list sorts alphabetically by Title. Title column is selected a second time, the list reverses, displaying reverse alphabetically by title.
Assumptions	The user has recipes in the application.
SRS Requirement	3.2.10.1

5.2.2.13 Sort Recipes by Author

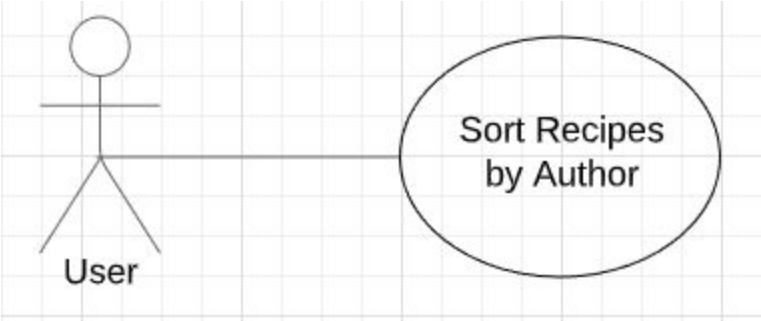


Figure 5.2.2.13.1 Use Case Diagram - Sort Recipes by Author

Use Case Number	12
Use Case	Sort Recipes by Author
Summary	User can sort list of recipes by Author both alphabetically and reverse alphabetically.
Actor	User
Trigger	User selects the column header when viewing the list to have it sorted by Author.
Primary Scenario	User is viewing a list of recipes and selects the Author column header.
Alternate Scenario	The user is already viewing the list after selecting the Author column header, and they select it again.
Exception Scenario	None
Pre-Conditions	User is viewing a list of recipes.
Post-Conditions	First time the Author column header is selected, the list sorts alphabetically by Author. The Author column header is selected a second time, the list reverses, displaying reverse alphabetically by author.
Assumptions	The user has recipes in the application.
SRS Requirement	3.2.10.1

5.2.2.14 Sort Recipes by Main Ingredient

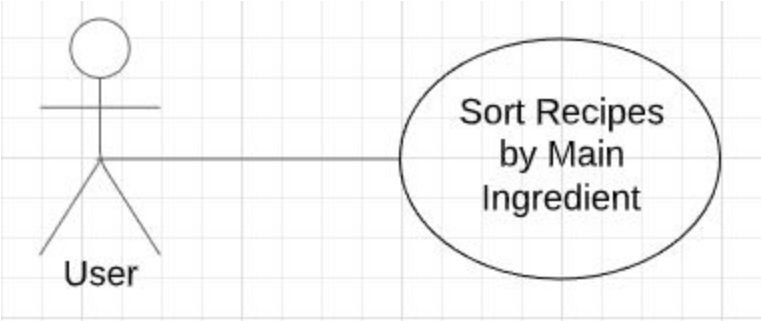


Figure 5.2.2.14.1 Use Case Diagram - Sort Recipes by Main Ingredient

Use Case Number	13
Use Case	Sort Recipes by Main Ingredient
Summary	User can sort list of recipes by Main ingredients both alphabetically and reverse alphabetically.
Actor	User
Trigger	User selects the column header when viewing the list to have it sorted by Main Ingredient.
Primary Scenario	User is viewing a list of recipes and selects the Main Ingredient column header.
Alternate Scenario	The user is already viewing the list after selecting the Main Ingredient column header, and they select it a second time.
Exception Scenario	None
Pre-Conditions	User is viewing a list of recipes.
Post-Conditions	First time the Main Ingredient column header is selected, the list sorts alphabetically. The Main Ingredient column header is selected a second time, the list reverses, displaying reverse alphabetically.
Assumptions	The user has recipes in the application.
SRS Requirement	3.2.10.1

5.2.2.15 View Individual Recipe

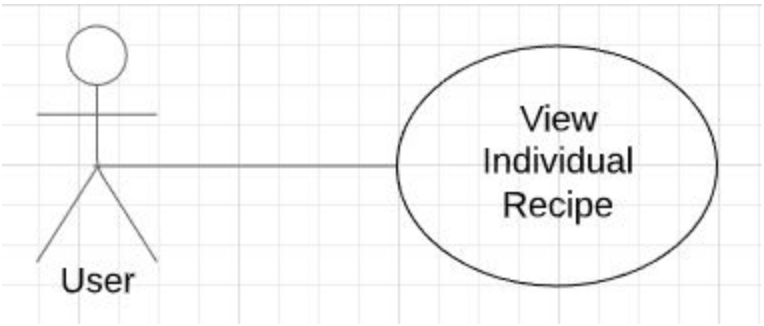


Figure 5.2.2.15.1 Use Case Diagram - View Individual Recipe

Use Case Number	14
Use Case	View Individual Recipe
Summary	User can select a recipe to view it in full detail.
Actor	User
Trigger	User clicks on the recipe name.
Primary Scenario	User is viewing a list of recipes. They click on the recipe title to view it in full.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a list of recipes and clicks on the recipe name.
Post-Conditions	Recipe loads on the screen displaying all of its information.
Assumptions	The Title Link to the recipe is properly connected to an existing recipe.
SRS Requirement	NA

5.2.2.16 Adjust Serving Size

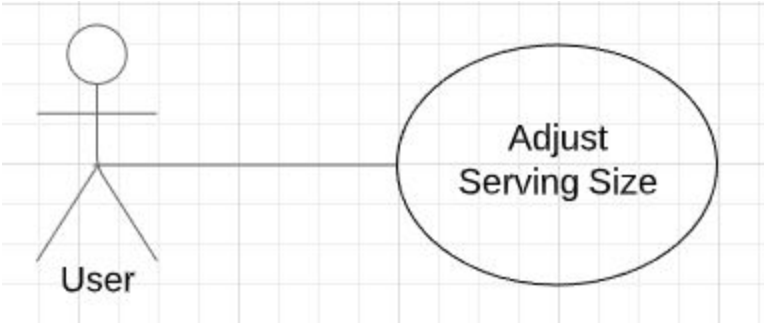


Figure 5.2.2.16.1 Use Case Diagram - Adjust Serving Size

Use Case Number	15
Use Case	Adjust Serving Size
Summary	User can adjust the serving size on the recipe to see adjusted ingredient quantities.
Actor	User
Trigger	Serving Quantity is edited while viewing a recipe.
Primary Scenario	User is viewing a recipe, they edit the Serving Quantity to a higher number.
Alternate Scenario	User is viewing a recipe, they edit the Serving Quantity to a lower number.
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and selects the Serving Quantity field to edit.
Post-Conditions	Recipe ingredient quantities are multiplied or divided by change to the Serving Quantity Field.
Assumptions	The recipe contains numeric ingredient quantities and not incalculable values such as “a pinch”.
SRS Requirement	3.2.8

5.2.2.17 Edit Recipe

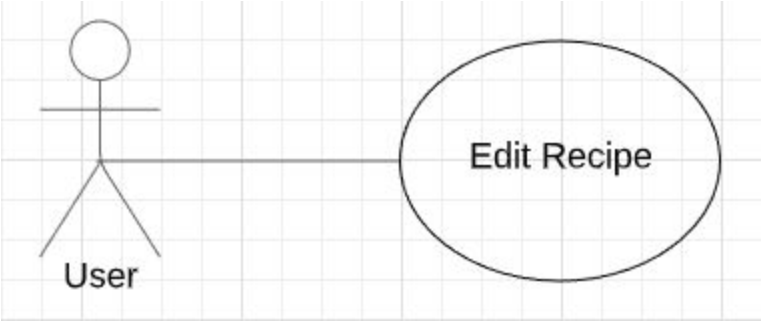


Figure 5.2.2.17.1 Use Case Diagram - Edit Recipe

Use Case Number	16
Use Case	Edit Recipe
Summary	User can make edits and adjustments to recipes they have created.
Actor	User
Trigger	User clicks on the “Edit Recipe” button while viewing a recipe.
Primary Scenario	User is viewing a recipe and wanted to edit an ingredient, quantity, or a direction step.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing an already existing recipe.
Post-Conditions	Once edit is selected, all fields are editable. Upon saving edit, they will reflect when viewing the recipe.
Assumptions	Recipes exists in the system and are editable at the database level.
SRS Requirement	3.2.5.1

5.2.2.18 Indicate Favorite Recipe

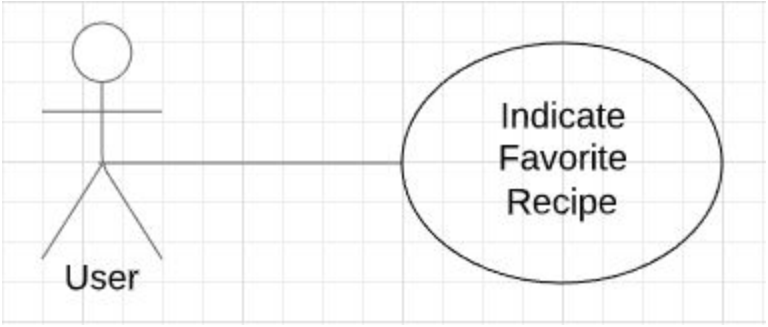


Figure 5.2.2.18.1 Use Case Diagram - Indicate Favorite Recipe

Use Case Number	17
Use Case	Indicate Favorite Recipe
Summary	User can mark recipes as favorite recipes to locate easier.
Actor	User
Trigger	User clicks the “Favorite” icon when viewing the recipe.
Primary Scenario	User wants to mark a recipe as a favorite to have it added to a separate favorite style list for easy access.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and clicks the “Favorite” icon to indicate it is a favorite recipe.
Post-Conditions	Recipe is added to a list of favorite recipes.
Assumptions	Recipe exists.
SRS Requirement	3.1.1.2.1.3

5.2.2.19 Customize Recipe

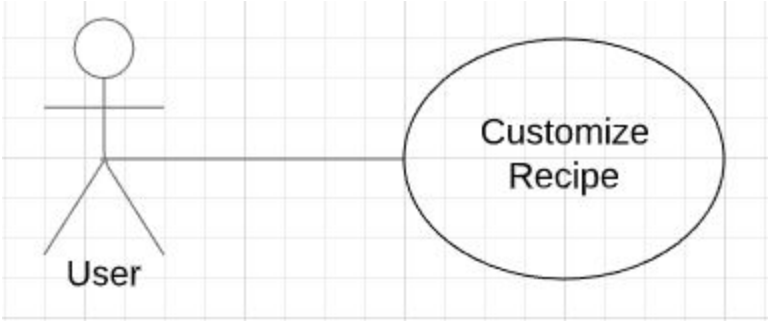


Figure 5.2.2.19.1 Use Case Diagram - Customize Recipe

Use Case Number	18
Use Case	Customize Recipe
Summary	User may want to take an existing recipe and customize it by altering ingredients, or other things outside the scope of simply editing.
Actor	User
Trigger	User clicks “Customize Recipe” while viewing a recipe.
Primary Scenario	While viewing a recipe they own, the user wants to make an alternate version with customizations.
Alternate Scenario	While viewing a public recipe created by another user, the user wants to make a copy and customize it.
Exception Scenario	None
Pre-Conditions	User is viewing a recipe already created in the system.
Post-Conditions	A new recipe is created with the customized details. The original recipe is untouched.
Assumptions	Original recipe exists until the customize recipe button is selected. Once copied it is no longer needed.
SRS Requirement	3.2.5.2

5.2.2.20 Export Recipe

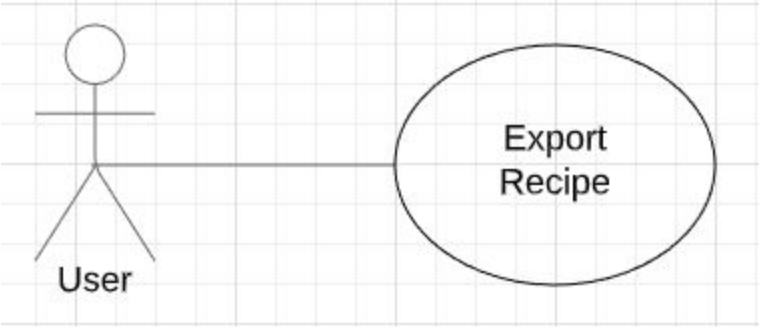


Figure 5.2.2.20.1 Use Case Diagram - Export Recipe

Use Case Number	19
Use Case	Export Recipe
Summary	User wants to export a recipe to a PDF, or other format either to store as a file or print to a physical copy.
Actor	User
Trigger	User selects “Export Recipe” button while viewing a recipe.
Primary Scenario	User wants to get a copy of the recipe to store on their computer for offline use or to print a physical copy.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe they would like to export.
Post-Conditions	User is presented the ability to download as a PDF which can be saved or printed.
Assumptions	The user is able to use PDFs or know how to print from a PDF.
SRS Requirement	3.1.2.1.1

5.2.2.21 Delete Recipe

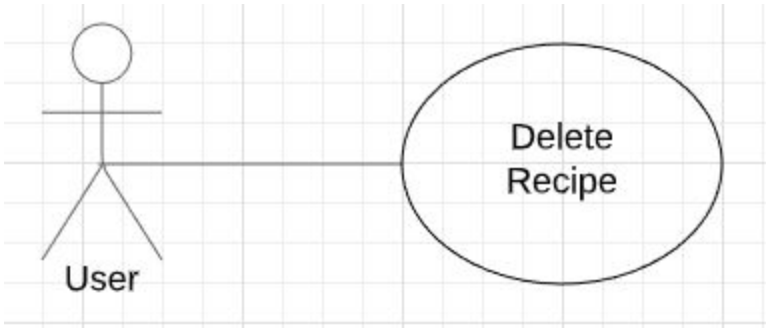


Figure 5.2.2.21.1 Use Case Diagram - Delete Recipe

Use Case Number	20
Use Case	Delete Recipe
Summary	User wants to delete a recipe they created, modified, or imported.
Actor	User
Trigger	User selects “Delete Recipe” button while viewing recipe.
Primary Scenario	User would like to delete a recipe they have created.
Alternate Scenario	User would like to delete a recipe they have customized from another.
Exception Scenario	None
Pre-Conditions	User is viewing a recipe they would like to delete
Post-Conditions	User is prompted about deleting the recipe. If user selects ‘yes’ the recipe is removed, if user selects ‘no’ the prompt closes.
Assumptions	A recipe exists.
SRS Requirement	3.2.5.1

5.2.2.22 Tag Recipe

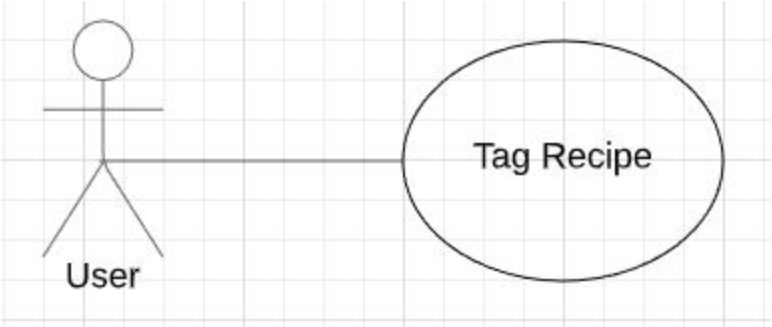


Figure 5.2.2.22.1 Use Case Diagram - Tag Recipe

Use Case Number	21
Use Case	Tag Recipe
Summary	User wants to tag a recipe to categorize in an easy to find way.
Actor	User
Trigger	User types tag name in tag field while viewing recipe, then the user selects “Apply Tag”.
Primary Scenario	User wants to add tag(s) to the recipe currently being viewed
Alternate Scenario	User wants to remove tag(s) from the recipe currently being viewed
Exceptional Scenario	None
Pre-Conditions	User is viewing a recipe they would like to tag
Post-Conditions	Tag(s) the user has specified and added/removed from the recipe
Assumptions	User knows what tags are.
SRS Requirement	3.2.11.5

5.2.2.23 Rate Recipe

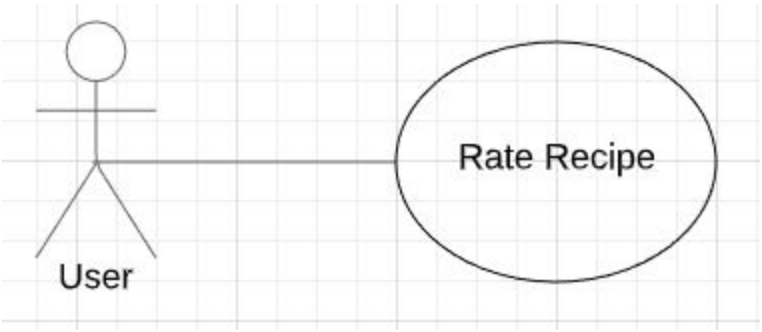


Figure 5.2.2.23.1 Use Case Diagram - Rate Recipe

Use Case Number	22
Use Case	Rate Recipe
Summary	User can rate a recipe between 1-5 stars to indicate their level of satisfaction with the recipe.
Actor	User
Trigger	User selects star rating of 1-5 within rating dropdown field while viewing recipe.
Primary Scenario	User wants to rate the recipe currently being viewed
Alternate Scenario	The app suggests the user should rate the recipe currently being viewed if it has not been rated yet
Exception Scenario	None
Pre-Conditions	User is viewing a recipe they would like to rate.
Post-Conditions	The recipe is rated by the user and the recipe is updated.
Assumptions	User knows it's a 5 star rating system.
SRS Requirement	3.2.7

5.2.2.24 Report Duplicate Content

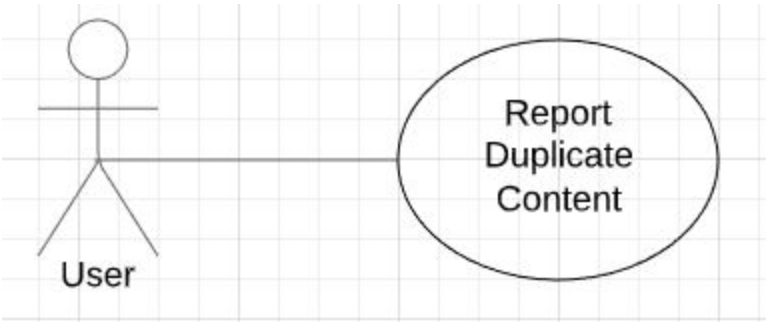


Figure 5.2.2.24.1 Use Case Diagram - Report Duplicate Content

Use Case Number	23
Use Case	Report Duplicate Content
Summary	User can report duplicate recipes to the application.
Actor	User
Trigger	User selects the “Report this Recipe as a Duplicate” button while viewing the recipe.
Primary Scenario	User wants to report the recipe currently being viewed as a duplicate.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe they want to report
Post-Conditions	User is prompted that reporting is the correct action
Assumptions	The recipe can be flagged for duplication
SRS Requirement	3.2.9

5.2.2.25 Report Inappropriate Content

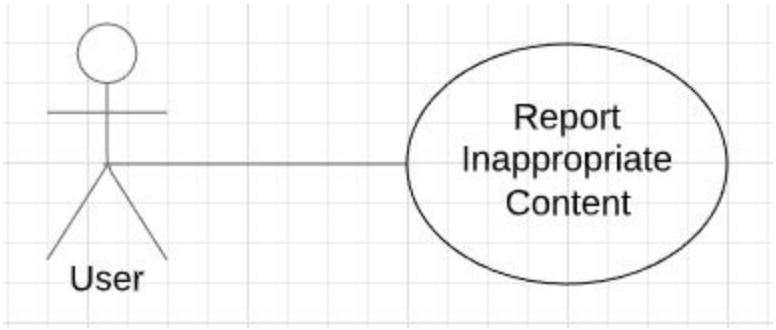


Figure 5.2.2.25.1 Use Case Diagram - Report Inappropriate Content

Use Case Number	24
Use Case	Report Inappropriate Content
Summary	User can report any content that violate Terms of Service.
Actor	User
Trigger	User selects the “Report this Recipe as a Inappropriate” button while viewing the recipe.
Primary Scenario	User wants to report a recipe for inappropriate content.
Alternate Scenario	System suggests that recipe has inappropriate content and asks the user if they would like to report
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and notices inappropriate content.
Post-Conditions	The recipe is flagged for inappropriate content
Assumptions	None
SRS Requirement	3.2.9

5.2.2.26 Share Recipe

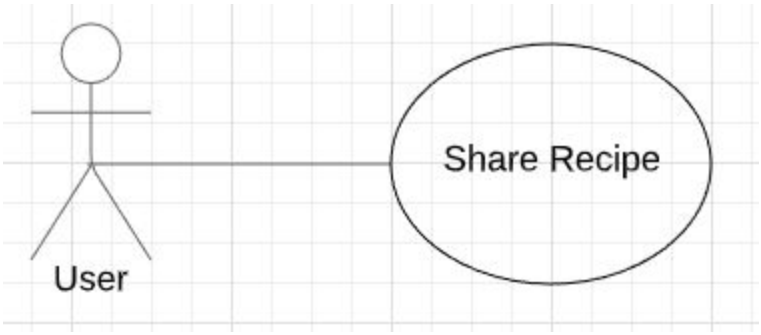


Figure 5.2.2.26.1 Use Case Diagram - Share Recipe

Use Case Number	25
Use Case	Share Recipe
Summary	User wants to be able to share a recipe with other users, either publicly or privately.
Actor	User
Trigger	User selects the “Share” button while viewing a recipe.
Primary Scenario	User wants to share the recipe they are currently viewing.
Alternate Scenario	User wants to export the recipe they are currently viewing
Exception Scenario	None
Pre-Conditions	User wants to share the recipe publicly or privately or wants to export the recipe.
Post-Conditions	User is prompted to share or export the recipe.
Assumptions	None
SRS Requirement	3.1.2.1.1

5.2.2.27 Plan Meals

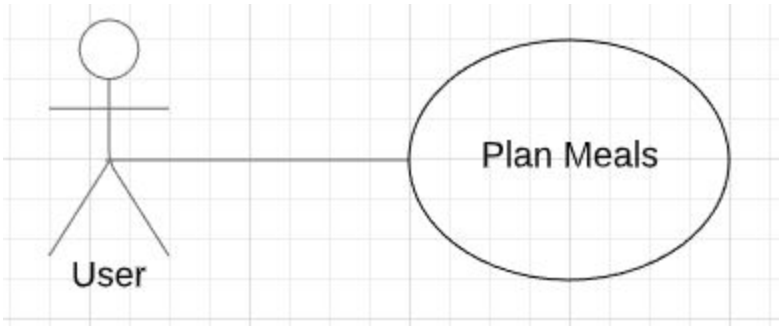


Figure 5.2.2.27.1 Use Case Diagram - Plan Meals

Use Case Number	26
Use Case	Plan Meals
Summary	User can plan meals either by days, weeks, or months.
Actor	User
Trigger	User selects the “Plan Meals” button from the main navigation.
Primary Scenario	User wants to plan meals for the day, the week, or the month.
Alternate Scenario	User wants the ingredients list for a meal plan.
Exception Scenario	None
Pre-Conditions	User is viewing the main navigation of the application.
Post-Conditions	User is taken to the meal planning screen.
Assumptions	The main navigation can be accessed from anywhere in the application.
SRS Requirement	3.2.12

5.2.2.28 Plan Daily Menu

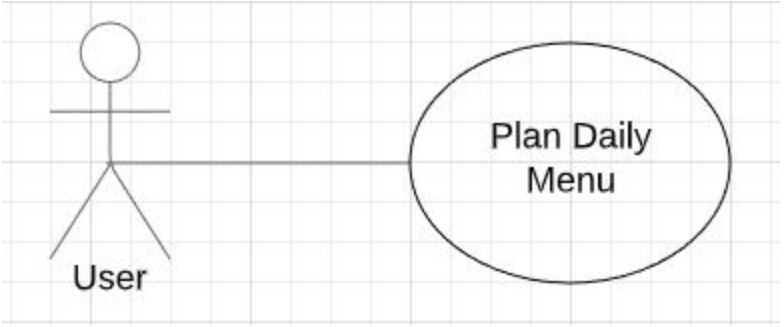


Figure 5.2.2.28.1 Use Case Diagram - Plan Daily Menu

Use Case Number	27
Use Case	Plan Daily Menu
Summary	User want to plan meals for a whole day using recipes in their application.
Actor	User
Trigger	User selects “Plan Daily Meals” while within the Weekly Meal Planning feature or from the Plan Meals feature.
Primary Scenario	User wants to plan meals for a specific day.
Alternate Scenario	User selects a specific day while in the Weekly Meal Planning feature.
Exception Scenario	None
Pre-Conditions	User is viewing the Plan Meals feature or are within the Weekly Meal Planning feature.
Post-Conditions	User is taken to the Daily Meal Planning feature.
Assumptions	None
SRS Requirement	3.2.12.1

5.2.2.29 Plan Weekly Menu

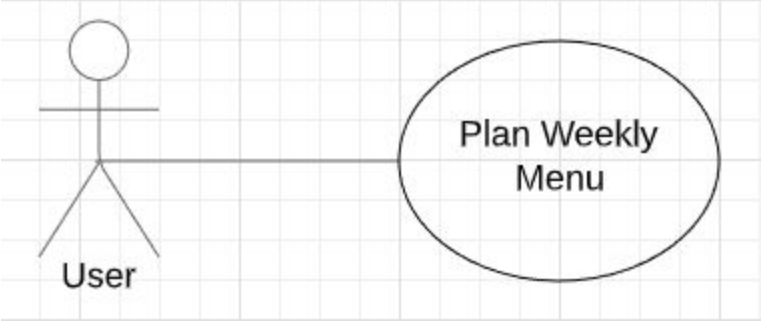


Figure 5.2.2.29.1 Use Case Diagram - Plan Weekly Menu

Use Case Number	28
Use Case	Plan Weekly Menu
Summary	User can plan a week's worth of meals using recipes they have access to in their application.
Actor	User
Trigger	User selects “Plan Weekly Menu” from within the Plan Meals feature.
Primary Scenario	User wants to plan meals for a specific week.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is within the Meal planning feature of the application.
Post-Conditions	User is taken to the Weekly Menu view.
Assumptions	None
SRS Requirement	3.2.12.1

5.2.2.30 Create Media

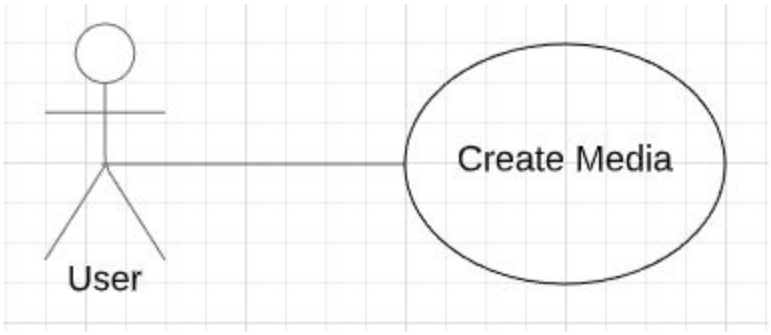


Figure 5.2.2.30.1 Use Case Diagram - Create Media

Use Case Number	29
Use Case	Create Media
Summary	User can create and add photos or videos related to the recipe.
Actor	User
Trigger	User selects “Create Media” while viewing a recipe.
Primary Scenario	User wants to add media to the recipe currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and selects the ‘Create Media’ button.
Post-Conditions	User has the ability to add photos or video links to the recipe.
Assumptions	None
SRS Requirement	3.2.11.6.1

5.2.2.31 View Media

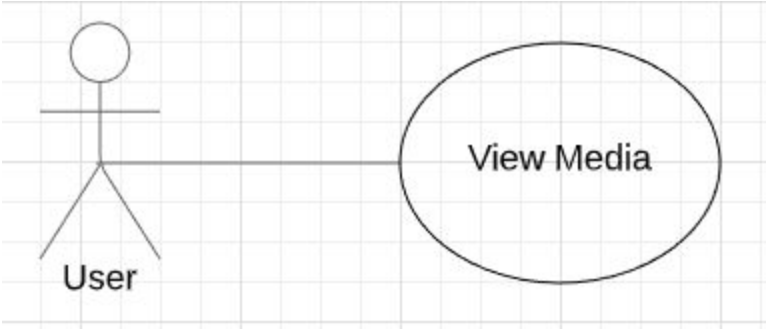


Figure 5.2.2.31.1 Use Case Diagram - View Media

Use Case Number	30
Use Case	View Media
Summary	User can view previously created media that are attached to a recipe.
Actor	User
Trigger	User clicks on the media from a list of attached media options on that recipe.
Primary Scenario	User wants to view the media that is attached to the recipe currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and recipe has attached media.
Post-Conditions	User can now view attached media.
Assumptions	The recipe has media attached to it.
SRS Requirement	3.2.11.4

5.2.2.32 Delete Media

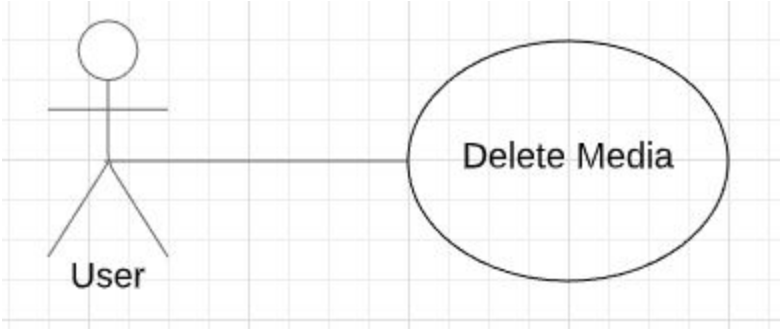


Figure 5.2.2.32.1 Use Case Diagram - Delete Media

Use Case Number	31
Use Case	Delete Media
Summary	User can delete media they have created and attached to recipes.
Actor	User
Trigger	User selects “Delete” while viewing media.
Primary Scenario	User wants to delete the media that is attached to the recipe currently being viewed.
Alternate Scenario	User is viewing media attached to the recipe and selects ‘delete’
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and recipe has attached media.
Post-Conditions	User is prompted to delete selected media.
Assumptions	The recipe has media attached to it.
SRS Requirement	3.2.11.6.2

5.2.2.33 Create Note

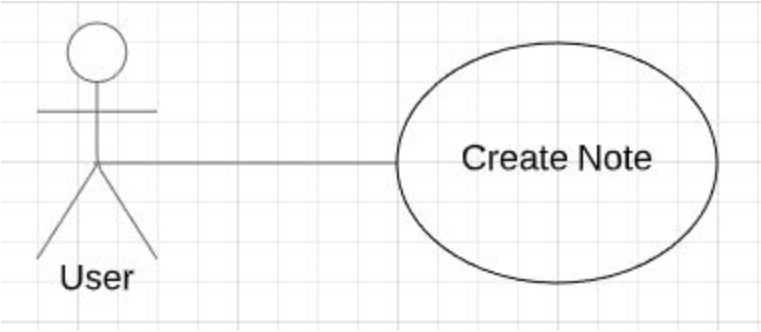


Figure 5.2.2.33.1 Use Case Diagram - Create Note

Use Case Number	32
Use Case	Create Note
Summary	User can create a note and attach it to a recipe to provide additional insights or opinions.
Actor	User
Trigger	User selects “Create Note” while viewing a recipe.
Primary Scenario	User wants to create a note for the recipe currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and wants to create a note.
Post-Conditions	User can now type to create a note on the recipe.
Assumptions	There are no notes created yet.
SRS Requirement	3.2.11.4.1

5.2.2.34 View Note

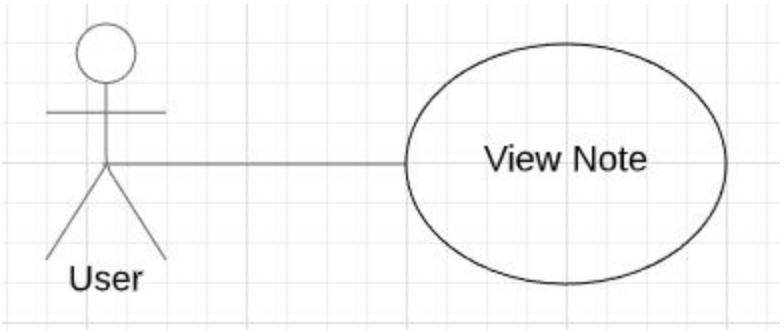


Figure 5.2.2.34.1 Use Case Diagram - View Note

Use Case Number	33
Use Case	View Note
Summary	User can view previously created notes that are attached to a recipe.
Actor	User
Trigger	User clicks on the note from a list of attached notes on that recipe while viewing a recipe.
Primary Scenario	User wants to view a note for the recipe currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and wants to view a note attached to the recipe.
Post-Conditions	User can now see the note on the recipe.
Assumptions	There are notes created for the recipe.
SRS Requirement	3.2.11.4

5.2.2.35 Edit Note

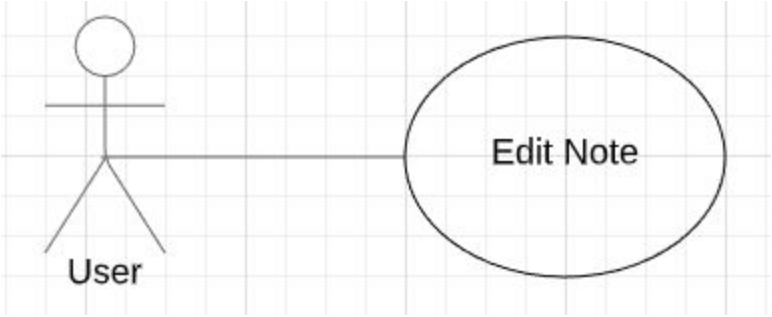


Figure 5.2.2.35.1 Use Case Diagram - Edit Note

Use Case Number	34
Use Case	Edit Note
Summary	User can edit a note they have previously added to a recipe.
Actor	User
Trigger	While viewing the note, user selects “Edit” to make changes.
Primary Scenario	User wants to edit a note for the recipe currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and wants to edit a note attached to the recipe.
Post-Conditions	User can now see and edit the text of the note on the recipe.
Assumptions	There are notes created for the recipe.
SRS Requirement	3.2.11.4.2

5.2.2.36 Delete Note

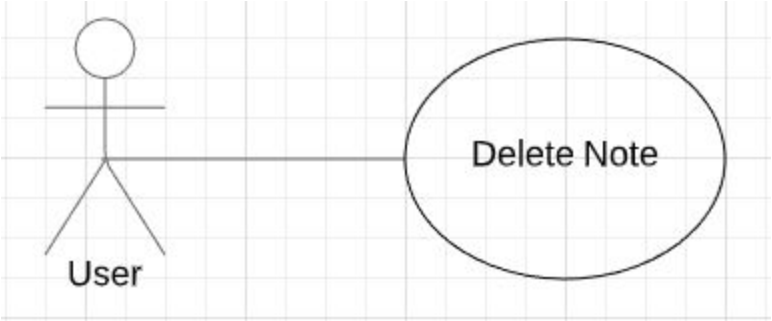


Figure 5.2.2.36.1 Use Case Diagram - Delete Note

Use Case Number	35
Use Case	Delete Note
Summary	User can delete a note they have created from a recipe.
Actor	User
Trigger	While viewing a note, the user selects “Delete”.
Primary Scenario	User wants to delete a note for the recipe currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe and wants to delete a note attached to the recipe.
Post-Conditions	User can now see the note on the recipe and is prompted to delete it.
Assumptions	There are notes created for the recipe.
SRS Requirement	3.2.11.4.3

5.2.2.37 Create Shopping List

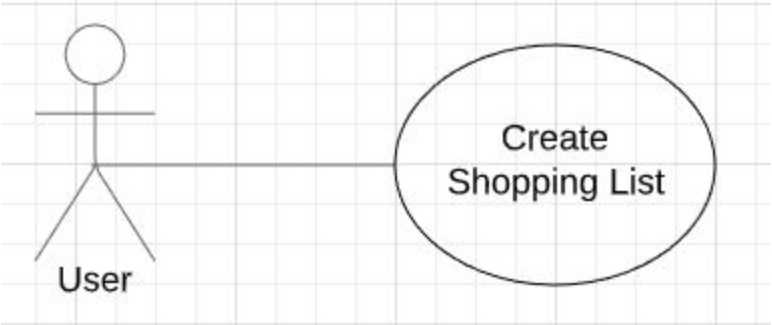


Figure 5.2.2.37.1 Use Case Diagram - Create Shopping List

Use Case Number	36
Use Case	Create Shopping List
Summary	User can create a shopping list in the application for their next shopping trip.
Actor	User
Trigger	User selects “Create Shopping List” from main navigation.
Primary Scenario	User wants to create a new shopping list.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing main navigation of the application.
Post-Conditions	User is taken to the shopping list feature.
Assumptions	The main navigation is viewable from anywhere in the application.
SRS Requirement	3.2.13

5.2.2.38 Generate Shopping List From Recipe

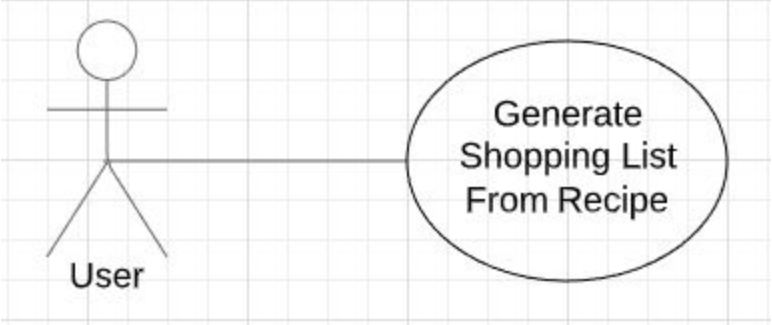


Figure 5.2.2.38.1 Use Case Diagram - Generate Shopping List From Recipe

Use Case Number	37
Use Case	Generate Shopping List From Recipe
Summary	User can add recipes to a shopping list to auto generate a list.
Actor	User
Trigger	User select “Add Recipe” from within the Shopping List feature or “Add to Shopping List” while viewing a recipe.
Primary Scenario	User wants to generate a shopping list based on a recipe that is currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a recipe.
Post-Conditions	User is taken to a shopping list with ingredients of the viewed recipe prefilled.
Assumptions	The recipe has ingredients.
SRS Requirement	3.2.13.1

5.2.2.39 Generate Shopping List From Menu

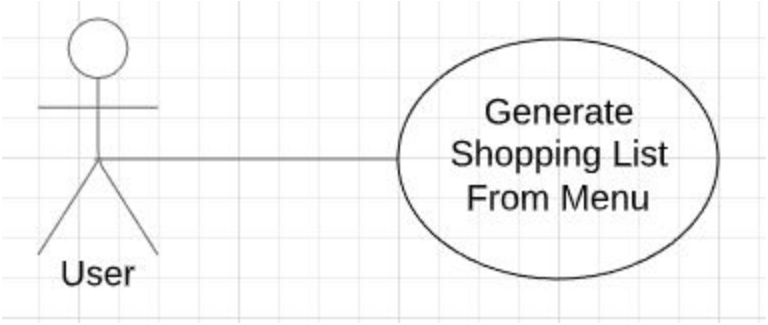


Figure 5.2.2.39.1 Use Case Diagram - Generate Shopping List From Menu

Use Case Number	38
Use Case	Generate Shopping List From Menu
Summary	User can add a menu to a shopping list to auto generate a list.
Actor	User
Trigger	User select “Add Menu” from within the Shopping List feature or “Add to Shopping List” while viewing a menu, either weekly or daily.
Primary Scenario	User wants to create a shopping list based on the daily menu or weekly menu currently being viewed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is viewing a weekly menu or daily menu.
Post-Conditions	User is taken to a shopping list prefilled with the ingredients from the week or day being viewed.
Assumptions	There are weekly and/or daily plans created.
SRS Requirement	3.2.13.2

5.2.2.40 Form User Group

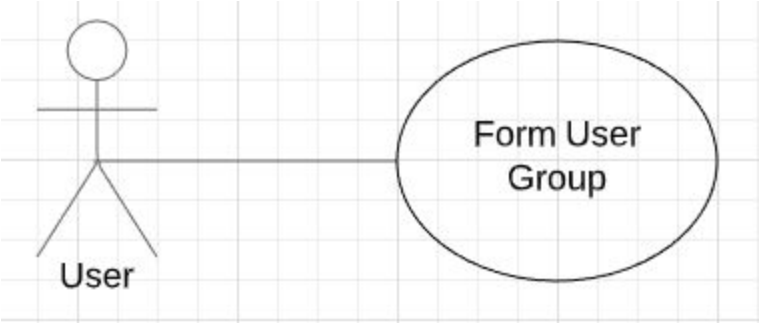


Figure 5.2.2.40.1 Use Case Diagram - Form User Group

Use Case Number	39
Use Case	Form User Group
Summary	User can create a group to share recipes with.
Actor	User
Trigger	User selects “Create User Group” from main navigation.
Primary Scenario	User wants to form a group of users to share recipes with among.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is logged in and in the User Groups sections of the application.
Post-Conditions	User group is formed, allowing those added to see it.
Assumptions	Only users in the group can see it. Any users the admin attempts to add exist as users in the system.
SRS Requirement	3.2.2

5.2.2.41 Add User to Group

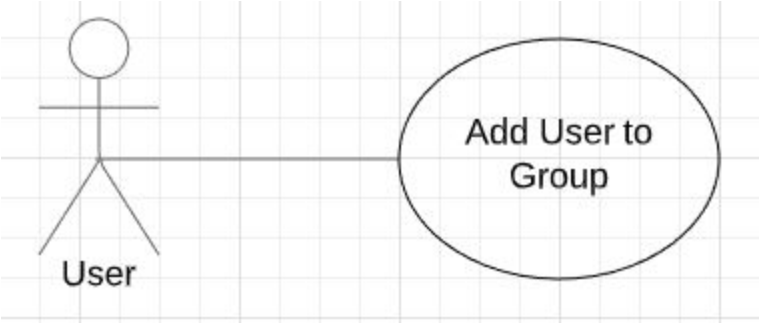


Figure 5.2.2.41.1 Use Case Diagram - Add User to Group

Use Case Number	40
Use Case	Add User to Group
Summary	User can add other users to their group they are the owner of.
Actor	User
Trigger	User selects “Add User” from within a group they have created.
Primary Scenario	Admin of a user group wants to add new users to a group they are a part of.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	Admin of user group is viewing the user group they wish to add users to.
Post-Conditions	Upon adding the user, the group becomes visible to the newly added user. The newly added user is added to the list of members.
Assumptions	The admin has correct permissions in the user group to add users. The user group exists. The users attempted to be added exist.
SRS Requirement	3.2.2.3.1

5.2.2.42 Remove User from Group

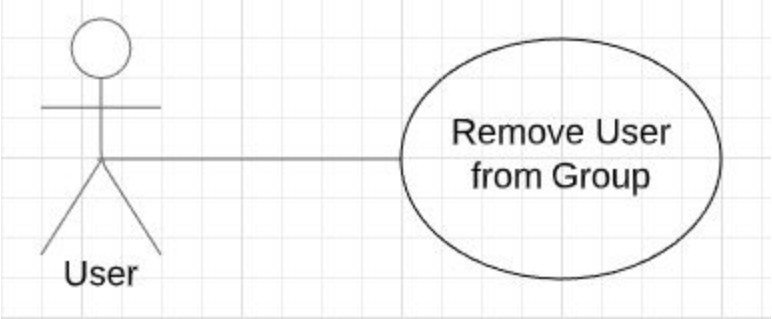


Figure 5.2.2.42.1 Use Case Diagram - Remove User from Group

Use Case Number	41
Use Case	Form User Group
Summary	User can remove other users from their group they are the owner of.
Actor	User
Trigger	User selects “Remove User” from within a group they have created.
Primary Scenario	Admin of a user group wants to remove users from a group they are a part of.
Alternate Scenario	None
Exceptional Scenario	None
Pre-Conditions	Admin of user group is viewing the user group they wish to remove users from.
Post-Conditions	Upon removing the user, the groups visibility is removed from the user. The removed user is removed from the list of members visible within the user group.
Assumptions	The admin has correct permissions in the user group to remove users. The user group exists. The users attempted to be removed exists as a current member.
SRS Requirement	3.2.2.3.1

5.2.2.43 Delete User Group

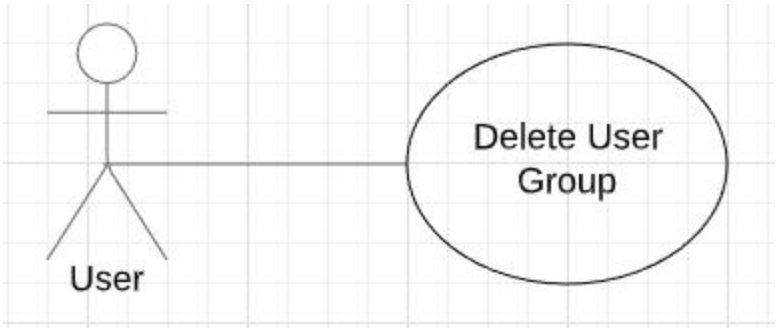


Figure 5.2.2.43.1 Use Case Diagram - Delete User Group

Use Case Number	42
Use Case	Delete User Group
Summary	User can delete a group they have created.
Actor	User
Trigger	User selects “Delete User Group” while viewing a user group they have created.
Primary Scenario	User wants to delete a user group they have formed or are and admin of.
Alternate Scenario	None
Exceptional Scenario	None
Pre-Conditions	User is logged in and viewing the User Groups they want to delete.
Post-Conditions	User group is deleted from the system, removing it from existing user group members view and the system.
Assumptions	User group exists. The user deleting the user group has the correct permission or is the original creator.
SRS Requirement	3.2.2.3.1

5.2.2.44 Register

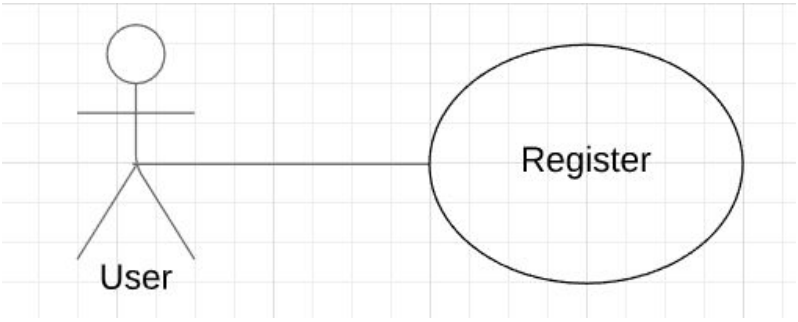


Figure 5.2.2.44.1 Use Case Diagram - Register

Use Case Number	43
Use Case	Register
Summary	User can register to use the application.
Actor	User
Trigger	User selects “Register”.
Primary Scenario	User would like to create a profile in Food For All, they click on the register button within the sign in page.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is on the sign in page, select the register button.
Post-Conditions	A user profile is generated for them using the basic information requested in the register portion.
Assumptions	User inputs the correct information in the correct format.
SRS Requirement	NA

5.2.2.45 Sign-In with Google

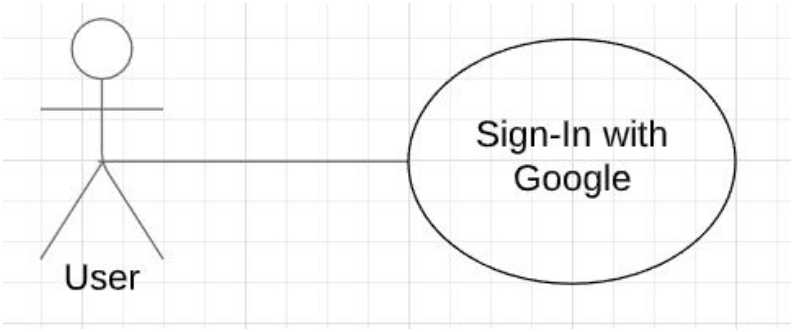


Figure 5.2.2.45.1 Use Case Diagram - Sign-In with Google

Use Case Number	44
Use Case	Sign-In with Google
Summary	User can register to use the application using Google's authentication and sign-in service.
Actor	User
Trigger	User selects “Sign-In with Google”.
Primary Scenario	User is attempting to register for Food For All, in the process they have the option to select “Sign-In with Google” which will use their Google account to authenticate them.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	The user is on the login page and has selected register.
Post-Conditions	A profile is generated for them using the default information provided by Google.
Assumptions	User has a Google account to authenticate with.
SRS Requirement	NA

5.2.2.46 Login

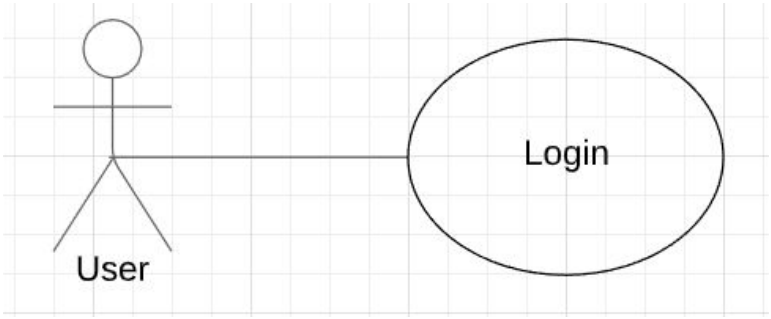


Figure 5.2.2.46.1 Use Case Diagram - Login

Use Case Number	45
Use Case	Login
Summary	User can login to view the application.
Actor	User
Trigger	User selects “Login”.
Primary Scenario	User would like to login to use the application.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is on the login page.
Post-Conditions	If login is successful, they are routed to the homepage.
Assumptions	Credentials are correct and exist in the system.
SRS Requirement	NA

5.2.2.47 Log Out

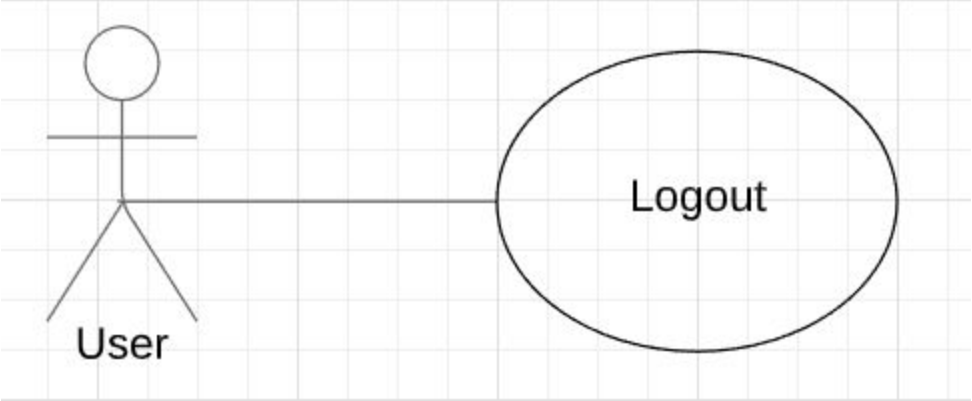


Figure 5.2.2.47.1 Use Case Diagram - Log Out

Use Case Number	46
Use Case	Log Out
Summary	User can log out of the application.
Actor	User
Trigger	User selects “Log Out”.
Primary Scenario	The user would like to log out, ending their session.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	The user is logged in and clicks the “Log Out” button in the navigation.
Post-Conditions	The user is logged out and their session is terminated.
Assumptions	The user is already logged in.
SRS Requirement	NA

5.2.2.48 Edit Profile

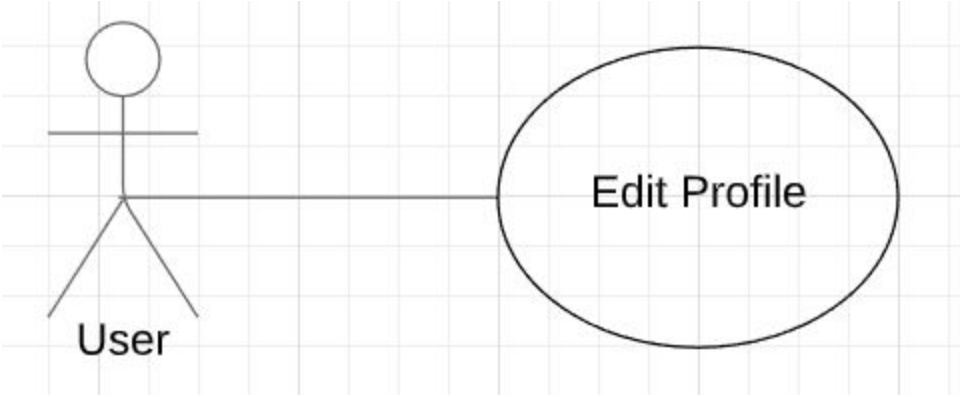


Figure 5.2.2.48.1 Use Case Diagram - Edit Profile

Use Case Number	47
Use Case	Edit Profile
Summary	User can edit their profile to update information.
Actor	User
Trigger	User selects “Edit Profile” from within the main navigation.
Primary Scenario	User would like to update their information.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	The user is logged in and navigated to the “Edit Profile” button.
Post-Conditions	The profile with editable fields is displayed for the user to make edits and save those changes.
Assumptions	The user is already logged in.
SRS Requirement	None

5.2.2.49 View Other User

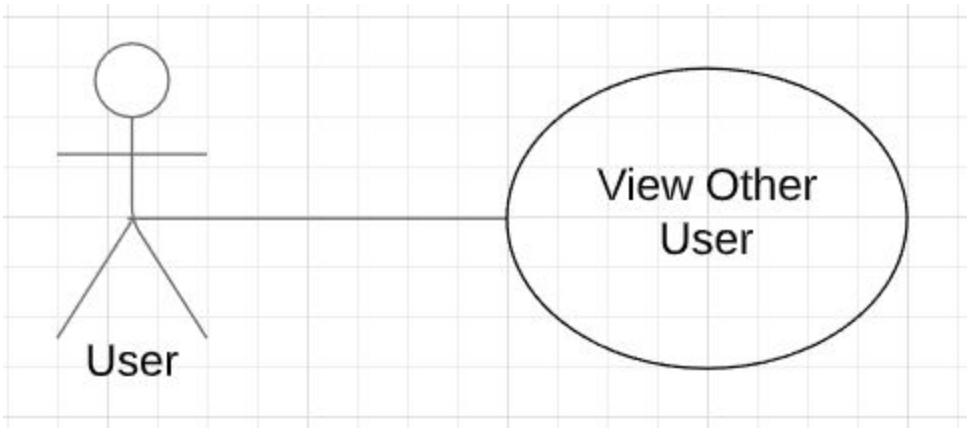


Figure 5.2.2.49.1 Use Case Diagram - View Other User

Use Case Number	48
Use Case	View Other User
Summary	User can view other users profiles, being able to see publicly shared recipes.
Actor	User
Trigger	User clicks on users name from their follow list, or recipe author.
Primary Scenario	User would like to view another user's profile for any number of reasons.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User can see the destination user's name, and click on the name.
Post-Conditions	Destination users profile is presented with public information and publicly shared recipes descending with most recent first.
Assumptions	Username selected is still an existing user in the system.
SRS Requirement	NA

5.2.2.50 Follow User

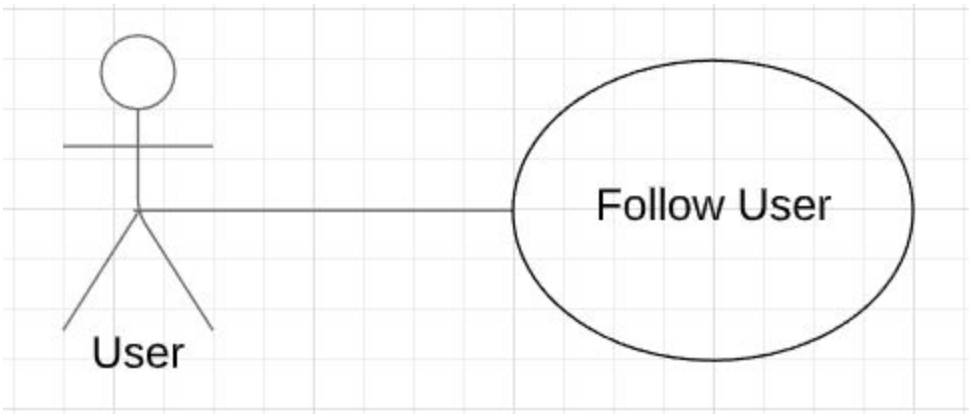


Figure 5.2.2.50.1 Use Case Diagram - Follow User

Use Case Number	49
Use Case	Follow User
Summary	User can follow another user.
Actor	User
Trigger	User selects “Follow” button.
Primary Scenario	User would like to follow another user to get updates when they post or share new recipes.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	User is on the profile page of the user they would like to follow.
Post-Conditions	The user followed is added to a list of users that are being followed for the signed in user.
Assumptions	The user is signed in, and they are on another user's profile other than their own.
SRS Requirement	NA

5.2.2.51 Unfollow User

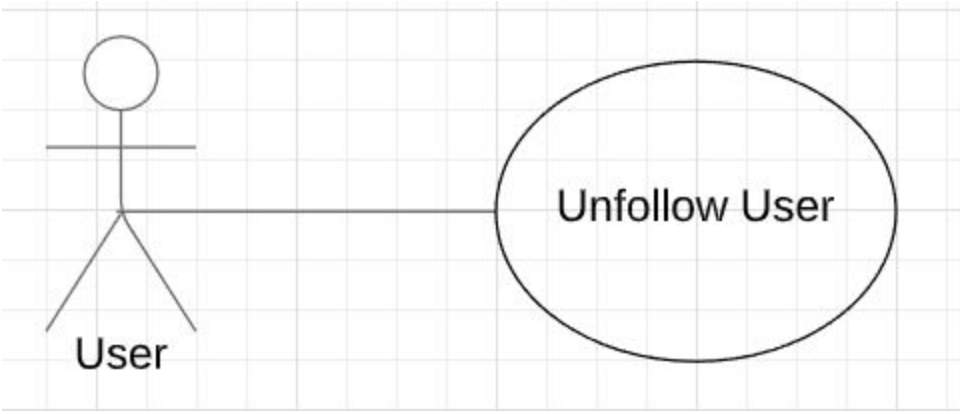


Figure 5.2.2.51.1 Use Case Diagram - Unfollow User

Use Case Number	50
Use Case	Unfollow User
Summary	User can unfollow a user.
Actor	User
Trigger	User selects “Unfollow” button.
Primary Scenario	Logged in user would like to unfollow a user they previously followed.
Alternate Scenario	None
Exception Scenario	None
Pre-Conditions	Logged in user is viewing a list of users they are currently following, or viewing a profile page of a user they follow.
Post-Conditions	The followed user is removed from the list of users logged in user is following.
Assumptions	User is logged in and already following the user they are attempting to unfollow.
SRS Requirement	NA

5.3 Composition

The composition viewpoint is described in IEEE Std 1016-2009 on pages 15-16, section 5.3: “The Composition viewpoint describes the way the design subject is (recursively) structured into constituent parts and establishes the roles of those parts.” In this section, the UML Package Diagram language and UML Component Diagram Language is used to provide an overview of the entirety of the Food For All web application to provide a clear context of the design elements.

Because the diagrams listed below are all drawn from the same references a special traceability section has been created in 5.3.5 to address the traceability of all diagrams.

5.3.1 High Level Application Package Composition

The following figure diagrams the high level package composition of the client and server portions of Food For All, including interfaces used directly by the application specific components.

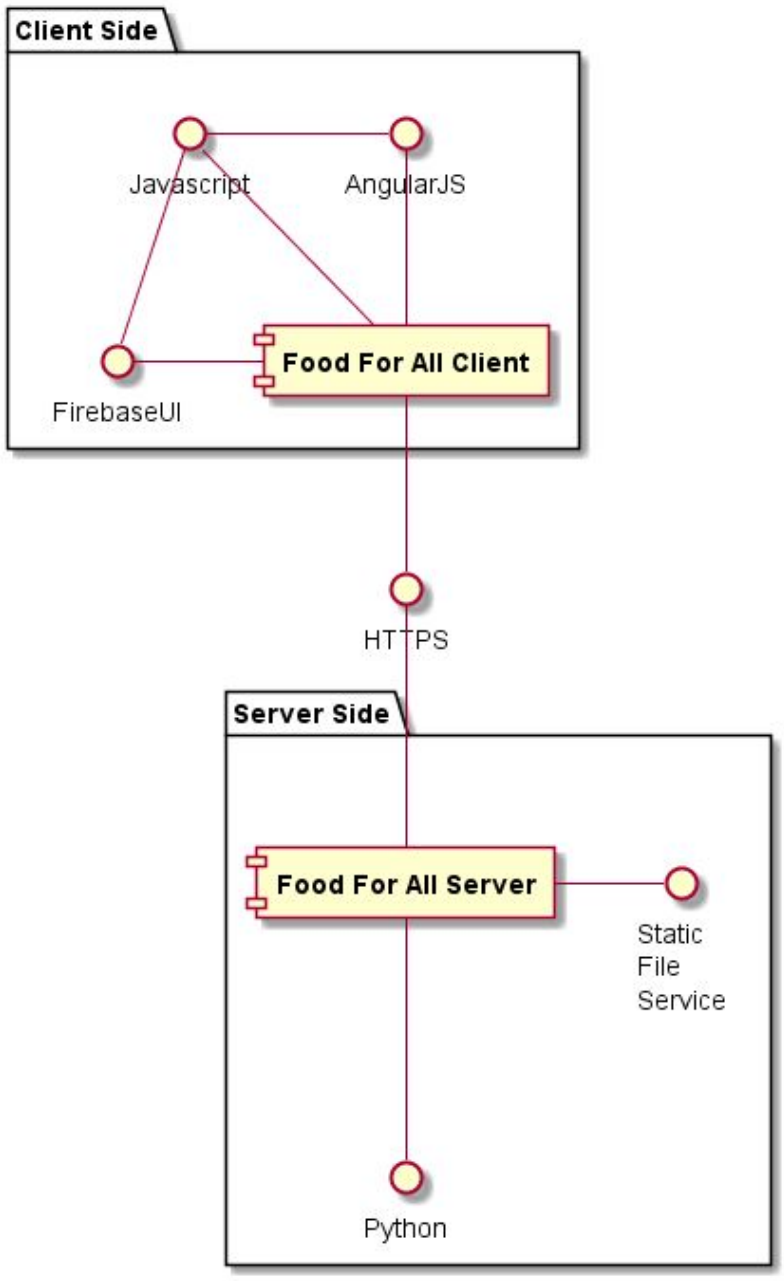


Figure 5.3.1: High level package composition of Food For All.

5.3.2 Client Side Component Elaboration

The following figure elaborates further the indirect interfaces (marked with bold text) that are dependencies of the Food For All client component.

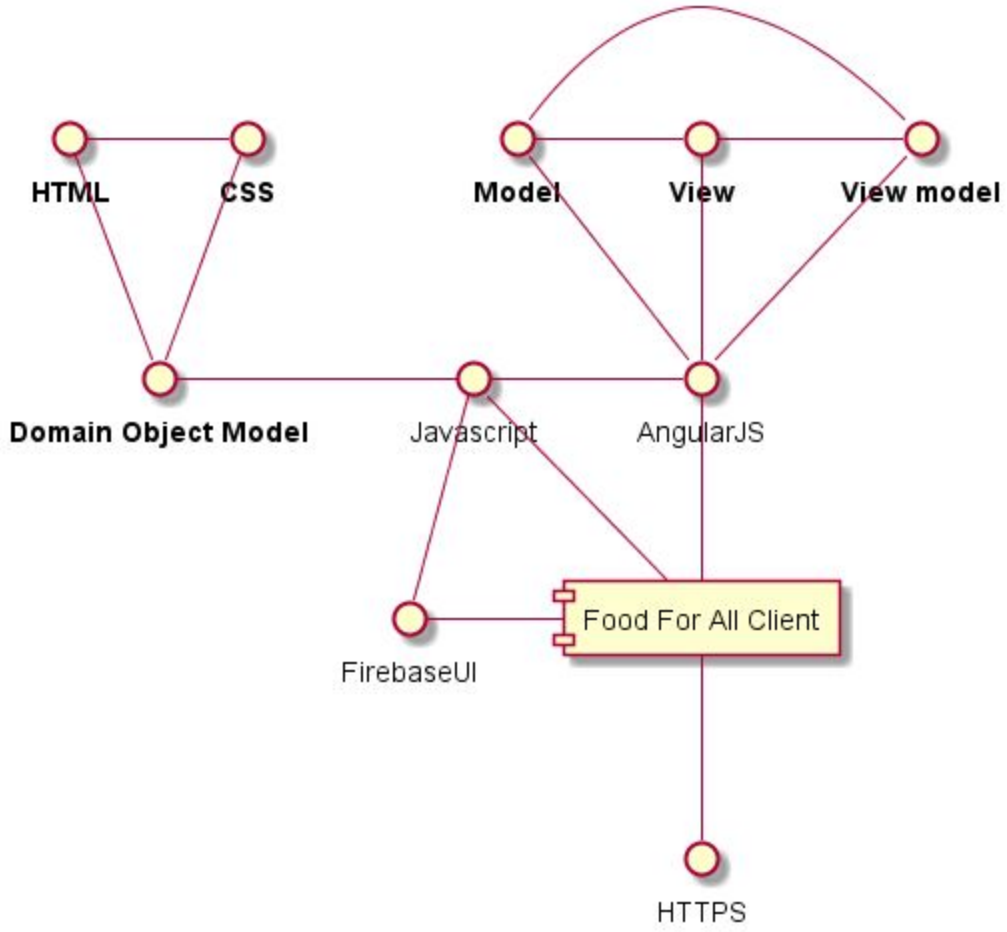


Figure 5.3.2: Elaborated interface dependencies of the Food For All client component.

5.3.3 Server Side Component Elaboration

The following figure elaborates further the indirect interfaces (marked with bold text) that are dependencies of the Food For All server component.

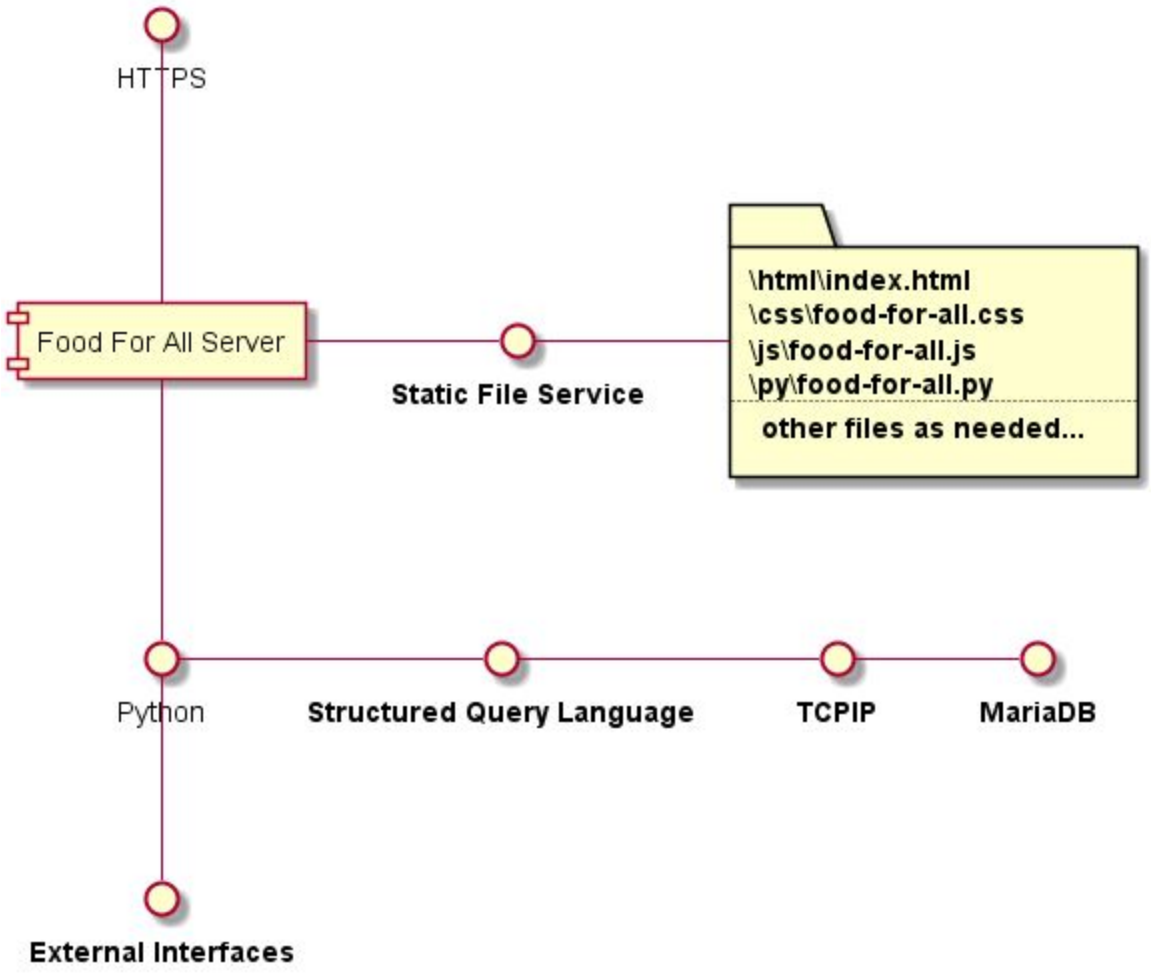


Figure 5.3.3: Elaborated interface dependencies of the Food For All server component.

5.3.4 External Interfaces Elaboration

The following figure elaborates further the external interfaces (marked with bold text) that are dependencies of the Food For All server component.

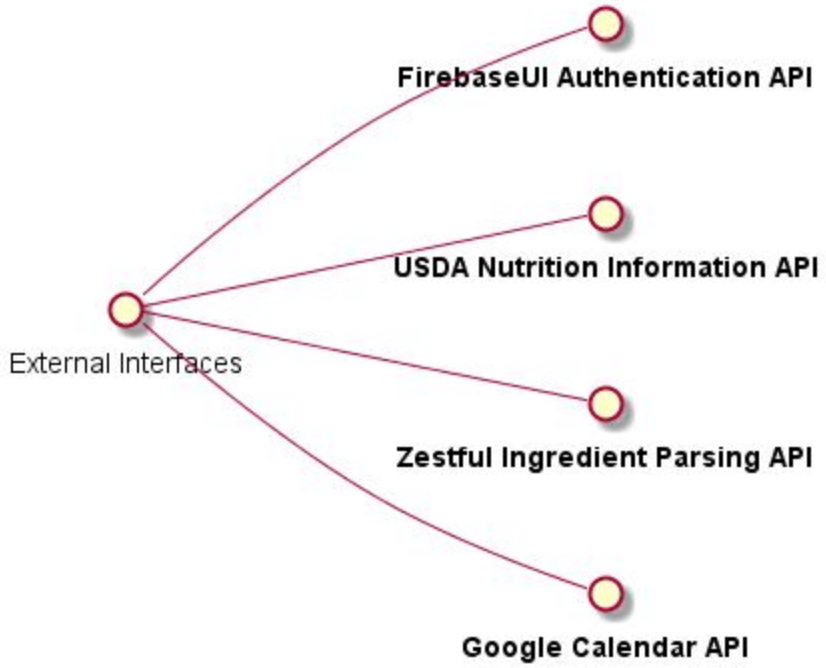


Figure 5.3.4: Elaborated external interface dependencies of the Food For All server component.

5.3.5 Traceability For Diagrams in section 5.3

Because all diagrams are based on the same sources the traceability for the diagrams is included in this section.

5.3.5.1 Client Side

SRS 3.1.1, 1.3.1.4, 3.1.1.2
SDD 5.7.1, 5.8

5.3.5.2 Shopping list

SRS 3.4.8
SDD 5.6.1.2, 5.6.1.6.4, 5.8.11

5.3.5.3 User and group Management

SRS 3.1.1.1.2, 3.2.2
SDD 5.6.1.2, 5.6.1.3, 5.6.1.4, 5.8.6, 5.8.10

5.3.5.4 Database

SRS 3.5, 3.1.1.2
SDD 5.6.1, 5.7.1

5.3.5.5 Menu Planner

SRS 3.4.8
SDD 5.6.1.2, 5.6.1.6, 5.8.9

5.3.5.6 Recipe Manager

SRS 1.2, 3.1.1.2, 3.2.3
SDD 5.6.1.2, 5.6.1.5, 5.6.1.7, 5.8.7, 5.8.8

5.3.5.7 APIs

SRS 1.3.1.4, 3.1.3
SDD 5.2.2.2

5.3.6 Languages and Softwares to be utilized

5.3.6.1 HTML
SRS 1.3.1.4, 3.4.3, 3.6.1.2
SDD 5.7.1,5.10.3.8,5.14

5.3.6.2 Javascript
SRS 1.3.1.4, 3.4.3, 3.6.1.2
SDD 5.7.1, 5.14

5.3.6.3 AngularJS
SRS 1.3.1.4, 3.4.3, 3.6.1.2
SDD 5.7.1

5.3.6.4 JQuery
SRS 1.3.1.4, 3.4.3, 3.6.1.2
SDD 5.7.1

5.3.6.3 CSS
SRS 1.3.1.4, 3.4.3, 3.6.1.2, 3.7.4.2
SDD 5.7.1, 5.14

5.3.6.4 Video Streaming OBS
SRS 3.4.3.2.1

5.3.6.5 MySQL/MariaDB
SRS 3.5.3
SDD 5.5.1.1, 5.5.1.4, 5.6.1 ,5.7.1, 5.12

5.3.6.6 Python
SRS 3.5.3
SDD 5.6.1, 5.7.1

5.4 Logical

The resources viewpoint is described in IEEE Std 1016-2009 on page 16, section 5.4: “The purpose of the Logical viewpoint is to elaborate existing and designed types and their implementations as classes and interfaces with their structural static relationships.” In this section, the UML Class Diagram language is used to elaborate the app’s class structure and relationships, as well as their members and methods. See figure 5.9 for a more visual detail on the relationships between the classes listed in this section.

5.4.1 UML Class Diagram Viewpoint

5.4.1.1 User

- **Type:** Class
- **Purpose:** Provides an object that holds information for user identification and authentication, as well as items a user has access to.
- **Description:** The user class manages user information for a single user. It uses Recipes and a ShoppingList.
- **Design Concerns from SRS:** 3.4.1.1.2, 3.5.4, 3.6.3.1, 3.6.3.2
- **Design Concerns from SDD:** 5.4.1.3, 5.6.1.3

User
- id: Integer - username: String - firstName: String - lastName: String - firebaseAccount: FirebaseAccount - profilePictureUrl: String - recipes: List<Recipe> - mealPlan: MealPlan - shoppingList: ShoppingList
+ getId(): Integer + setUsername(username: String): Void + getUserName(): String + getFirstName(): String + getFullName(): String + getProfilePicture(): Bitmap + addRecipe(recipe: Recipe): Void + removeRecipe(recipe: Recipe): Void

+ getRecipes(): List<Recipe> + setMealPlan(mealPlan: MealPlan): Void + getMealPlan(): MealPlan + setShoppingList(shoppingList: ShoppingList): Void + getShoppingList(): ShoppingList

Figure 5.4.1.1a User

5.4.1.1.1 id: Integer

The id will uniquely identify each User in the system.

5.4.1.1.2 username: String

The username will be used as a display form of identification.

5.4.1.1.3 firstName: String

FirstName will hold the user's first name.

5.4.1.1.4 lastName: String

LastName will hold the user's last name.

5.4.1.1.5 firebaseAccount: FireAuth

FirebaseAccount will be the User's Firebase account object. When a user signs in, Google returns a json object that will be parsed into a FirebaseAccount object. From here, different data can be retrieved.

5.4.1.1.6 profilePictureUrl: String

The profilePictureUrl will hold the URL of the web address that contains the User's profile picture.

5.4.1.1.7 mealPlanId: String

The Meal Plan ID will uniquely identify a User's Meal Plan.

5.4.1.1.8 shoppingListId: String

The Shopping List ID will uniquely identify a User's Shopping List.

5.4.1.1.9 getId(): Integer

GetId will return the User's ID.

5.4.1.1.10 setUsername(username: String): Void

SetUsername will set the user's username.

5.4.1.1.11 getUsername(): String

GetUsername will return the User's Username.

5.4.1.1.12 getFirstName(): String

GetFirstName will return the user's first name.

5.4.1.1.13 getFullName(): String

GetFullName will return the user's First Name and Last Name as a concatenated String.

5.4.1.1.14 getProfilePicture(): Bitmap

GetProfilePicture will return the User's profile picture at the profilePictureURL. This will default to the Google account picture until a user sets a custom one.

5.4.1.1.15 addRecipe(recipe: Recipe): Void

The addRecipe function will add only one recipe to the user's list of recipes.

5.4.1.1.16 removeRecipe(recipe: Recipe): Void

The removeRecipe function will remove only one recipe from the user's list of recipes.

5.4.1.1.17 getRecipes(): List<Recipe>

The getRecipes function will return the user's entire list of recipes.

5.4.1.1.18 setMealPlanId(id: MealPlan): Void

SetMealPlan will set the user's unique MealPlan.

5.4.1.1.19 getMealPlan(): MealPlan

GetMealPlan returns the User's Meal Plan.

5.4.1.1.20 setShoppingList(shoppingList: ShoppingList): Void

SetShoppingList will set the User's unique ShoppingList.

5.4.1.1.21 getShoppingList(): ShoppingList

GetShoppingList will return the User's Shopping List.

5.4.1.2 Group

- **Type:** Class
- **Purpose:** The Group class will provide an object that stores Users as members of itself, and the id associated with the group.
- **Description:** The Group Class manages a List of members.
- **Design Concerns from SRS:** 3.5.4.4
- **Design Concerns from DFD:** 5.4.1.12, 5.6.1.4

Group
- id: Integer - name: String - users: List<Member>
+ setId(id: Integer): Void + getId(): Integer + addUser(user: User): Void + removeUser(): Void + getUsers(): List<User>

Figure 5.4.1.2a Logical - Group

5.4.1.2.1 id: Integer

The id will be used to differentiate different groups

5.4.1.2.2 name: String

The name of the group will be set by the group members. This will not differentiate groups completely, for there can be duplicate group names. The id will control the differentiation of groups.

5.4.1.2.3 users: List<Member>

The list of member objects will hold all users associated with the group.

5.4.1.2.4 setId(id: Integer): Void

The setId function will set the id of the group. This will only be used once upon initial setup, and will check to make sure the initial id is not a duplicate of any others in the database.

5.4.1.2.5 getId(): Int

The getId function returns the Group’s unique id.

5.4.1.2.6 addUser(user: User): Void

The addUser function will add one single User to the group.

5.4.1.2.7 removeUser(): Void

The removeUser function will remove one single User from the group.

5.4.1.2.8 getUsers(): List<User>

The getUsers function will return the entire list of Users associated with the group.

5.4.1.3 Recipe

- **Type:** Class
- **Purpose:** The Recipe class will store the information users need to describe and interact with a recipe.
- **Description:** The Recipe Class manages recipe details and allows the website to manipulate or display it's information.
- **Design Concerns from SRS:** 3.2.4.1, 3.2.4.2, 3.2.8.1, 3.2.8.1.1, 3.2.11.3, 3.2.11.4, 3.2.11.5, 3.5.4.1
- **Design Concerns from SDD:** 5.6.1.5, 5.6.1.7

Recipe
- title: String - author: String - tags: List<String> - rating: List<Rating> - minServings: Integer - maxServings: Integer - nutritionFacts: NutritionFacts - ingredients: List<Ingredient> - isPublic: Bool - Instructions: String - media: String - notes: String
+ editTitle(title: String): Void + getTitle(): String + setAuthor(author: String): Void + getAuthor(): String + addTag(tag: String): Void

+ removeTag(tag: String): Void
+ getTags(): List<String>
+ setServings(min: Integer, max, Integer): Void
+ getMinServings(): Integer
+ getMaxServings(): integer
+ setNutritionFacts(nutritionFacts: NutritionFacts): Void
+ getNutritionFacts(): NutritionFacts
+ editInstructions(String: note): Void
+ getInstructions(): String
+ editNotes(note: String): Void
+ getNoes(): String

Figure 5.4.1.3a Logical - Recipe

5.4.1.3.1 title: String

The title will describe the name of the recipe.

5.4.1.3.2 author: String

The author will hold the name of the User that created the recipe.

5.4.1.3.3 tags: List<String>

The tags will be used for categorizing the recipe, and enhancing search features.

5.4.1.3.4 ratings: List<Rating>

The ratings list will hold ratings from every user that has created one for an instance of a recipe.

5.4.1.3.5 minServings: int

The minServings will describe the minimum number of servings a recipe will feed

5.4.1.3.6 maxServings: int

The maxServings will describe the maximum number of servings a recipe will feed

5.4.1.3.7 nutritionFacts: NutritionFacts

The nutritionFacts will hold all nutrition information for an instance of a recipe.

5.4.1.3.8 ingredients: List<Ingredient>

The ingredients list will hold all recipe ingredients set by the author.

5.4.1.3.9 isPublic: Bool

The isPublic value will be set by the author to allow or restrict a recipe's accessibility.

5.4.1.3.10 Instructions: String

The Instructions will be set by the author to describe how a recipe should be followed.

5.4.1.3.11 media: String

The media string will be a URL or an embed link to related media content for the recipe.

5.4.1.3.12 editTitle(title: String): Void

The editTitle function will change the title of the Recipe.

5.4.1.3.13 getTitle(): String

The getTitle function will return the title.

5.4.1.3.14 setAuthor(author: String): Void

The setAuthor function will set the author of the Recipe.

5.4.1.3.15 getAuthor(): String

The getAuthor function will return the name of the author.

5.4.1.3.16 addTag(tag: String): Void

The addTag function will add a new tag to the list of tags.

5.4.1.3.17 removeTag(tag: String): Void

The removeTag function remove a new tag from the list of tags.

5.4.1.3.18 getTags(): List<String>

The getTags function will return the entire list of tags.

5.4.1.3.19 setServings(min: Integer, max, Integer): Void

The setServings will set both the min and max servings for a recipe.

5.4.1.3.20 getMinServings(): Integer

The getminServings will return only the minimum number of servings a recipe can provide.

5.4.1.3.21 getMaxServings(): integer

The getminServings will return only the maximum number of servings a recipe can provide.

5.4.1.3.22 setNutritionFacts(nutritionFacts: NutritionFacts): Void

The setNutritionFacts function sets the recipes NutritionFacts object.

5.4.1.3.23 getNutritionFacts(): NutritionFacts

The getNutritionFacts function returns the recipe's nutrition facts.

5.4.1.3.24 editInstructions(String: note): Void

The editInstructions function replaces the Instructions in a recipe.

5.4.1.3.25 getInstructions(): String

The getInstructions function returns the entire string of instructions from a recipe.

5.4.1.3.26 editNotes(notes: String): Void

The setNotes method will replace the entire string of notes.

5.4.1.3.27 getNotes(): String

The getNotes method will return the entire string of notes.

5.4.1.4 Rating

- **Type:**Class
- **Purpose:** Holds a User's rating for a recipe.
- **Description:** The Rating class holds the details of a User's rating. It includes a star rating between 1 and 5, and a comment. The Recipe class has a list of these.
- **Design Concerns from SRS:** 3.5.4.1, 3.2.7.1, 3.2.7.2
- **Design Concerns from SDD:** 5.6.1.3.5**,** 5.4.1.3

Rating
- rating: Float - Comment: String - User: User
+ setRating(rating: Float): Void + getRating(): Int

+ editComment(comment: String): Void + getComment(): String + setUser(user: User): Void + getUser(): User

Figure 5.4.1.4a Logical - Rating

5.4.1.4.1 rating: Float

The rating will the number of stars a Recipe is given by a user. The value will be between 1 and 5, and it is a float to be able to represent half star values.

5.4.1.4.2 comment: String

The comment will hold the user's comment associated with their rating.

5.4.1.4.3 user: User

The user will represent the user that gave a rating.

5.4.1.4.4 setRating(rating: Float): Void

The setRating method will set the rating value. It should bounds-check to assure that the value is between 1 and 5.

5.4.1.4.5 getRating(): Float

The getRating method will return the rating value.

5.4.1.4.6 editComment(comment: String): Void

The editComment method will replace the comment value with a new comment.

5.4.1.4.7 getComment(): String

The getComment method will return the user's comment as a string.

5.4.1.4.8 setUser(user: User): Void

The setUser method will set the Rating's user object.

5.4.1.4.9 getUser(): User

The getUser will return the user associated with a Rating.

5.4.1.5 ShoppingList

- **Type:**Class

- **Purpose:** Provides an object to store and manage a list of ShoppingListItems.
- **Description:** The Shopping List class will manage a List of ShoppingListItems whose status can be tracked by a user.
- **Design Concerns from SRS:** 3.2.13.1, 3.5.4.2
- **Design concerns from SDD:** 5.4.1.6

ShoppingList
- items: List<ShoppingListItem>
+ addItem(item: ShoppingListItem, bool: Bool): Void + removeItem(item: ShoppingListItem): Void + getItems(): List<ShoppingListItem>

Figure 5.4.1.5a Logical - Shopping List

5.4.1.5.1 ingredients: List<ShoppingListItem>

The items list will keep track of the items the User needs to purchase for their Recipes. It includes a bool to keep track of the state it is in: whether the user has it or not.

5.4.1.5.2 addItem(item: ShoppingListItem, bool: Bool): Void

The addItem method will add another item to the map of ShoppingListItems. As the map is created, items will be combined and their units converted.

5.4.1.5.3 removeItem(item: ShoppingListItem): Void

Remove will remove one ingredient from the list of

5.4.1.5.4 getItems(): List<ShoppingListItem>

GetItems will return the entire list of ShoppingItems.

5.4.1.6 ShoppingListItem

- **Type:**Class
- **Purpose:** Provides an object to store a list of ingredients that are the same, and allows the user to see a more shopping friendly unit of measurement.
- **Description:** The ShoppingListItem class will use a list of ingredients and convert their quantities to one main ingredient and quantity. If a different unit is required, the class will use a unit member and to convert the mainIngredient quantity to the requested type.

- **Design Concerns from SRS:** 3.2.13.1, 3.5.4.2
- **Design concerns from SDD:** 5.5.1.4, 5.8.9.1

ShoppingListItem
- ingredients: List<Ingredient> - mainIngredient: Ingredient - unit: String - acquired: Bool
+ getMainIngredient(): Ingredient + addIngredient(ingredient: Ingredient): Void + removeIngredient(ingredient: Ingredient): Void + setUnit(unit: String): Void + setAcquired(acquired: Bool): Void + getAcquired(): Bool

Figure 5.4.1.6a Logical - Shopping List Item

5.4.1.6.1 ingredients: List<Ingredient>

The ingredient will hold the hidden list of ingredients.

5.4.1.6.2 mainIngredient: Ingredient

The mainIngredient will hold the ingredient type and total quantity of the list of ingredients.

5.4.1.6.3 unit: String

The unit will define the type unit that the user will see in their list. It will hold the unit that will display to the user.

5.4.1.6.4 acquired: Bool

Acquired will allow the user to differentiate between ingredients they have and don't have.

5.4.1.6.5 getMainIngredient(): Ingredient

GetMainIngredient will convert the list of ingredients to one ingredient, the mainIngredient, and return the mainIngredient object. If a different unit is requested, this class will convert the quantities to a new quantity.

5.4.1.6.6 addIngredient(ingredient: Ingredient): Void

AddIngredient will add ingredients to the shopping list.

5.4.1.6.7 removeIngredient(ingredient: Ingredient): Void

RemoveIngredient will remove ingredients from the shopping list.

5.4.1.6.8 setUnit(unit: String): Void

SetUnit will set the unit member, whitelisting supported units. No getter is required because the unit is only used by the class in Quantity conversion.

5.4.1.6.11 setAcquired(acquired: Bool): Void

SetAcquired will set the acquired value.

5.4.1.6.12 getAcquired(): Bool

GetAcquired will return the acquired value.

5.4.1.7 Ingredient

- **Type:**Class
- **Purpose:** Provide an object to store the ingredient names and quantity.
- **Description:** The Quantity Class will manage name, userHas, for the recipe that the user is shopping for.
- **Design Concerns from SRS**: 3.5.4
- **Design Concerns from SDD:** 5.4.1.3

Ingredient
- name: String - quantity: Quantity
+ setName(name: String): Void + getName(): String + setQuantity(quantity: Quantity): Void + getQuantity(): Quantity

Figure 5.4.1.7a Logical - Ingredient

5.4.1.7.1 name: String

The name will describe what the actual ingredient is.

5.4.1.7.2 quantity: Quantity

The quantity will quantify the ingredient.

5.4.1.7.3 setName(name: String): Void

SetName will set the ingredient’s name.

5.4.1.7.4 getName(): String

GetName will return the ingredient's name.

5.4.1.7.5 setQuantity(quantity: Quantity): Void

SetQuantity will set the ingredient's Quantity object.

5.4.1.7.6 getQuantity(): Quantity

GetQuantity will return the ingredient's Quantity.

5.4.1.8 Meal

- **Type:**Class
- **Purpose**: The Meal class will manage the date and the recipe for the user to plan a meal.
- **Description:** The Meal class will manage a recipe and schedule of when the recipe should be made.
- **Design Concerns from SRS:** SRS 3.5.4.1**,** SRS 3.5.4.3**,** SRS 3.5.4.1
- **Design Concerns from SDD:** 5.6.1.5, 5.6.1.6.2

Meal
- date: Date - recipe: Recipe
+ Meal(date: Date, recipe: Recipe): Meal + changeDate(date: Date): Void + changeRecipe(recipe: Recipe): Void + getDate(): Date + getRecipe: Recipe

Figure 5.4.1.8a Logical - Meal

5.4.1.8.1 date: Date

The date for when the recipe was added or altered

5.4.1.8.2 recipe: Recipe

Holds the recipe and all the information that is needed to execute the recipe.

5.4.1.8.3 Meal(date: Date, recipe: Recipe): Meal

Has the recipe and the date that it will be planned to prepare.

5.4.1.8.4 changeDate(date: Date): Void

Has the date so that the user can adjust when the recipe will be made.

5.4.1.8.5 changeRecipe(recipe: Recipe): Void

Enables the user to edit the recipe.

5.4.1.8.6 getDate(): Date

Will retrieve the date that the meal will be planned.

5.4.1.8.7 getRecipe: Recipe

Will retrieve the recipe that the user wants to schedule.

5.4.1.9 MealPlan

- **Type:**Class
- **Purpose:** Store the daily and weekly meal planners
- **Description:** The Meal Plan Class will will manage a map with the keys date, and recipe that will be used to find the daily and weekly meal planners.
- **Design Concerns from SRS:** 3.2.12.1, 3.2.13.3, 3.3.8, 3.5.4.3
- **Design Concerns from SDD:** 5.6.1.6.3

Meal Plan
- meals:List<Meal>
+ addMeal(meal: Meal): Void + removeMeal(meal: Mesl): Void

Figure 5.4.1.9a Logical - Meal Plan

5.4.1.9.1 meals: Map<Date, Recipe>

Meals will be an arranged map of meals with the key and value of date and recipe.

5.4.1.9.2 addMeal(meal:Meal): Void

AddMeal will enable the user to add a meal to their meal planner

5.4.1.9.3 removeMeal(meal:Meal): Void

RemoveMeal will enable the user to add a meal to their meal planner

5.4.1.10 Nutrient

- **Type:** Class
- **Purpose:** The Nutrient class will provide a single label associated with a Quantity.
- **Description:** The Nutrient class will allow for new nutrients to be added to Recipes, with different labels and units.The label will be the nutrient, while the quantity will describe the amount of the nutrient contained.
- **Design Concerns from SDD:** 5.6.1.3.1.1, 5.4.1.9

Nutrient
- label: String - quantity: Quantity
+ setNutrient(label: String, quantity: Quantity): Void + getLabel(): String + getQuantity(): Quantity + displayNutrient(): Void

Figure 5.4.1.10a Logical - Nutrient

5.4.1.10.1 label: String

Provides the label for the ingredient.

5.4.1.10.2 quantity: Quantity

Lists the amount of the ingredient.

5.4.1.10.3 setNutrient(label: String, quantity: Quantity): Void

Sets the label and quantity to an ingredient and sets its nutrients.

5.4.1.10.4 getLabel(): String

Gets the label string.

5.4.1.10.5 getQuantity(): Quantity

Gets the quantity for the ingredient.

5.4.1.10.6 displayNutrient(): Void

DisplayNutrient will display the nutrients.

5.4.1.11 NutritionFacts

- **Type:** Class
- **Description:** NutritionFacts will hold a number of servings, and a list of type Nutrient. These nutrients may include but are not limited to: calories, macronutrients, and micronutrients like vitamins. A recipe will have a NutritionFacts object.
- **Design Concerns from SRS:** 3.2.6.1, 3.2.6.1.1
- **Design Concerns from SDD:** 5.4.1.3, 5.4.1.9, 5.6.1.3.1.1

NutritionFacts
- Servings: float - nutrients: List<Nutrient>
+ setServings(): void + getServings(): float + addNutrient(): void + getNutrients(): List<Nutrient>

Figure 5.4.1.11a Logical - Nutrition Facts

5.4.1.11.1 Servings: float

The servings will represent the number of servings an item has. Every Recipe will have one and only one of these. The float data type is used to allow for fractions.

5.4.1.11.2 setServings(): void

The setServings function will set the NutritionFacts serving item. Error checking should be done for values that are 0 or less.

5.4.1.11.3 getServings(): float

The getServings function will return the number of servings as a float.

5.4.1.11.4 addNutrient(): void

The addNutrient function will insert another nutrient to the end of the nutrients list. Error checking should be done to make sure that the item does not already exist in the list.

5.4.1.11.5 getNutrients(): List<Nutrient>

The getNutrients function returns the entire list of nutrients.

5.4.1.12 Quantity

- **Type:** Class
- **Purpose:** Provides an object to store an amount needed and the units label.
- **Description:** The Quantity Class will manage the units and amounts of an item. It is for the purpose of an item that needs an identifier of unit measurement. For example, an Ingredient will have measurements such as “oz” or “cups” associated with a number, and this class provides that detail. A nutrient will also have this.
- **Design Concerns from SRS:** 3.5.4
- **Design Concerns from SDD:** 5.4.1.9, 5.4.1.6

Quantity
- Units: String - Amount: Float
+ setUnits(units: String): Void + setAmount(amount: Float): Void + getUnits(): String + getAmount(): Float

Figure 5.4.1.12a Logical - Quantity

5.4.1.12.1 Units: string

The units variable shall denote what the amount refers to. This may be oz, cups, lbs, gallons, etc.

5.4.1.12.2 Amount: float

The amount shall contain the value that represents an item. Being a float, it allows for fractional measurements.

5.4.1.12.3 setUnits(units: String): Void

The setUnits function shall set the units variable to the input string.

5.4.1.12.4 setAmount(amount: Float): Void

The setAmount function shall set the units to an appropriate value. It should filter for negative values, since those should not exist.

5.4.1.12.5 getUnits(): String

The getUnits function shall return the units string.

5.4.1.12.6 getAmount(): Float

The getAmount function shall return the amount stored in Quantity.

5.4.1.13 Member

- **Type:** Class
- **Purpose:** The Member class will provide a way to link Users to their permissions for an item.
- **Description:** The Member class will contain a User and a permission. The permission will be used to authorize a User for different levels of interaction with another object. It will be used in the Group class.
- **Design Concerns from SDD:** 5.4.1.2

Member
- user: User - permission: String
+ setUser(user: User): Void + getUser(): User + setPermission(permission: String): Void + getPermission(): String

Figure 5.4.1.13a Logical - Member

5.4.1.13.1 user: User

The user will hold a User object.

5.4.1.13.2 permission: String

The permission will hold a word that identifies a level of interaction the associated user has with an item.

5.4.1.13.3 setUser(user: User): Void

SetUser will set the Member's User object.

5.4.1.13.4 getUser(): User

GetUser will get the Member's User object.

5.4.1.13.5 setPermission(permission: String): Void

SetPermission will set the Member’s permission descriptor. It should perform some syntax checking for whitelisted characters.

5.4.1.13.6 getPermission(): String

GetPermission will get the Member’s permission descriptor.

5.5 Dependency

The Dependency viewpoint is described in IEEE Std 1016-2009 on page 17, section 5.5. The Dependency viewpoint specifies the relationships of interconnection and access among entities.

5.5.1 UML Component Diagram

5.5.1.1 share Recipe

- **Type:** Function
- **Purpose:** To set the permissions of access to a recipe
- **Description:** Share recipe uses the recipe id and the scope (private, group, or public) of who can view the recipe. It will then insert the information into the database, providing only the users in the same scope (private, group, or public) access to the recipe.
- **Design concerns from SRS:** **3.3.10**

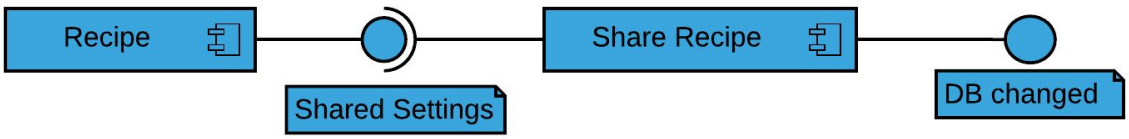


Figure 5.5.1.1.1. Component Diagram - Share Recipe

5.5.1.3 Nutrition Information API

- **Type:**API
- **Purpose:** To get the nutritional facts for a recipe
- **Description:** USDA Nutritional information API will allow the sending of ingredients or food item with responses that provide all nutritional information in the USDA nutritional database that is relative to the recipe or food item that you are making/inquiring about.
- **Design Concerns from SRS:** **3.1.3.1.1.1**

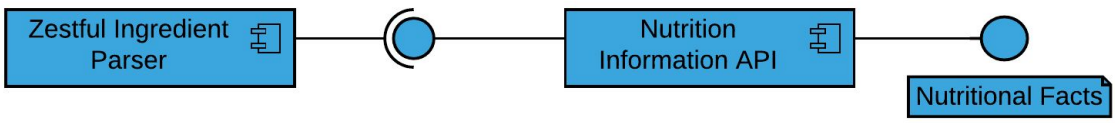


Figure 5.5.1.3.1. Component Diagram - Nutrition Information API

5.5.1.5 FirebaseUI Authentication API

- **Type:** API
- **Purpose:** To provide authentication of users
- **Description:** FirebaseUI Authentication API that is used will be a standard login page/form that verifies the user's credentials and allows the user to 'login' and use the application.
- **Design concerns from SRS:** 3.1.3.1.1.3

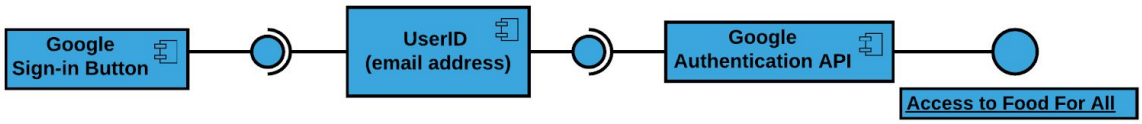


Figure 5.5.1.5.1. Component Diagram - Google Authentication API

5.5.1.6 Zestful Ingredient Parsing API

- **Type:** API
- **Purpose:** Capture individual ingredients from a recipe and convert them in a useable format to create a shopping list and send to the nutrition information API.
- **Description:** Zestful Ingredient parsing API will scan the ingredients of a recipe and parse them into separate ingredients. Then the Zestful Ingredient parsing API will communicate with USDA Nutritional API and send ingredients one at a time to retrieve accurate, portioned, and relevant nutritional information that relates to what is being cooked. The parsed ingredients can also be used to create a shopping list.
- **Design Concerns from SRS:** 3.1.3.1.1.4

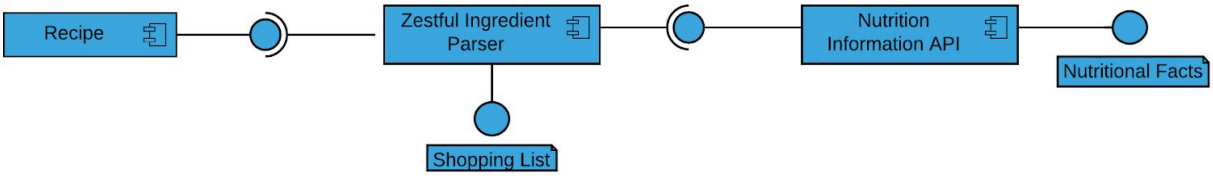


Figure 5.5.1.6.1. Component Diagram - Zestful Ingredient Parser

5.5.1.7 Google Calendar API

- **Type:** API
- **Purpose:** To create, edit, and share a meal and menu planners
- **Description:** Google calendar API uses the calendar ‘insert’ method to create a google calendar, using the calendarID It provide adding recipes to the calendar using the events ‘insert’ method. It also provides the ability to share calendars with other users using the access control rule insert method and providing and users email address using the security API.
- **Design concerns from SRS:** **3.1.3.1.1.5**, 3.1.4.1.1, 3.1.4.1.2

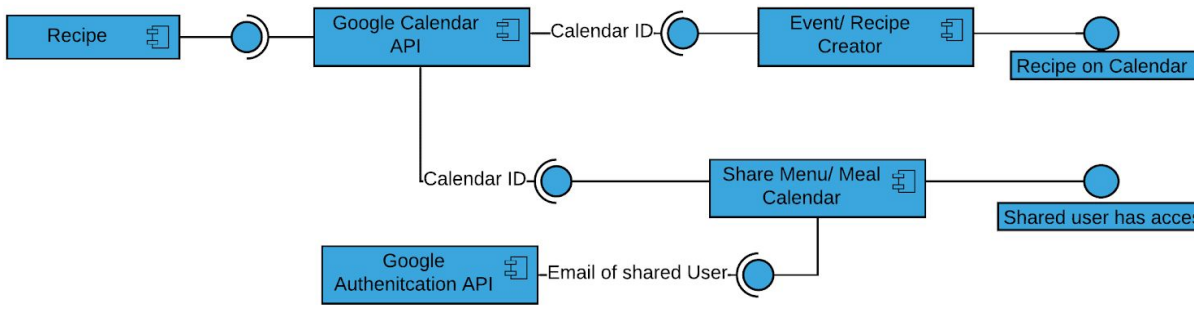


Figure 5.5.1.7.1. Component Diagram - Google Calendar API

5.5.1.8 Google Calendar Event Creations

- **Type:** Process
- **Purpose:** Create a Google calendar event with a recipe title.
- **Description:** With the recipe title and calendar id on the calendar the user want to add an event to it will create an event with the recipe title on the Google calendar.
- **Design Concerns from SRS:** 3.1.4.1.1

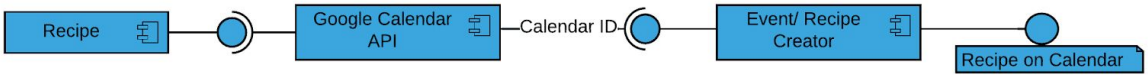


Figure 5.5.1.8.1. Component Diagram - Google Calendar Event Creations

5.5.1.9 Share Google Calendar Events

- **Type:** Process
- **Purpose:** Share meal and menu plans.
- **Description:** Share google calendar events will be used with the google API, using the calendarID and email of the shared user that is provided by the security API, to provide the user to share menu and meal plans with another user.
- **Design Concerns from SRS:** 3.1.4.1.2

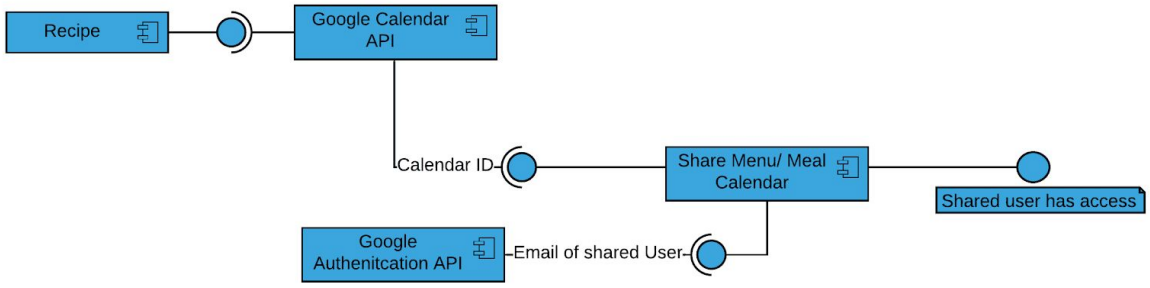


Figure 5.5.1.9.1. Component Diagram - Share Google Calendar Events

5.6 Information

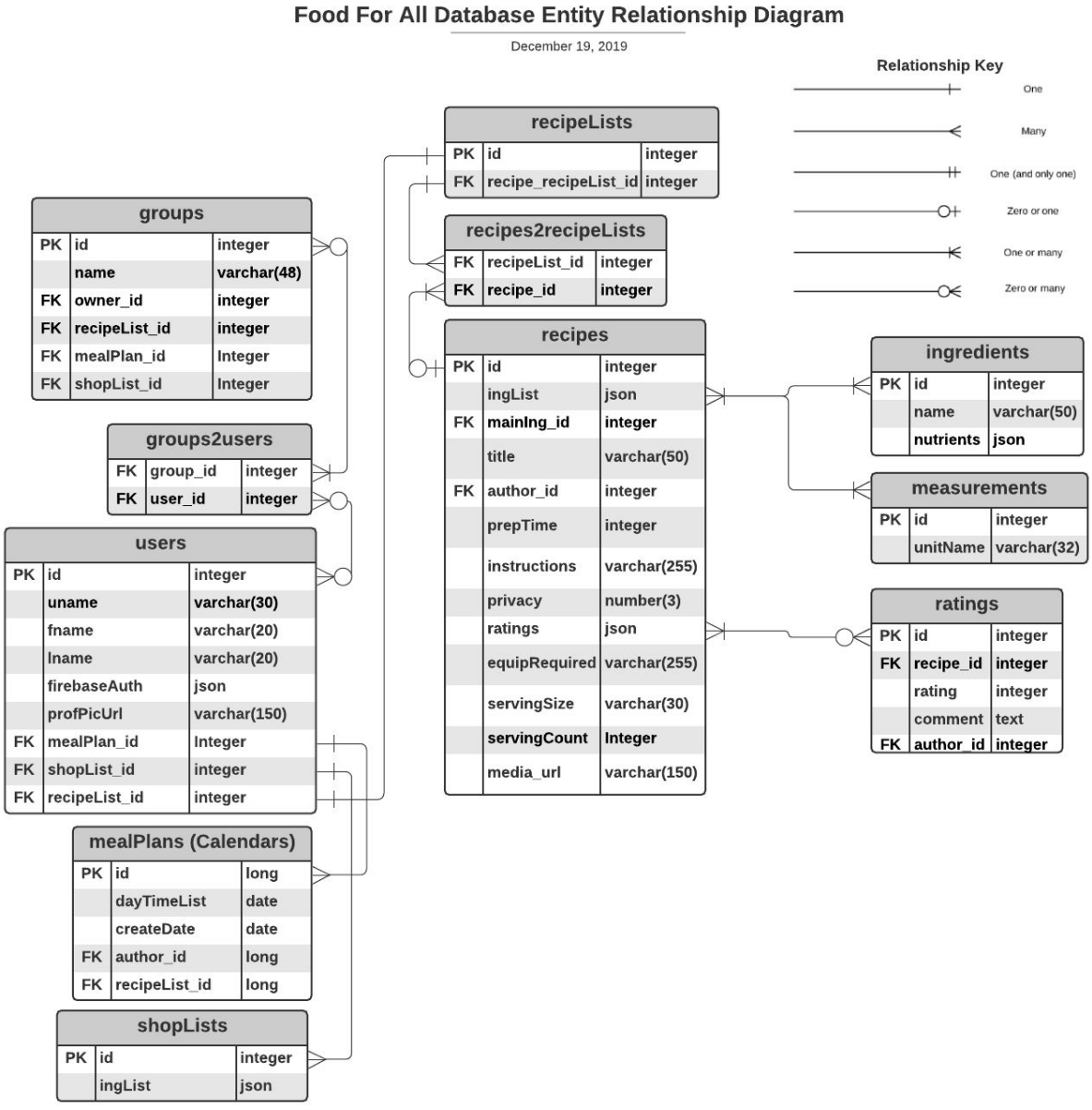
The **Information viewpoint**, as described in IEEE 1016-2009 section 5.6 on page 17, is applicable when there is a substantial persistent data content expected with the design subject. The information viewpoint includes concerns regarding persistent data, content, data management, access, and definitions.

5.6.1 Database

Database tables have been designed to work in conjunction with the models created in Python, which will act as our server-side language. These database entities are meant to be easily transferred to the classes within our model through the creation of a proprietary adapter that will query the database tables and fill our model objects through assignment from the query results. This will lead to a better abstracted product and will allow for better future-proofing post launch if we eventually decide to port the program to other platforms.

The following graphic displays database structure along with entity relationships within the tables:

Fig 5.6.1



** Due to space constraints, not all Foreign Keys are connected but can be implied.*

*** JSON fields imply data that has many values inside, many of them specific to the parent entity instance, which may be too complicated to visualize in database table/entity relationships. Some of this JSON data may be retrieved from an API and therefore not structured in our database.*

5.6.1.1 Database Access and Privileges

Clients shall have access to create, modify, and delete objects which they author, including their user account, recipes, groups, meal plans, shopping lists, and reviews. The **Service** layer will act as a “middle-man” by **sanitizing** input before the **Controller** sends data requests to Service layer to access database. This means that before any user input is sent to the controller it will be parsed, and any special characters, like parentheses or semicolons, will be removed from the input to avoid vulnerability exploits. Clients shall *not* have access to modify other authors’ data (nor execute database and table level commands) and, in most cases, may only view data of another author that is not **private**. No client shall have direct access to the database. The Controller shall relay input and the Service layer layer shall be the only layer that has direct access to the database. Clients only have direct access to the **View** layer, which displays data and HTML to the user. (SRS 3.5.4, SRS 3.5.4.4)

5.6.1.2 Service Layer

The notion of a service layer within our application will contain database querying services and data sanitization methods to create a higher level of abstraction, insure all queries are performed uniformly, and all data is formatted correctly for proper use by the models within the application and tables within the database.

5.6.1.2.1 Data Sanitization and Proprietary Adapter

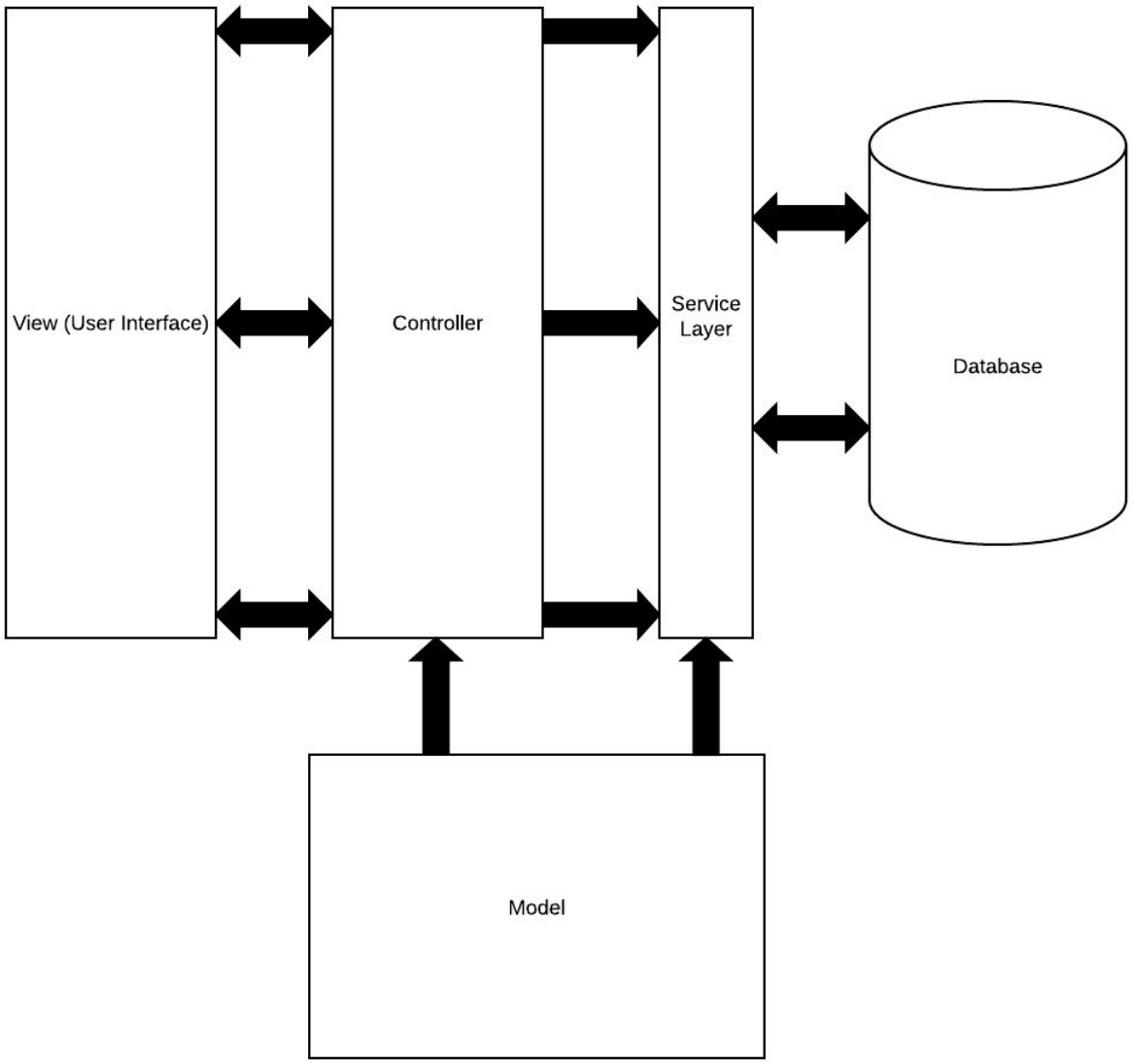
These sanitization methods will be used as a proprietary adapter between the application’s models and the database. The service layer will contain various methods that will query the database given keywords passed through the function and will return a list of the corresponding model objects to be used by the application that match the keyword in the query. Update methods in the service layer will work in a similar fashion, where it will be passed a model object and update the corresponding database entry after sanitizing input and making it compatible with the database tables. Sanitizing input can include but is not limited to: converting units between numbers, changing the data types for greater security or modifying data to be better handled within the application or database. Sanitization is simply modifying the data to make it compatible and secure for its intended use.

5.6.1.2.2 Intended Use and Benefits

These service layer functions will commonly be called within the controllers. The benefit of creating a service layer is that if our query ever need be modified, we need to only modify it within our service layer opposed to every time the query is used. It also provides a greater amount of traceability among function calls and allows quick modification to potential bugs that will greatly aid in debugging. Furthermore, if we ever need to change or restructure our database, changes only to the service layer will be required. The controller, models, and view will all be able to function identically and maintenance will be significantly easier.

The following graphic displays service layer relationship with other aspects of MVC design pattern:

Fig 5.6.2



5.6.1.3 User

A **user** shall have a username, first name, last name, and FirebaseUI Authentication API information as part of their personal details. Each user shall have a meal plan list, a shopping list, and a recipe list. (SRS 3.7.2.1)

5.6.1.3.1 Username

The username shall be held within the user table, along with other personal account information such as first and last name, and FirebaseUI Auth information. (SRS 3.5.4)

5.6.1.3.2 Authentication

The database will not hold any hashed versions of passwords thus not involved in user authentication and will delegate such authentication functionality to the FirebaseUI Authentication API. (SRS 3.1.3)

5.6.1.3.2.1 Firebase SDK Authentication API

The Firebase SDK Authentication API will store, encrypt, and retrieve information such as email, passwords, and other important user data for user authentication. The Firebase SDK Authentication API returns a token_id (UID) which is a token issued to client upon successful login. (SRS 3.1.3)

5.6.1.3.3 Profile Picture

The profile picture of a user will be stored in the database as a URL to the picture that we will store on the server. (SRS 3.5.4)

5.6.1.4 Group

A group shall be created by a single user and shall contain one or more users, invited by the group administrator.

5.6.1.4.4 Group table

The group table will hold rows of data that have the group_id and a user_id. This will allow users to be members of multiple groups. (SRS 3.5.4.4)

5.6.1.5 Recipe

A **public** recipe is owned by one author, but is viewable by every user. This allows the recipe to show up in searches and filters of any user, and to be “liked” or “favorited”. If the recipe privacy is set to **followers**, only other users who “follow” that user can see that recipe. Else, the recipe

is **private** to the author, unless they share it in a direct message or a group. This setting in the database shall function like a struct or enum variable with each of these settings, either labeled as such or codified from 0 to 2. (SRS 3.5.4.1)

5.6.1.5.1 Recipe Associated Users

A recipe belongs to a single author/group. This is tracked with a **author_id** in the recipe object in the database. (SRS 3.5.4.1)

5.6.1.5.2 Recipe Associated Group

A recipe can be in multiple groups, even while private, public, or otherwise outside the group. Group members shall be able to add recipes to a group (giving the group shared access to those recipes). Group recipe modifications shall not change the global recipe that already existed. This necessitates the creation of a separate recipe **instance** when added to a group (copy of the global recipe but with a different **recipe_id**). This way, groups can make changes to a recipe just as a single user would to another author's recipe that is then saved as one of their own, rather than modifying the original. (SRS 3.5.4.4.1)

5.6.1.5 Search Efficiency (Recipe Table)

The properties of a recipe object shall be the parameters by which recipes can be searched, sorted, and filtered.The database language used in the Model layer shall be the driving force behind users' ability to query and return recipe records that match their query parameters. (SRS 3.5.4.1)

5.6.1.5.3 Copyright Agreement

When signing up to use the application for the first time, a user shall be presented with a EULA that includes language prohibiting the publishing of recipe content that is copyrighted (such as recipe descriptions, images, etc. Preparation/Cooking steps and ingredient lists cannot be copyrighted.). Upon clicking the "accept terms" checkbox and clicking "okay", the user enters into this binding agreement to publish only content legally publishable, and that we (the application) are granted permission to control, edit, remove, or use this content. This agreement is tracked as a boolean value in our database, showing every user that agrees to abide by the EULA. In reality, no user shall have a False value in this field. If they don't agree, their account shall not be created and they shall not be permitted to use the application. If there is ever a False value in this field found in the database, something is wrong, as this points to a user who somehow circumvented our agreement and had an account created anyway. This needs to be protected against in the sanitization process in the controller or service layer.

5.6.1.6 Menu Planner

5.6.2.6.1 Add Recipe to Menu Planner

A Service layer function that is able to update the database's Menu Planner table by passing the recipe's foreign key value and making it associated with the Menu plan object that will be in the update statement. (SRS 3.5.4.3)

5.6.1.6.2 Edit Recipe in Menu Planner

Modified recipes within the database will be handled by creating a duplicate of the original and changing the primary key value to become a unique recipe. This now modified recipe will be set to private as default and only viewable by the users of the specific meal plan it is associated with. (SRS 3.5.4.3)

5.6.1.6.3 Meal Plan Object

The meal plan object will be represented both as a model class within the application and as its own database table within the database. The service layer will handle the transferring and modification between the two. The database table will hold the following values: meal_plan_id, recipe_list_id which will link to a table containing recipes linked to the meal plan, create_date to track it's origin, it's original author, and a dayTime_list that tracks its duration. (SRS 3.5.4.3)

5.6.1.6.4 Shopping List

The shopping list will be stored in the database as a table with rows of data that have shopping list ids linked to specific ingredients. (SRS 3.5.4.2)

5.6.1.7 Recipe

5.6.1.3.1 List of Ingredients

The list of ingredients will be held in an ingredients_list table that will have the recipe_id and the ingredient_id. (SRS 3.5.4.1)

5.6.1.3.1.1 Nutrients

Recipe ingredients shall also be stored with a JSON field to store nutrition information grabbed from nutrition API upon "creating" ingredient in database. (SRS 3.2.6)

5.6.1.3.2 Instructions

Recipe cooking instructions will be held as text that will have the ability to be parsed for formatting purposes. (SRS 3.5.4.1)

5.6.1.3.3 Title

The recipe's title, stored as a string in the recipe table. (SRS 3.5.4.1)

5.6.1.3.4 Author

The recipe's author will be stored as a foreign key value of user_id to link to the specific user who created the recipe. (SRS 3.5.4.1)

5.6.1.3.5 Rating

Recipe ratings will be stored in a separate rating_list table that will have rows of the various ratings for that recipe. The processing for overall rating will be done in the service layer. (SRS 3.5.4.1)

5.6.1.3.6 Tags

A string with a comma delimiter containing all of the tags on the recipe. (SRS 3.5.4.1)

5.6.1.3.7 Serving Size

A string containing the serving size of the recipe. A serving count column contains the amount of people the recipe serves. (SRS 3.5.4.1)

5.6.1.3.9 Media

A url to the media to be imbedded within the webpage contained within the recipe table. (SRS 3.5.4.1)

5.6.1.3.10 Required Equipment

A string of equipment that is required for the recipe, separated by a comma delimiter. (SRS 3.5.4.1)

5.6.1.3.11 Time to Prepare

An integer containing the number of minutes required to prepare and cook the recipe. (SRS 3.5.4.1)

5.7 Patterns

The patterns viewpoint is described in IEEE Std 1016-2009 on page 18, section 5.7. The primary concern of a patterns viewpoint is the reuse of design patterns, styles and frameworks. Section 5.7 of the SDD will be primarily concerned with describing and illustrating the AngularJS framework.

5.7.1 AngularJS Framework

AngularJS is an open-source JavaScript based framework that acts as a template for web applications that are run on a client's web browser (SRS 3.6.1.1), meaning, all code is executed on the web-browser (SRS 1.3.1.3). Due to AngularJS's comprehensive handling of all JavaScript code (SRS 1.3.1.4), any applications developed upon its framework are cross-browser compliant (SRS 3.2.1.3). AngularJS is a powerful front-end web app tool for overcoming the challenges of creating single-page applications by simplifying its development and testing. Released in October of 2010, the AngularJS framework has a solid foothold in the industry and is well maintained by Google.

Using AngularJS allows the development team to use many tools that would otherwise not be available (SDD 5.14.9). These tools include but are not limited to Libraries, Data-binding, Directives, Reusability, and Testability. AngularJs has many libraries that are easily added on to enhance the app development experience. These libraries include resources to complete HTTP authentication (SRS 3.5.5), data handling, file uploading, etc. These libraries raise the potential capabilities of each developer as they use them. Data-binding, is an automatic way of updating the view anytime the model changes, which frees up time for the developer by reducing or eliminating DOM manipulation concerns. Directives, allow an immense amount of control over HTML and allow the developer to invent HTML (SRS 1.3.1.4) syntax specific to their application. Reusable Components, allow the developer to focus on what the user sees or what the application does rather than seeing CSS (SRS 1.3.1.4) behavior, or DOM structure. Testability, AngularJs comes with end-to-end scenario runner which eliminates test flakiness by understanding the inner workings of the code.

With all the tools and the testability that AngularJS provides, it allows for streamline development of single-page applications (SDD 5.14.9) that are scalable and reusable. This is important for the development process as the application will need the flexibility to continually grow and adapt to new users and data.

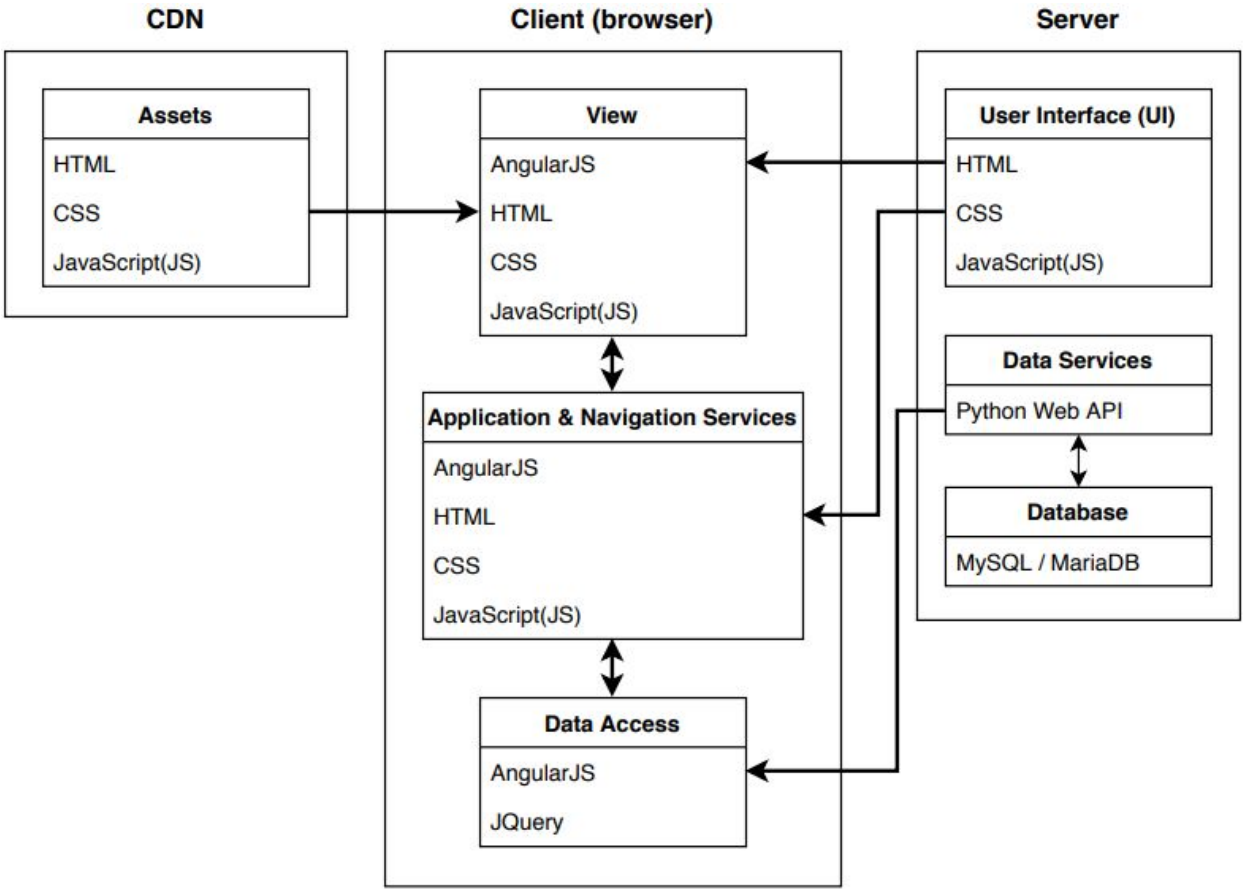


Figure 5.7.1.1. Patterns - AngularJS

Figure 5.7.1.1 illustrates how AngularJS would be implemented for the Food For All app. The Content Delivery Network (CDN) would provide important assets to the Client via its view where the AngularJS framework would manipulate the style, layout, and code to provide a respectable single-page application.While the Server will provide the client important data via the User Interface, Data Services, and Database, the Client would utilize AngularJS to make the different modules accessible by the developer.

5.7.2 Single Page Application

A single page application (SPA) is an application that runs inside the clients web-browser which does not need to reload in order to show new content. As the user request new content, the web-page is dynamically updated without having to reload the page and all of its content. For example, Netflix uses a single page design to serve up its video content. The user browses video content without ever leaving the main page. All content in a SPA are served up using JavaScript frameworks such as AngularJS (SDD 5.7.1). Since all code such as HTML, CSS, and JavaScript are loaded only once, the speed and responsiveness is greater than in multi page apps (MPA). Figure 5.7.2.1 illustrates the Single Page Application pattern.

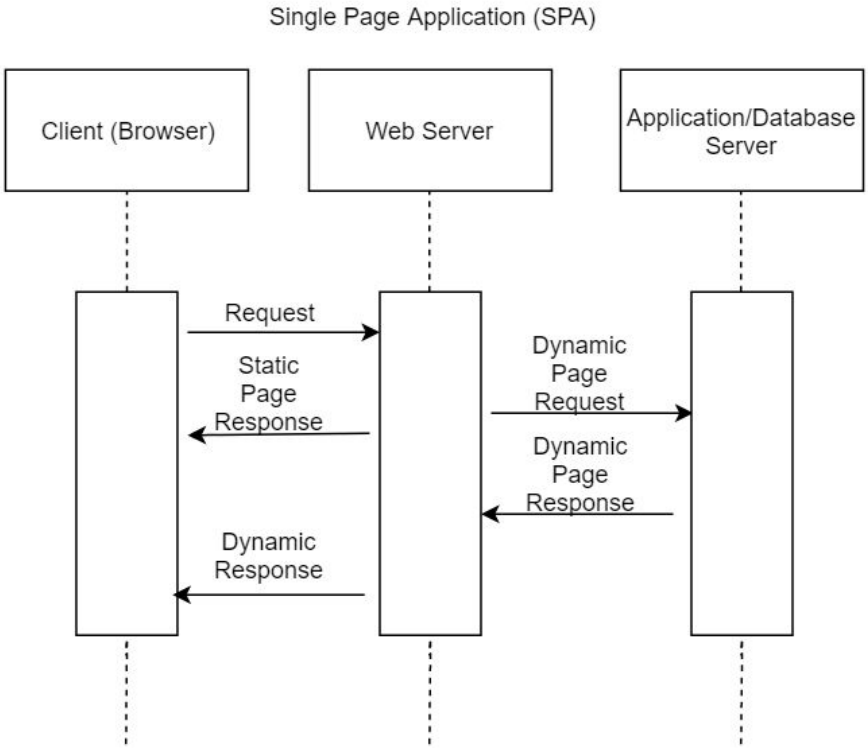


Figure 5.7.2.1. Patterns - Single Page Application

5.8 Interface

The User Interface viewpoint is described in IEEE Std 1016-2009 on page 19-20, section 5.8.The purpose of the Interface viewpoint is to provide details for how the user interface is to be designed. The user interface will use a responsive design that will allow the interface to adapt to the screen size and orientation of the users device.

5.8.1 Navigation

The navigation view is a basic overview of what the navigation should look like in desktop (see figure 5.8.1.1), mobile profile(see figure 5.8.1.2), and mobile landscape (see figure 5.8.1.3) views.

5.8.1.1 Desktop View

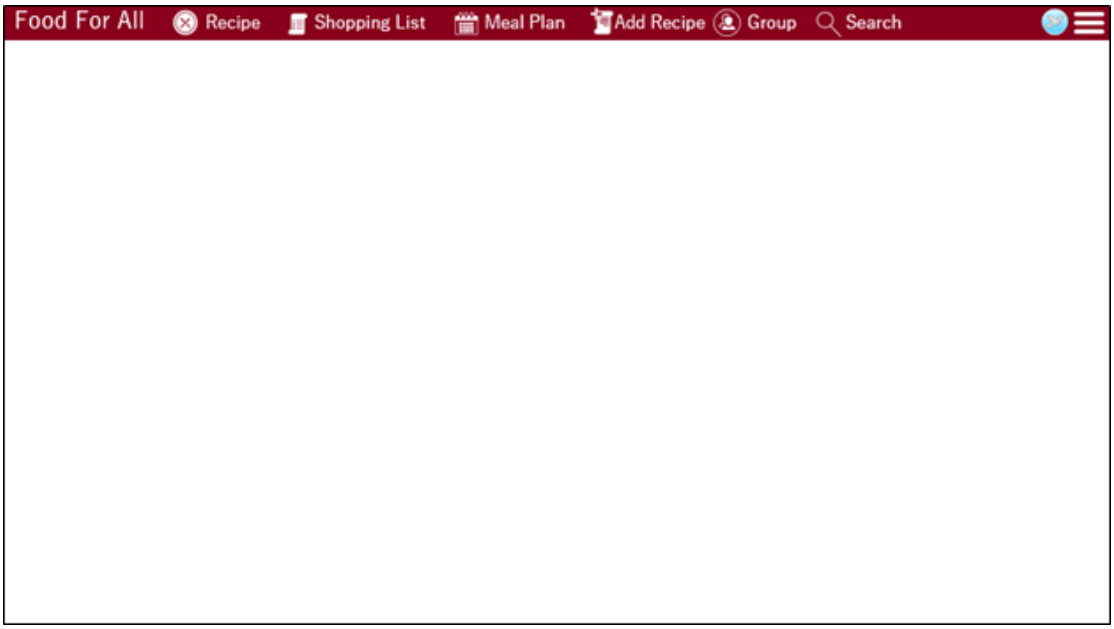


Figure 5.8.1.1: User Interface - Navigation View for Desktop

5.8.1.2 Mobile Profile View

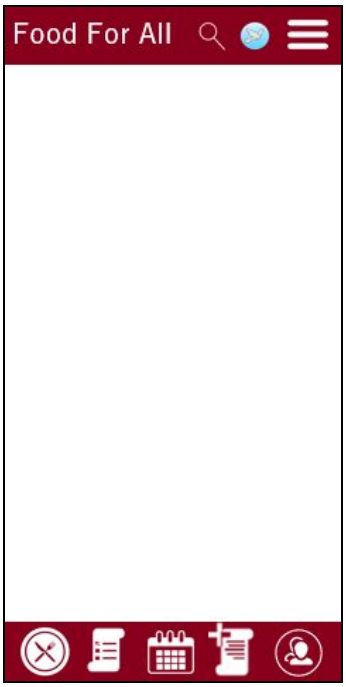


Figure 5.8.1.2: User Interface - Navigation View for mobile

5.8.1.3 Mobile Landscape View

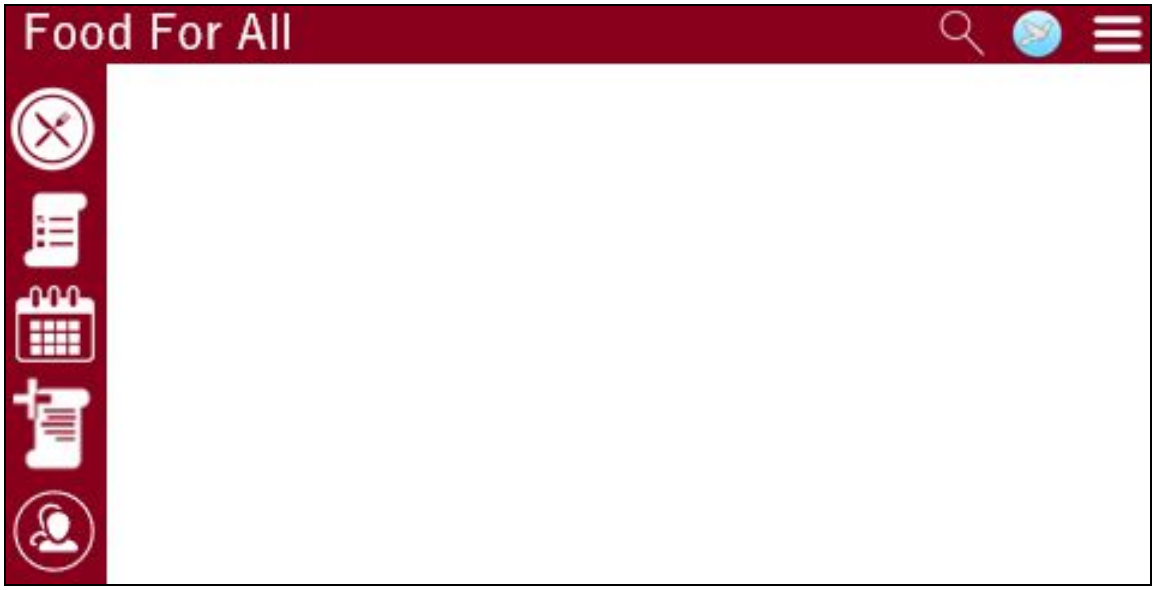


Figure 5.8.1.3: User Interface - Navigation View for mobile landscape

5.8.1.4 Navigation Buttons

The navigation buttons views are the basic overview of what the view recipes button (see figure 5.8.1.4.1), the add/create recipe button (see figure 5.8.1.4.2), the view shopping list button (see figure 5.8.1.4.3), the view meal planner button (see figure 5.8.1.4.4), the hamburger button (see figure 5.8.1.4.5), the group button (see figure 5.8.1.4.6), the profile icon button (see figure 5.8.1.4.7), and the search button (see figure 5.8.1.4.8) each should look like.


View Recipes SRS 3.5.4.1

Figure 5.8.1.4.1: User Interface -View Recipes


Add/Create Recipe SRS 3.7.1.1

Figure 5.8.1.4.2: User Interface - Add/Create Recipe


View Shopping List SRS 3.5.4.2

Figure 5.8.1.4.3: User Interface -View Shopping List


View Meal Planner SRS 3.5.4.3

Figure 5.8.1.4.4: User Interface - View Meal Planner


Hamburger

Figure 5.8.1.4.5: User Interface - Hamburger


Group SRS 3.5.4.4

Figure 5.8.1.4.6: User Interface - Group


Profile Icon

Figure 5.8.1.4.7: User Interface - Profile Icon


Search SRS 3.2.10

Figure 5.8.1.4.8: User Interface - Search

5.8.2 Login

The login view is what the login page should look like in desktop(see figure 5.8.2.2), mobile profile(see figure 5.8.2.3), and mobile landscape (see figure 5.8.2.4) views.

5.8.2.1 This is a form that will allow the user to login SRS 3.7.2.1

5.8.2.1.1 Username Input Field

5.8.2.1.2 Password Input Field

5.8.2.1.3 Submission Button

5.8.2.1.4 Reset Password Button

5.8.2.2 Desktop View

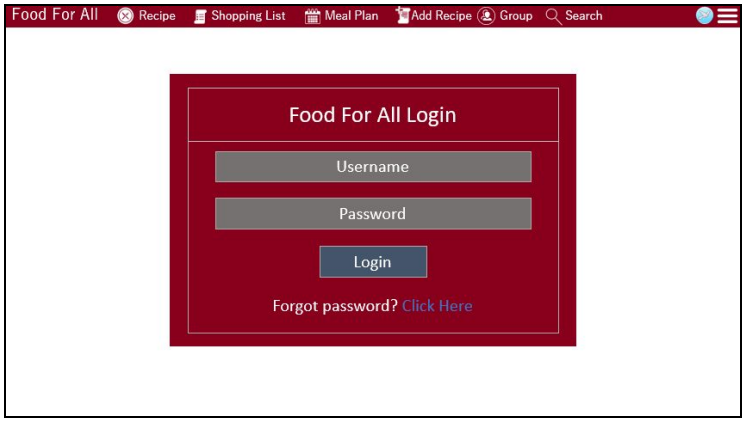


Figure 5.8.2.2: User Interface - Login View for Desktop

5.8.2.3 Mobile Profile View

Figure 5.8.2.3: User Interface - Login View for mobile

5.8.2.4 Mobile Landscape View

Figure 5.8.2.4: User Interface - Login View for mobile landscape

5.8.3 Hamburger

The hamburger view is what the hamburger should look like in desktop(see figure 5.8.3.2), mobile profile(see figure 5.8.3.3), and mobile landscape (see figure 5.8.3.4) views.

5.8.3.1 This is a button that holds hidden setting options.

5.8.3.1.1 User Account SRS 3.5.4

This link sends the user to the user account page so they can edit their profile picture and add or remove recipes.

5.8.3.1.2 Preferences SRS 3.6.6.1

This link takes the user to the preferences page where they will be able to adjust things such as color blind settings.

5.8.3.1.3 Notifications SRS 3.7.1.2

This link gives the user the option to enable or disable group notifications.

5.8.3.1.4 App Rating SRS 3.5.4.1

This link will take the user to the App Store where they will be able to rate the application.

5.8.3.1.5 Log Out SRS 3.7.2.1

This link logs the user out of their profile and send them to the base version of the website without profile options.

5.8.3.2 Desktop View

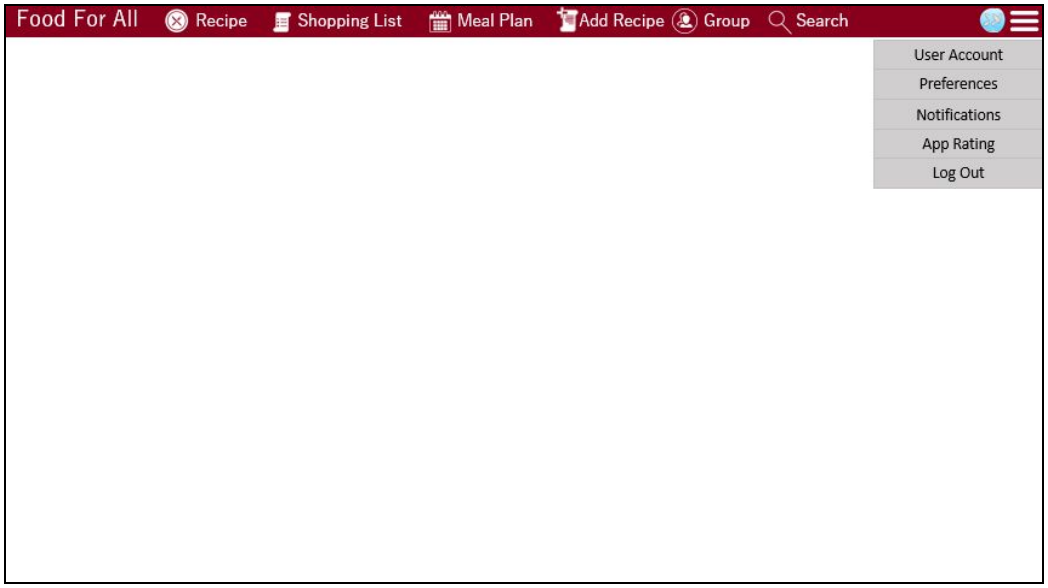


Figure 5.8.3.2: User Interface - Hamburger View for Desktop

5.8.3.3 Mobile Profile View

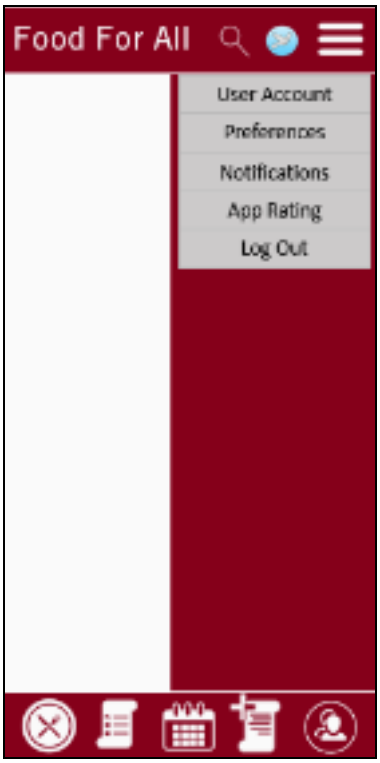


Figure 5.8.3.3: User Interface - Hamburger View for mobile

5.8.3.4 Mobile Landscape View

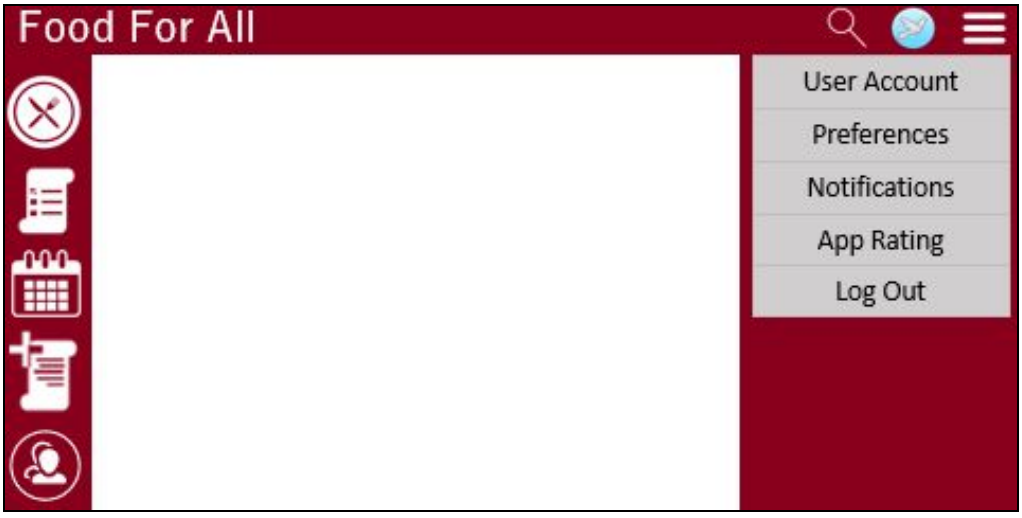


Figure 5.8.3.4: User Interface - Hamburger View for mobile landscape

5.8.4 Preferences

The preferences view is what the preferences should look like in desktop(see figure 5.8.4.2), mobile profile(see figure 5.8.4.3), and mobile landscape (see figure 5.8.4.4) views.

5.8.4.1 Theme - This will include options for colorblind modes SRS 3.6.6.1

5.8.4.2 Desktop View

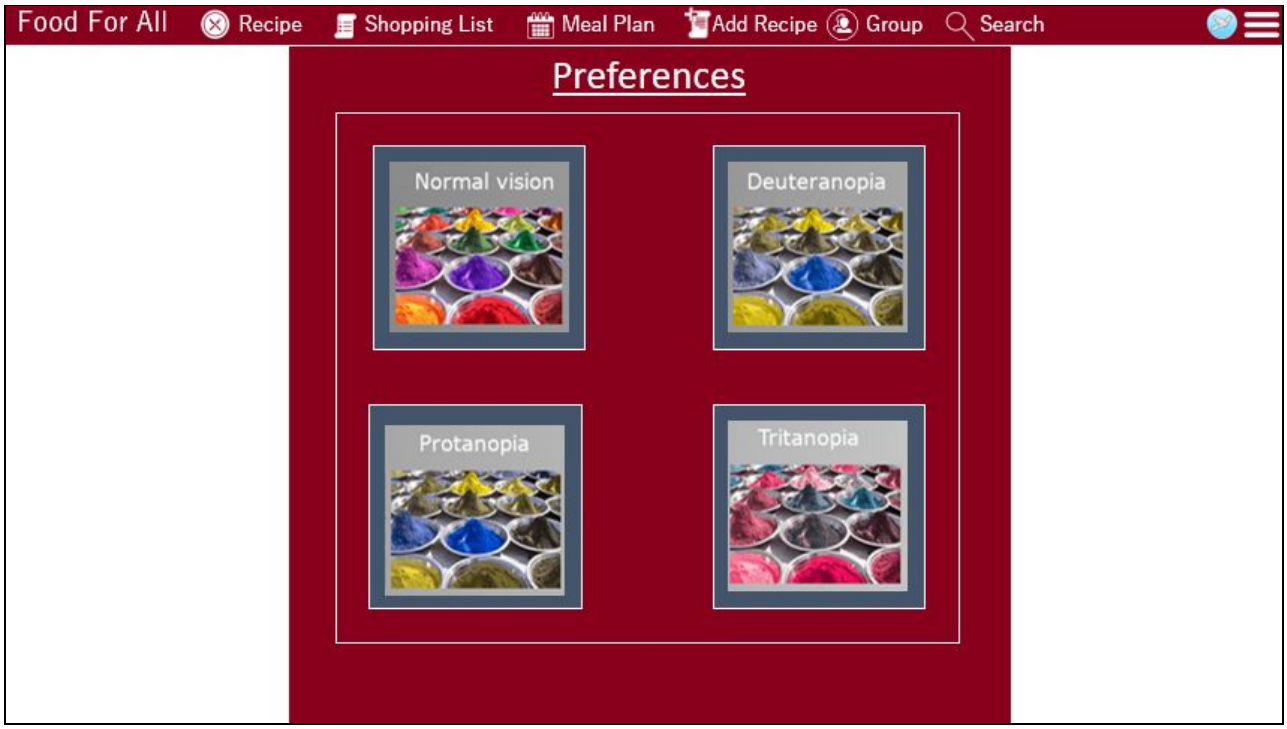


Figure 5.8.4.2: User Interface - Preferences View for Desktop

5.8.4.3 Mobile Profile View

Figure 5.8.4.3: User Interface - Preferences View for mobile

5.8.4.4 Mobile Landscape View

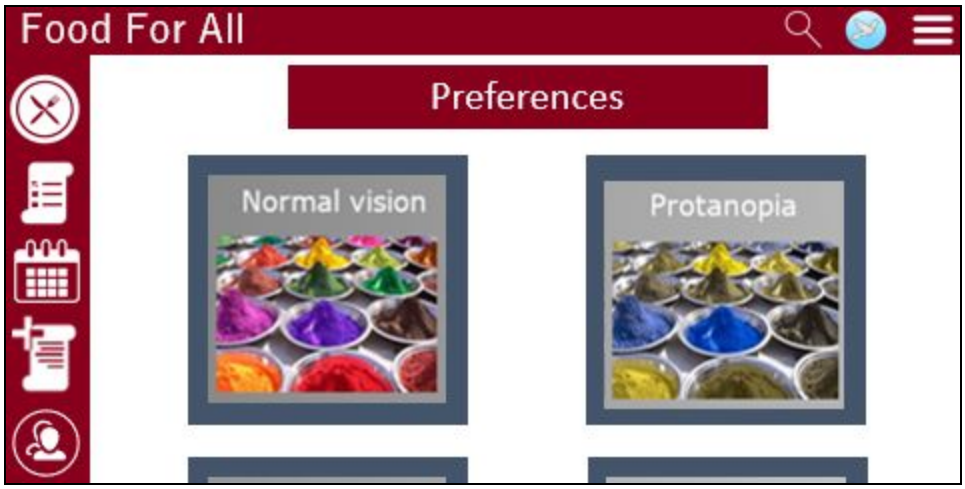


Figure 5.8.4.4: User Interface - Preferences View for mobile landscape

5.8.5 Account Settings

The account view is what account settings should look like in desktop(see figure 5.8.5.2), mobile profile(see figure 5.8.5.3), and mobile landscape (see figure 5.8.5.4) views.

5.8.5.1 This is a page is where the user can change their profile info. SRS 3.5.4

5.8.5.1.1 Username

5.8.5.1.2 Password

5.8.5.1.3 Profile Picture

5.8.5.2 Desktop View

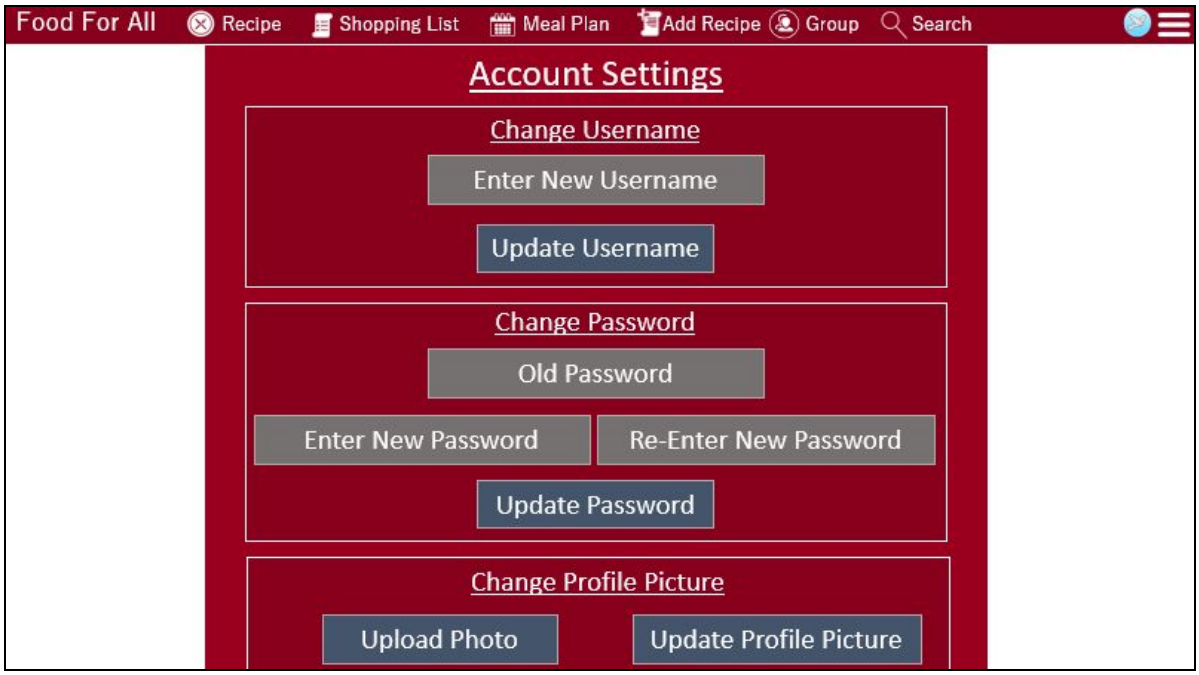


Figure 5.8.5.2: User Interface - Account Settings View for Desktop

5.8.5.3 Mobile Profile View

Figure 5.8.5.3: User Interface - Account Settings View for mobile

5.8.5.4 Mobile Landscape View

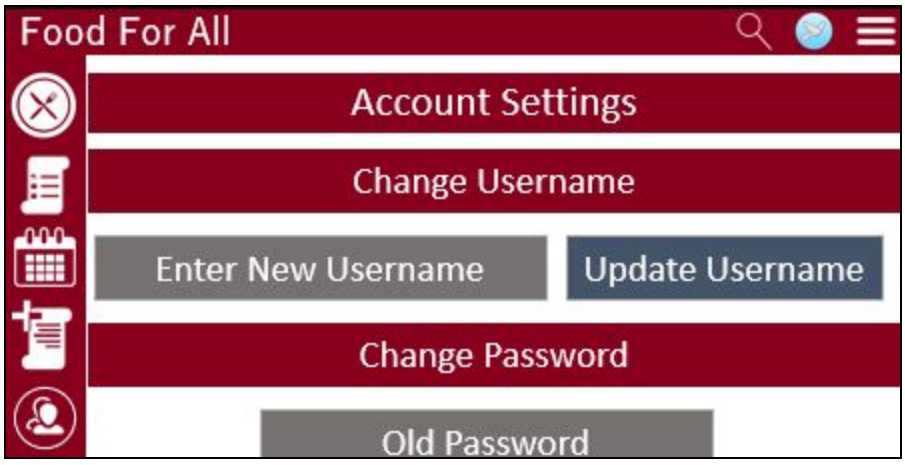


Figure 5.8.5.4: User Interface - Account Settings View for mobile landscape

5.8.6 Groups/Friends Tab

This is what the groups/friends tab should look like in desktop (see figure 5.8.6.2.1), mobile profile (see figure 5.8.6.2.2), and mobile landscape (see figure 5.8.6.2.3) views.

5.8.6.1 This is a tab that shows all the users groups and friends SRS 3.5.4.4

5.8.6.1.1 Add friend

5.8.6.1.2 Remove friend

5.8.6.1.3 View friend's profile

5.8.6.2 List of groups and friends

5.8.6.2.1 Desktop View

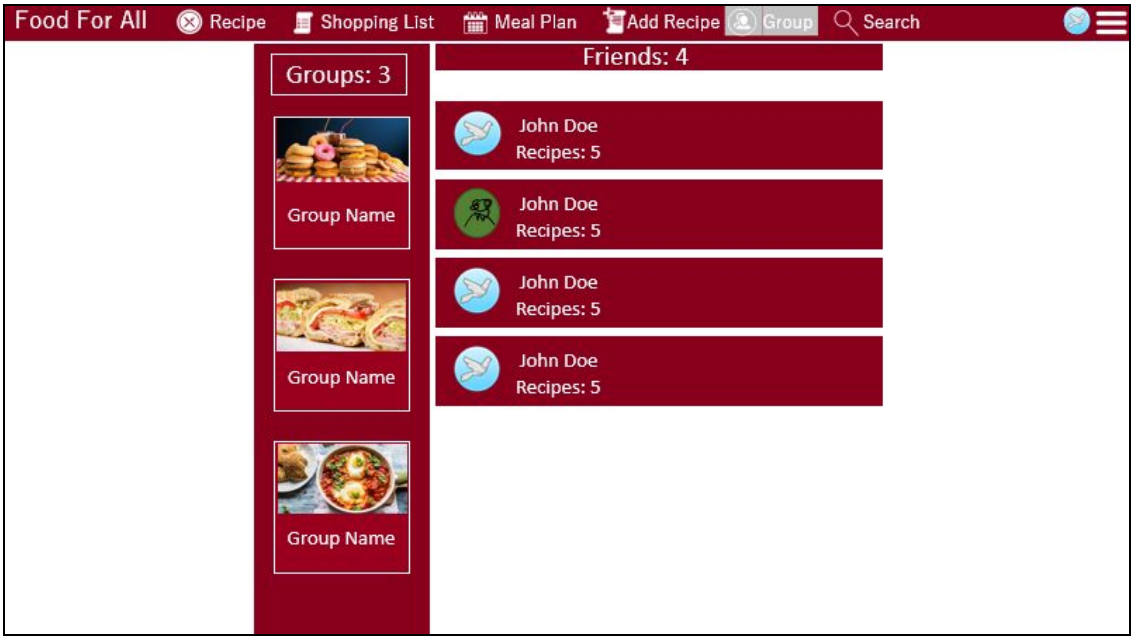


Figure 5.8.6.2.1: User Interface - Group/Friends Tab View for Desktop

5.8.6.2.2 Mobile Profile View

Figure 5.8.6.2.2: User Interface - Group/Friends Tab View for mobile

5.8.6.2.3 Mobile Landscape View

Figure 5.8.6.2.3: User Interface - Group/Friends Tab View for mobile landscape

5.8.6.3 Group Recipes Page

The groups view is what the group recipes page should look like in desktop(see figure 5.8.6.3.1), mobile profile(see figure 5.8.6.3.2), and mobile landscape (see figure 5.8.6.3.3) views.

5.8.6.3.1 Desktop View

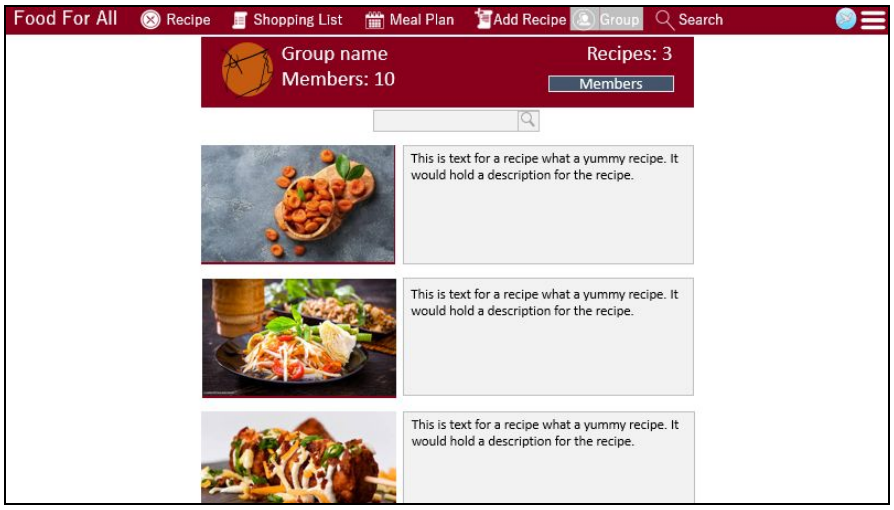


Figure 5.8.6.3.1: User Interface - Group Recipes View for Desktop

5.8.6.3.2 Mobile Profile View

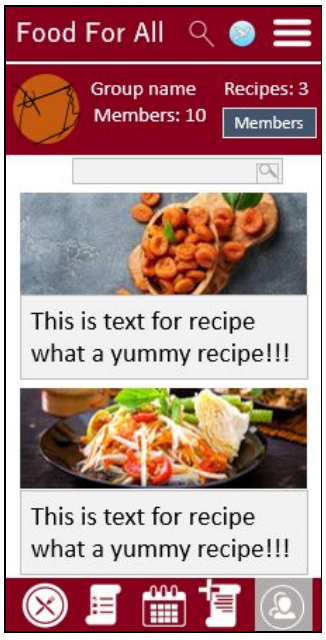


Figure 5.8.6.3.2: User Interface - Group Recipes View for mobile

5.8.6.3.3 Mobile Landscape View

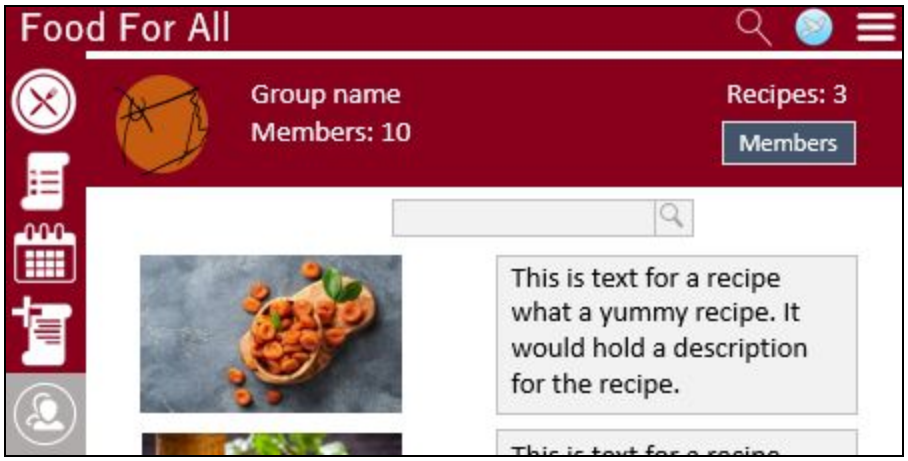


Figure 5.8.6.3.3: User Interface - Group Recipes View for mobile landscape

5.8.6.4 List of group members

The group member list view is what the list of group members should look like in desktop(see figure 5.8.6.4.1), mobile profile(see figure 5.8.6.4.2), and mobile landscape (see figure 5.8.6.4.3) views.

5.8.6.4.1 Desktop View

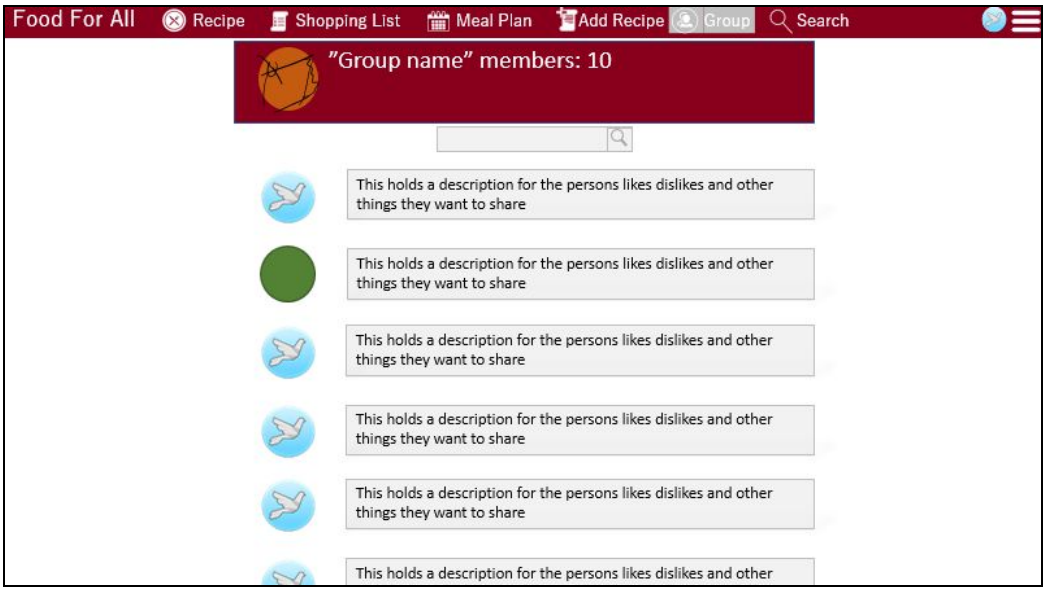


Figure 5.8.6.4.1: User Interface - Group Members View for Desktop

5.8.6.4.2 Mobile Profile View

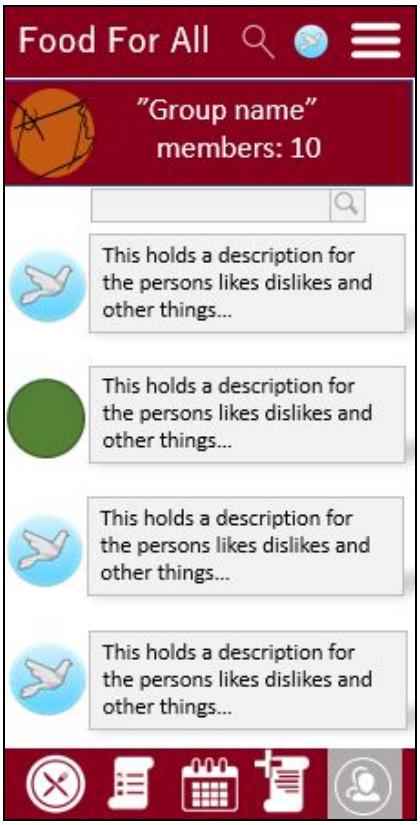


Figure 5.8.6.4.2: User Interface - Group Members View for mobile

5.8.6.4.3 Mobile Landscape View

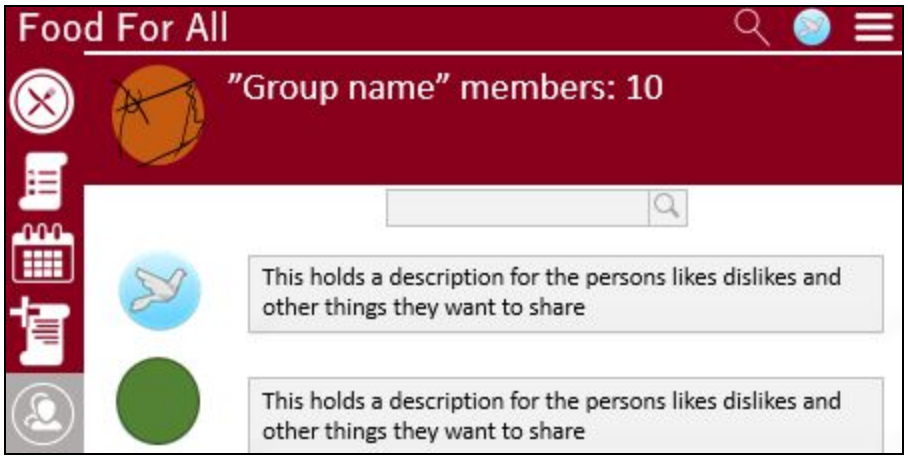


Figure 5.8.6.4.3: User Interface - Group Members View for mobile landscape

5.8.6.5 Friend Profile Page

The friend profile view is what the friend profile should look like in desktop(see figure 5.8.6.5.1), mobile profile(see figure 5.8.6.5.2), and mobile landscape (see figure 5.8.6.5.3) views.

5.8.6.5.1 Desktop View

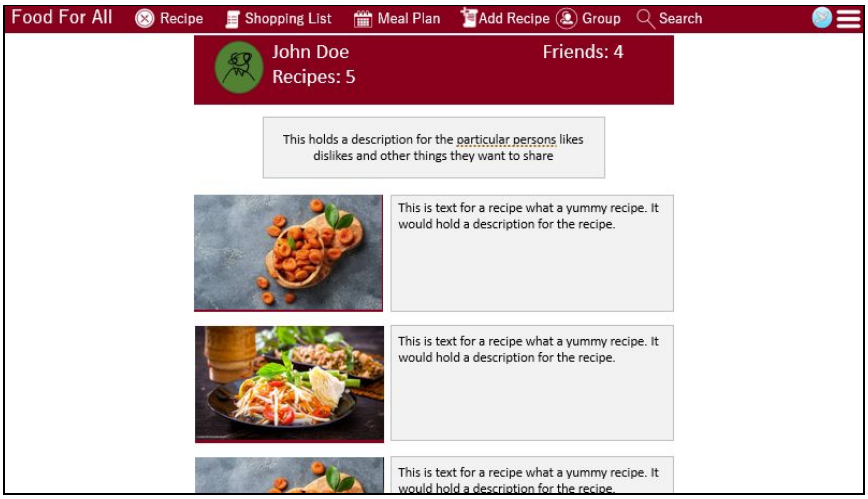


Figure 5.8.6.5.1: User Interface - Friend Profile View for Desktop

5.8.6.5.2 Mobile Profile View

Figure 5.8.6.5.2: User Interface - Friend profile View for mobile

5.8.6.5.3 Mobile Landscape View

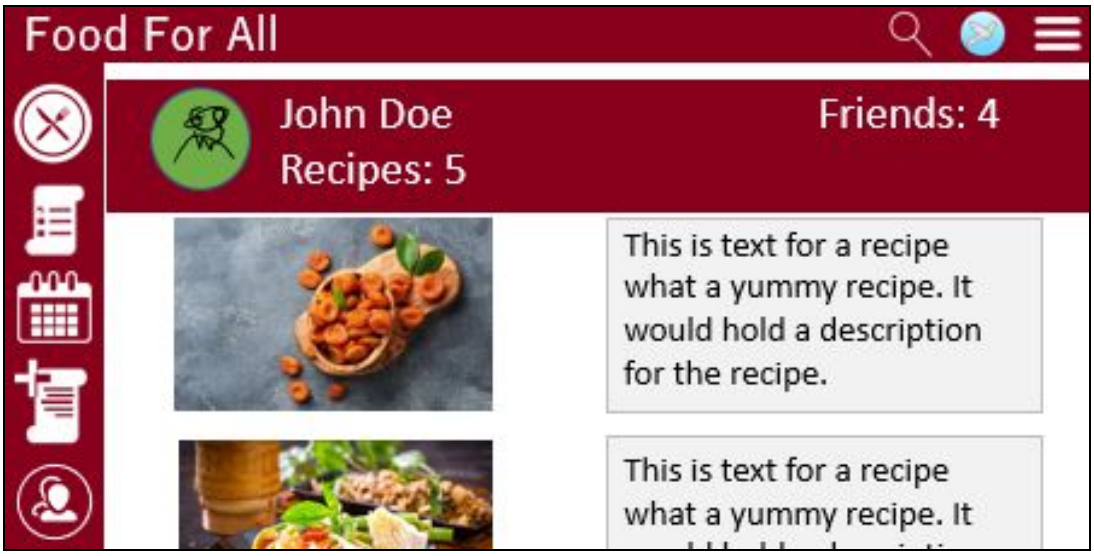


Figure 5.8.6.5.3: User Interface - Friend Profile View for mobile landscape

5.8.7 Upload/Create Recipe

The upload/create recipe view will show what a upload/create page should look like in desktop(see figure 5.8.7.2), mobile profile(see figure 5.8.7.3), and mobile landscape (see figure 5.8.7.4) views.

5.8.7.1 Upload recipe SRS 3.7.1.1

This page will allow for the uploading of new recipes.

5.8.7.1.1 Copy existing recipe

This will allow the user to make a copy of an existing recipe.

5.8.7.1.2 New recipe

5.8.7.2 Desktop View

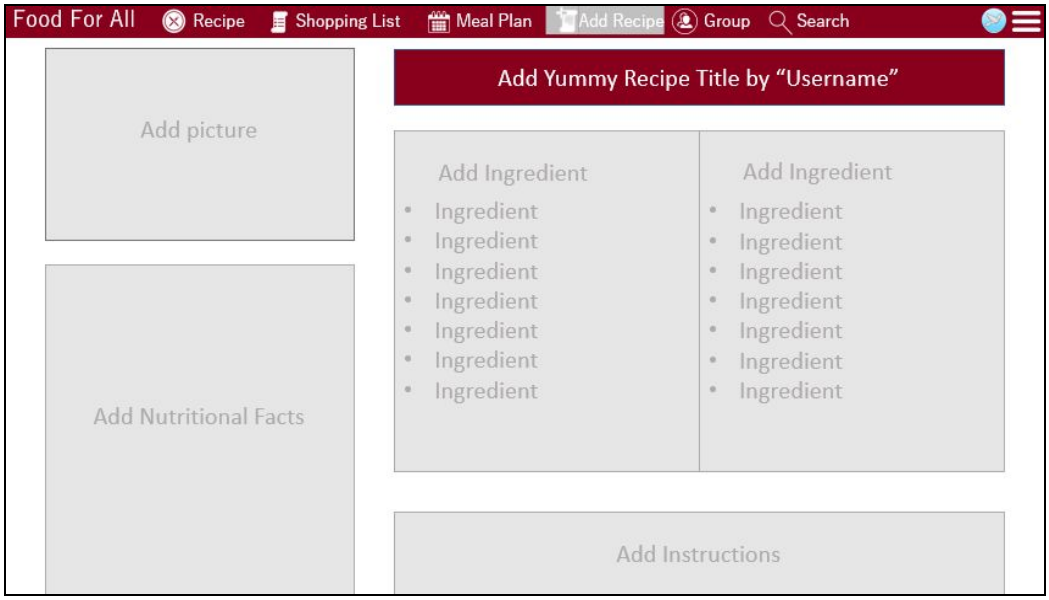


Figure 5.8.7.2: User Interface - Upload/Create Recipe View for Desktop

5.8.7.3 Mobile Profile View

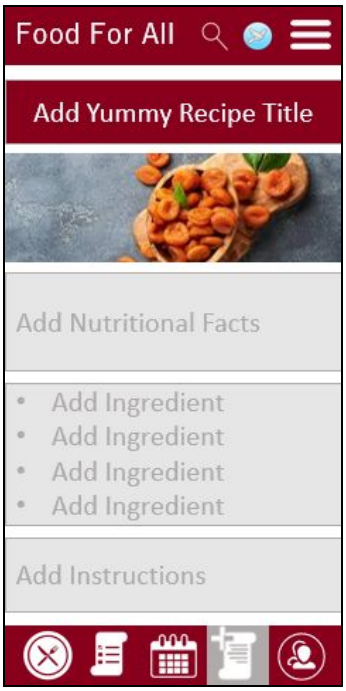


Figure 5.8.7.3: User Interface - Upload/Create Recipe View for mobile

5.8.7.4 Mobile Landscape View

Figure 5.8.7.4: User Interface - Upload/Create Recipe View for mobile landscape

5.8.8 Recipe

The Recipe view will show what a list of recipes should look like in desktop(see figure 5.8.8.4.2), mobile profile(see figure 5.8.8.4.3), and mobile landscape (see figure 5.8.8.4.4) views.

5.8.8.1 This page will contain a recipe SRS 3.5.4.1

5.8.8.1.1 Title

5.8.8.1.2 Author

5.8.8.1.3 Rating

5.8.8.1.4 Photo

5.8.8.1.5 Serving size

5.8.8.1.6 Ingredients

5.8.8.1.7 Instructions

5.8.8.1.8 Notes

5.8.8.2 It will allow a user to save/bookmark the recipe to save in favorites. SRS 3.5.4.4

5.8.8.2.1 Save/bookmark

5.8.8.2.2 Add to favorites

5.8.8.3 It will allow the user a button to “modify” the recipe and save it as their own SRS 3.1.1.2.1.3

5.8.8.3.1The new recipe should give credit to the original

5.8.8.4 All Recipes

5.8.8.4.1 The user will be able to sort the recipes.

5.8.8.4.2 Desktop View

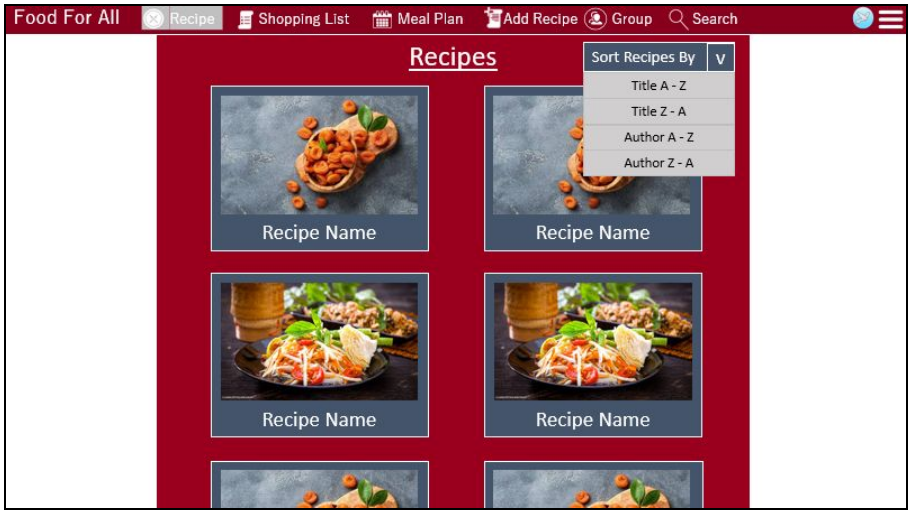


Figure 5.8.8.4.2: User Interface - Recipe View for Desktop

5.8.8.4.3 Mobile Profile View

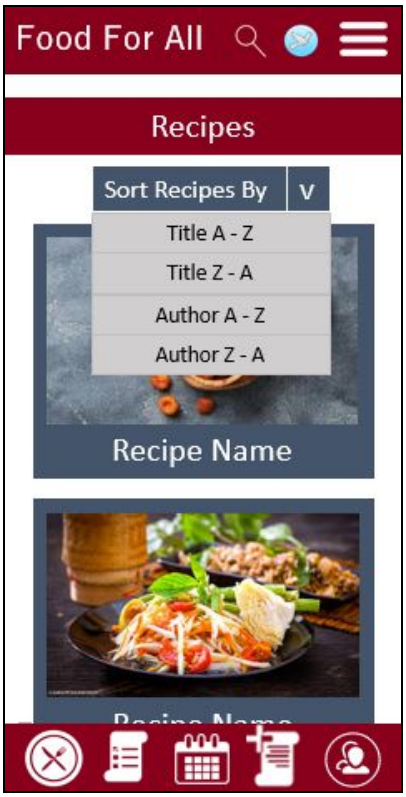


Figure 5.8.8.4.3: User Interface - Recipe View for mobile

5.8.8.4.4 Mobile Landscape View

Figure 5.8.8.4.4: User Interface - Recipe View for mobile landscape

5.8.8.5 Individual Recipe

The Individual Recipe view will show what a single recipe should look like in desktop(see figure 5.8.8.5.1), mobile profile(see figure 5.8.8.5.2), and mobile landscape (see figure 5.8.8.5.3) views.

5.8.8.5.1 Desktop View

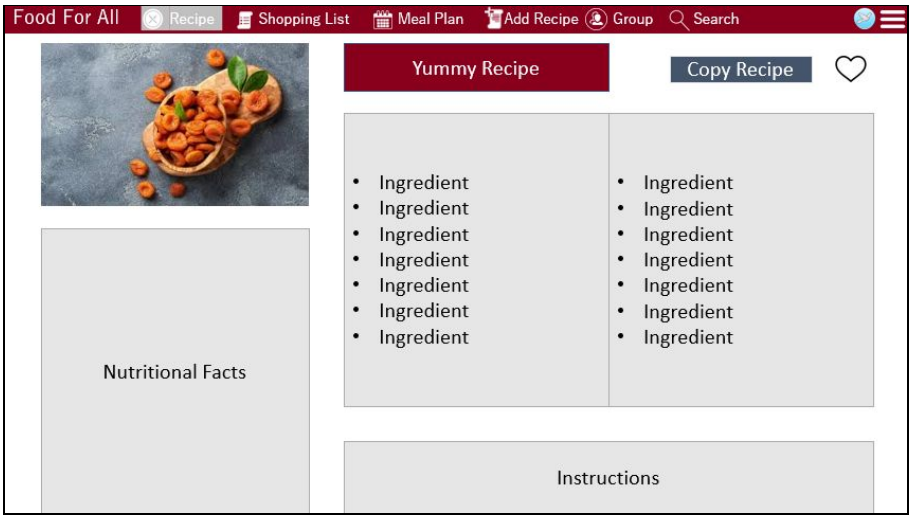


Figure 5.8.8.5.1: User Interface - Individual Recipe View for Desktop

5.8.8.5.2 Mobile Profile View

Figure 5.8.8.5.2: User Interface - Individual Recipe View for mobile

5.8.8.5.3 Mobile Landscape View

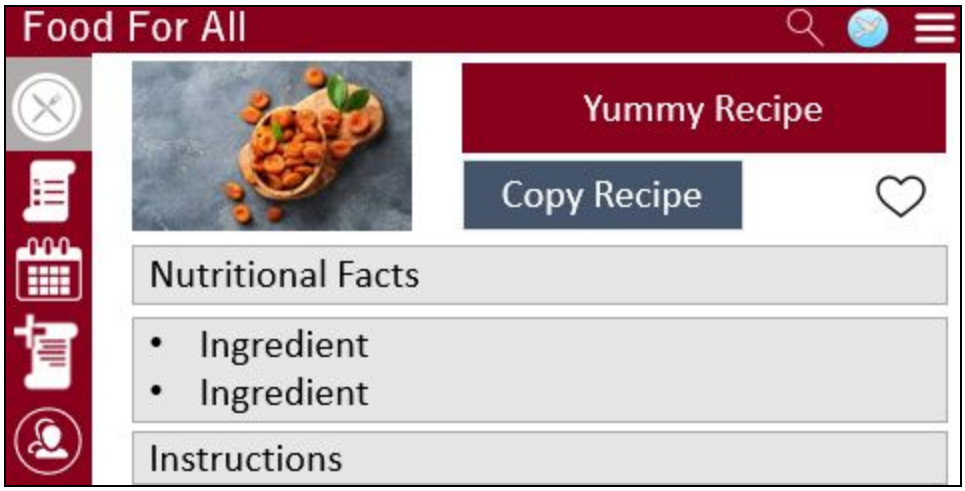


Figure 5.8.8.5.3: User Interface - Individual Recipe View for mobile landscape

5.8.9 Meal Plan

The Meal Plan view will allow the user to see what meals they have planned on a daily and weekly basis.

5.8.9.1 This page will include a form for planning your meals SRS 3.5.4.3

5.8.9.1.1 Daily meals

5.8.9.1.2 Weekly meals

5.8.9.2 Daily Meal Plan

The daily meal plan will show what a daily meal plan should look like in desktop(see figure 5.8.9.2.1), mobile profile(see figure 5.8.9.2.2), and mobile landscape (see figure 5.8.9.2.3) views.

5.8.9.2.1 Desktop View

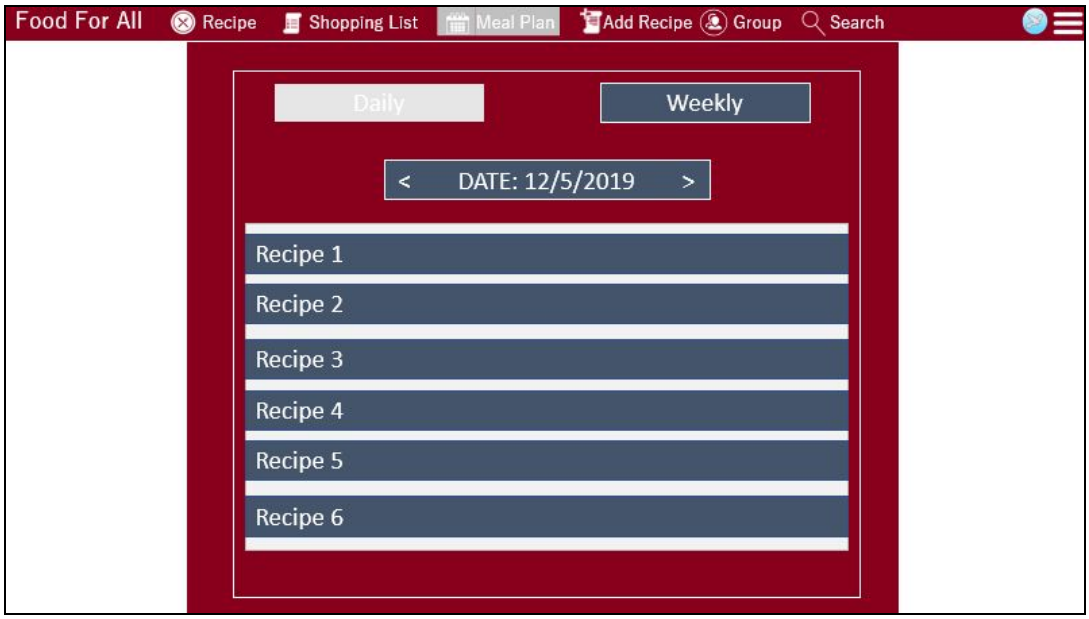


Figure 5.8.9.2.1: User Interface - Daily Meal Plan View for Desktop

5.8.9.2.2 Mobile Profile View

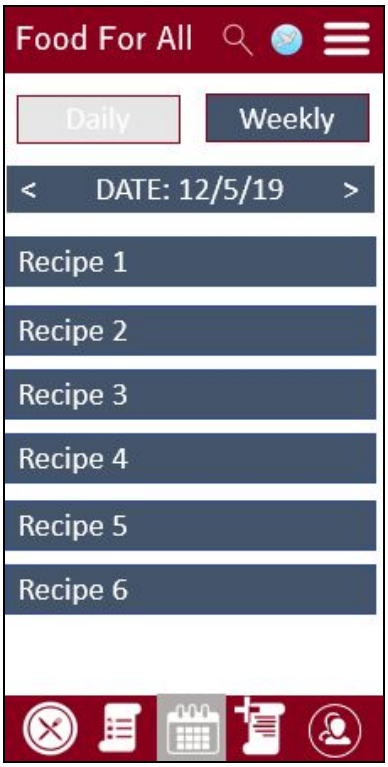


Figure 5.8.9.2.2: User Interface - Daily Meal Plan View for mobile

5.8.9.2.3 Mobile Landscape View

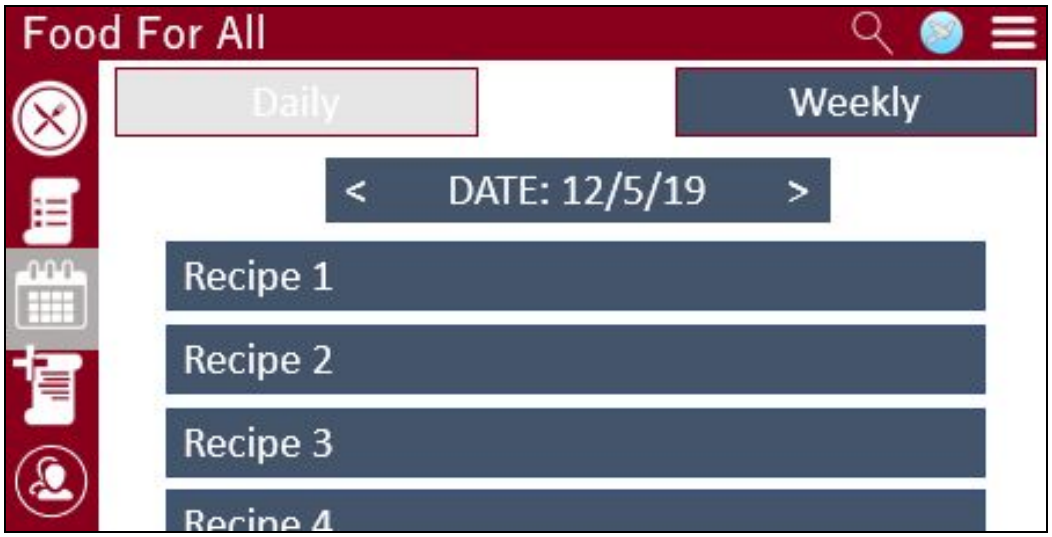


Figure 5.8.9.2.3: User Interface - Daily Meal Plan View for mobile landscape

5.8.9.3 Weekly Meal Plan

The weekly meal plan view is what the weekly meal plan should look like in desktop(see figure 5.8.9.3.1), mobile profile(see figure 5.8.9.3.2), and mobile landscape (see figure 5.8.9.3.3) views.

5.8.9.3.1 Desktop View

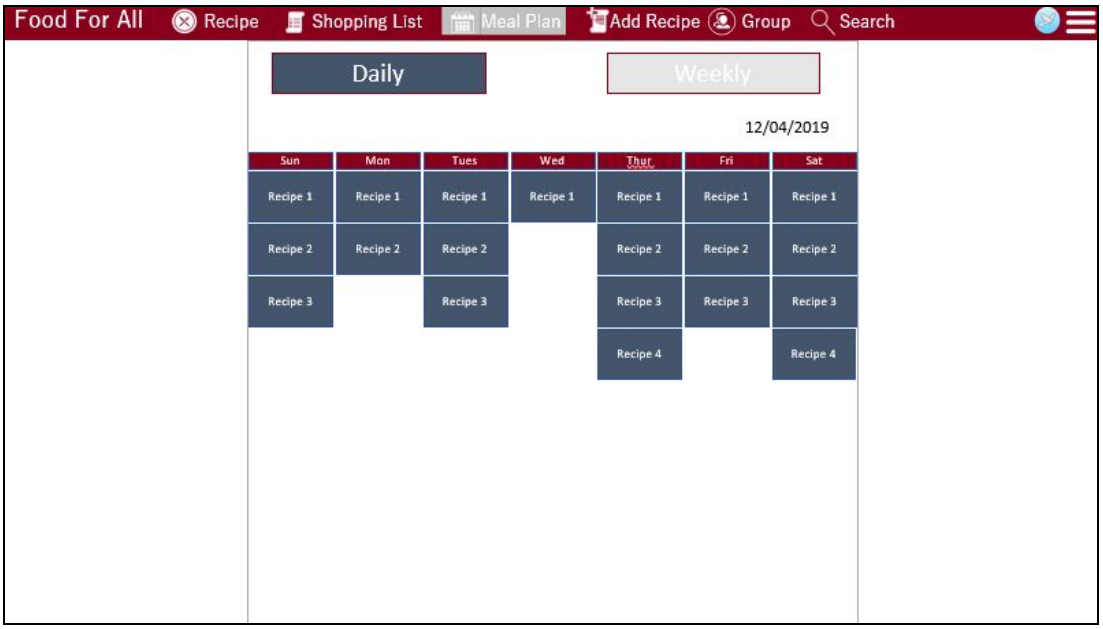


Figure 5.8.9.3.1: User Interface - Weekly Meal Plan View for Desktop

5.8.9.3.2 Mobile Profile View

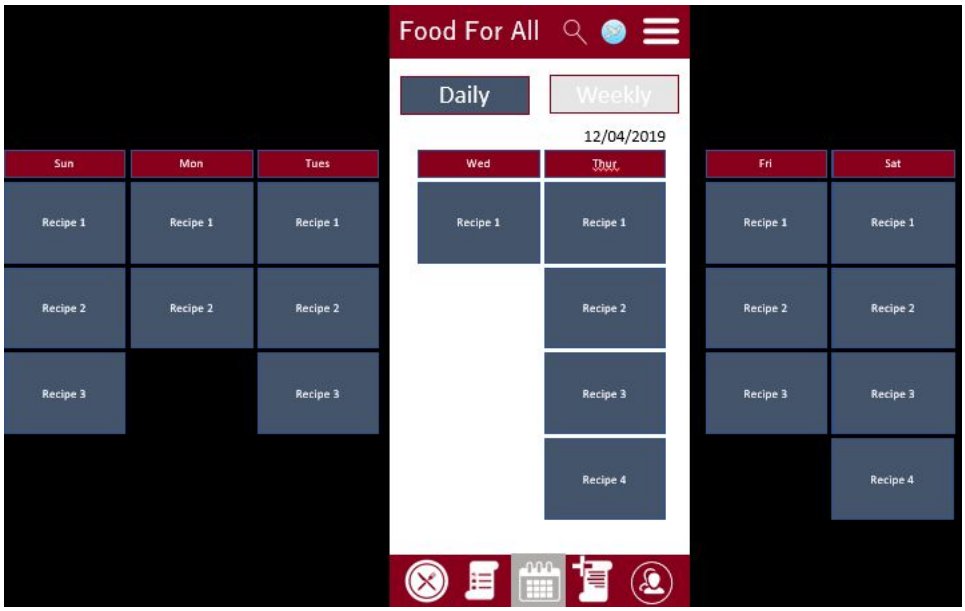


Figure 5.8.9.3.2: User Interface - Weekly Meal Plan View for mobile

5.8.9.3.3 Mobile Landscape View

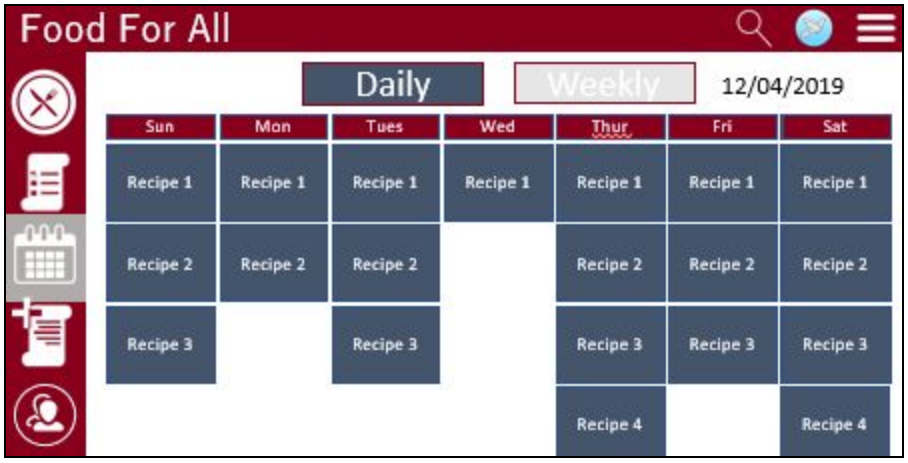


Figure 5.8.9.3.3: User Interface - Weekly Meal Plan View for mobile landscape

5.8.10 User Profile SRS 3.5.4

The User Profile view will show what a user profile should look like in desktop(see figure 5.8.10.3), mobile profile(see figure 5.8.10.4), and mobile landscape (see figure 5.8.10.5) views.

5.8.10.1 This page will contain the personal recipes uploaded by the user.

5.8.10.2 Contains the ability to make account public or private.

5.8.10.3 Desktop View

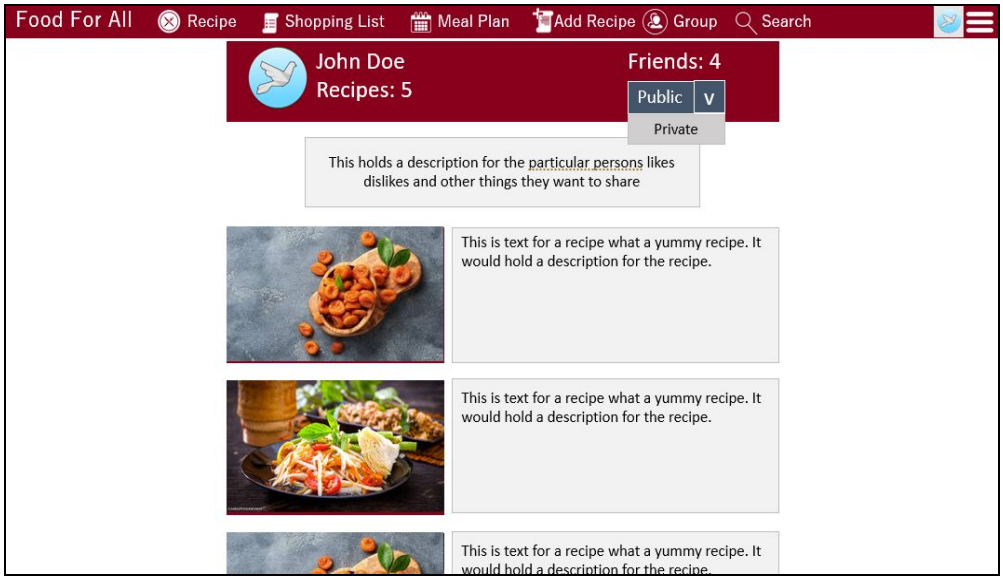


Figure 5.8.10.3: User Interface - User Profile View for Desktop

5.8.10.4 Mobile Profile View

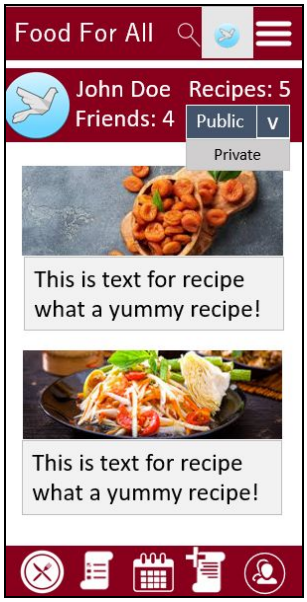


Figure 5.8.10.4: User Interface - User Profile View for mobile

5.8.10.5 Mobile Landscape View

Figure 5.8.10.5: User Interface - User Profile View for mobile landscape

5.8.11 Shopping List SRS 3.5.4.2

The Shopping list view will show what a shopping list should look like in desktop(see figure 5.8.11.2), mobile profile(see figure 5.8.11.3), and mobile landscape (see figure 5.8.11.4) views.

5.8.11.1 This page will contain a list of items to be purchased by the user.

5.8.11.2 Desktop View

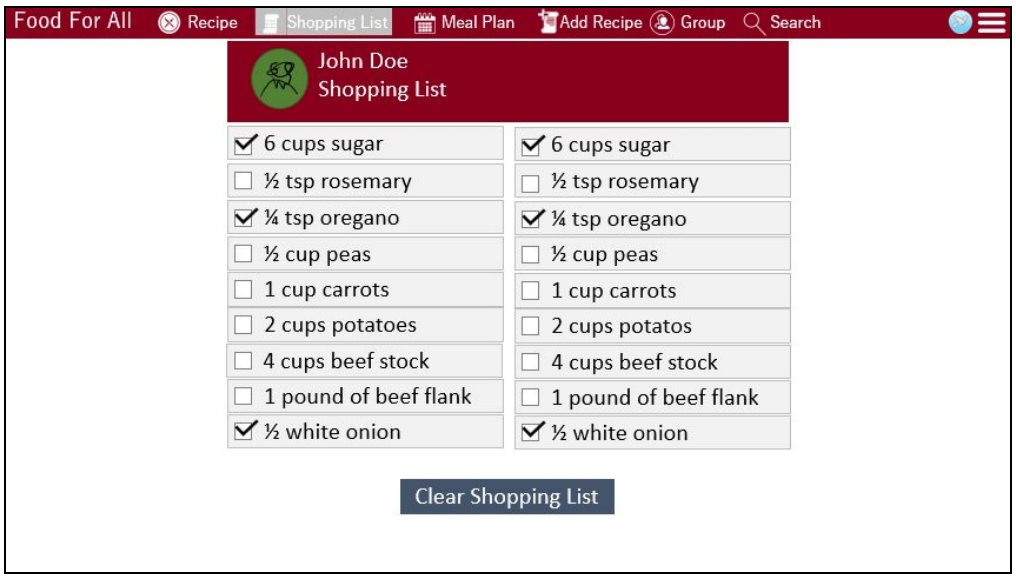


Figure 5.8.11.2: User Interface - Shopping List View for Desktop

5.8.11.3 Mobile Profile View

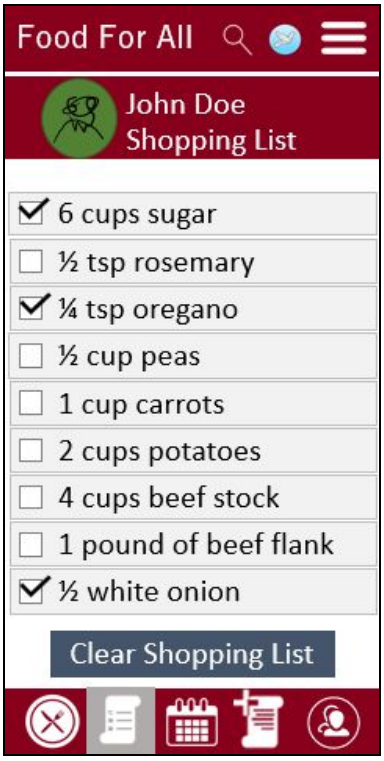


Figure 5.8.11.3: User Interface -Shopping List View for mobile

5.8.11.4 Mobile Landscape View

Figure 5.8.11.4: User Interface -Shopping List View for mobile landscape

5.8.12 Search

The search view will show what the search should look like in desktop(see figure 5.8.12.2), mobile profile(see figure 5.8.12.3), and mobile landscape (see figure 5.8.12.4) views.

5.8.12.1 This page will allow the user to search for specific recipes. SRS 3.2.10

5.8.12.1.1 Filter search results by title SRS 3.2.10.1

5.8.12.1.2 Filter search results by author SRS 3.2.10.2

5.8.12.1.3 Filter search results by ingredients SRS 3.2.10.3

5.8.12.1.4 Filter search results by keywords SRS 3.2.10.4

5.8.12.2 Desktop View

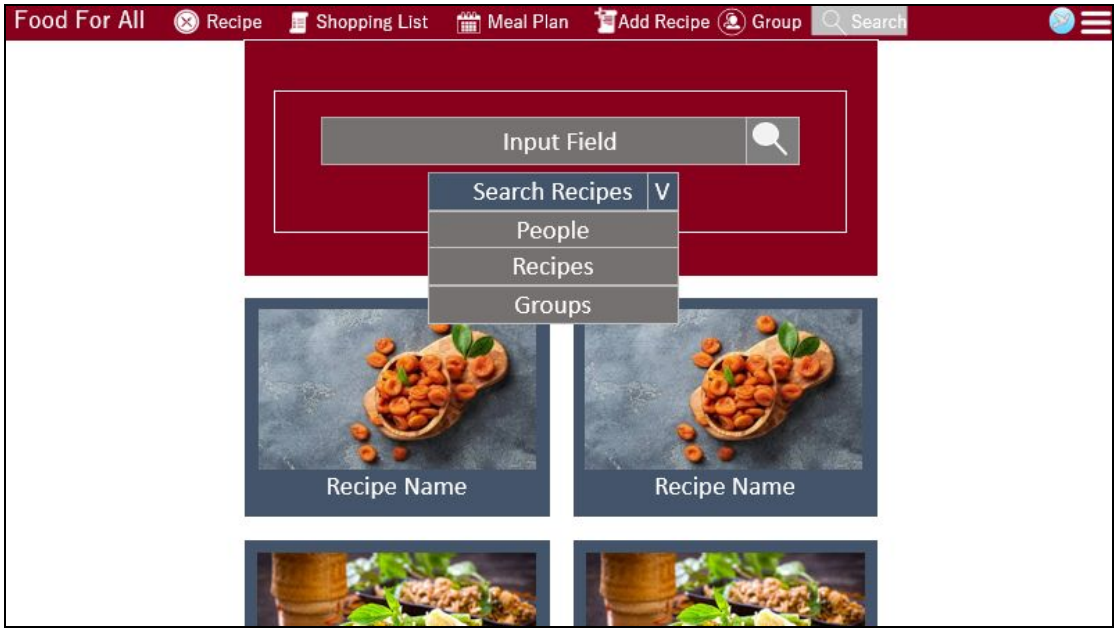


Figure 5.8.12.2: User Interface - Search View for Desktop

5.8.12.3 Mobile Profile View

Figure 5.8.12.3: User Interface - Search View for mobile

5.8.12.4 Mobile Landscape View

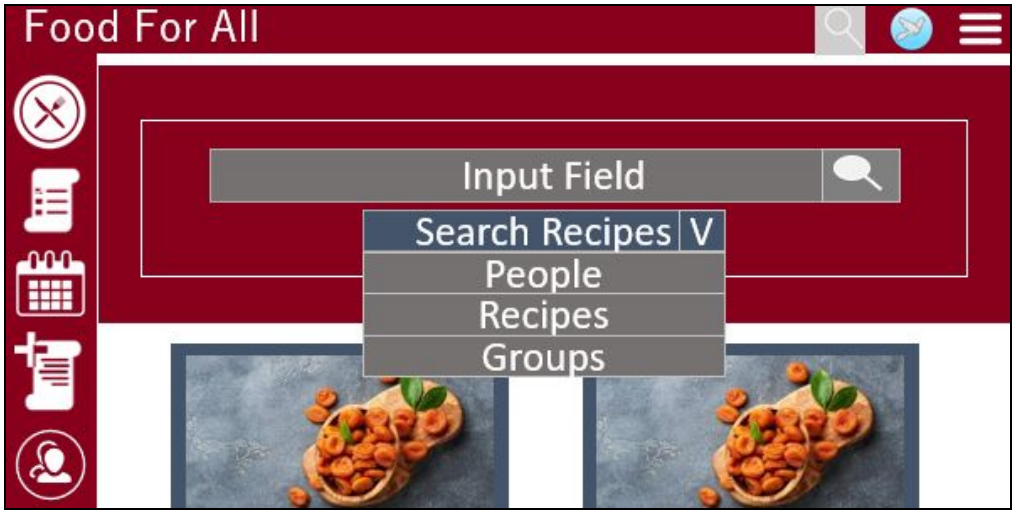


Figure 5.8.12.4: User Interface - Search View for mobile landscape

5.9 Structure

The Structure viewpoint is described in IEEE Std 1016-2009 on page 20, section 5.9.The purpose of the Structure viewpoint is to document the internal constituents and organization of the design. For sections 5.9.1 - 5.9.13 refer to Figure 5.9.

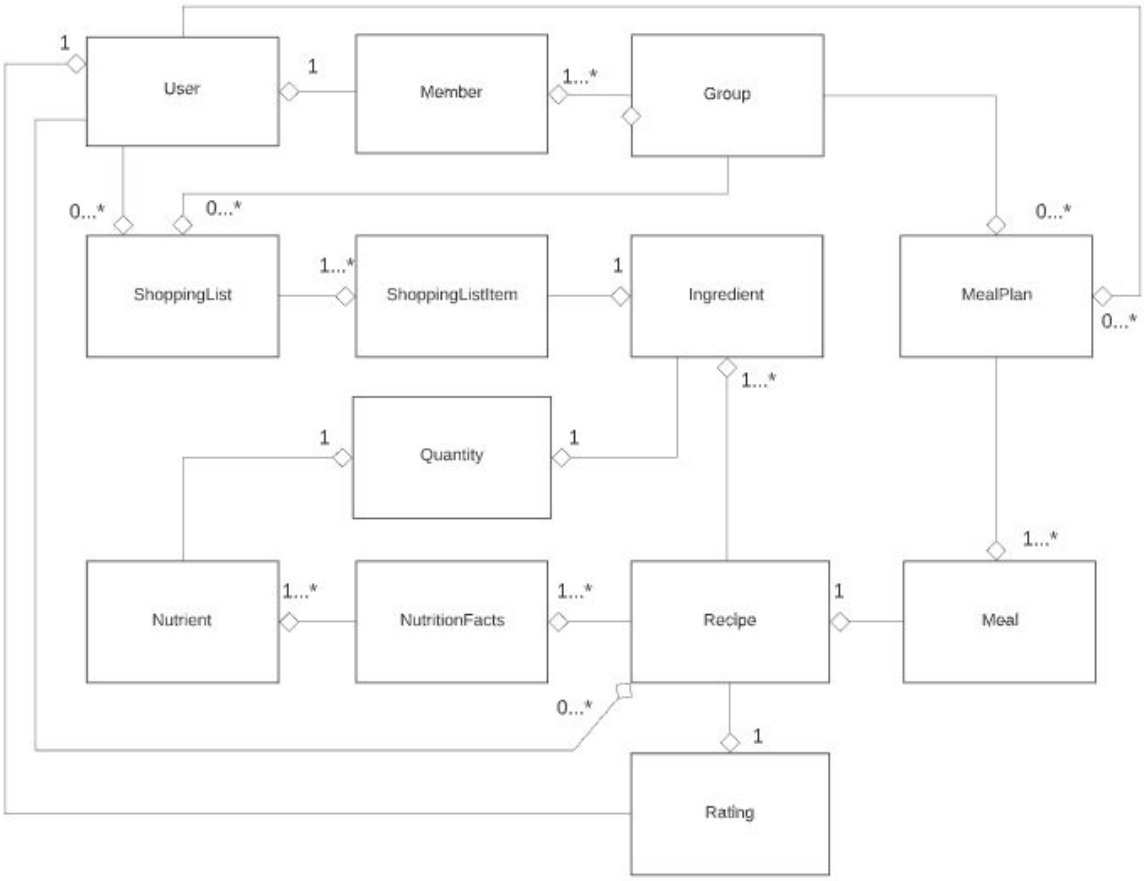


Figure 5.9 - Food For All Structure Diagram

5.9.1 User - See section SDD 5.4.1.1 for more details on the class.

5.9.1.1 User ‘has-a’ list of Recipes

5.9.1.2 User ‘has-a’ MealPlan

5.9.1.3 User ‘has-a’ ShoppingList

5.9.2 Member - See section SDD 5.4.1.13 for more details on the class.

5.9.2.1 Member ‘has-a’ User.

5.9.3 Group - See section SDD 5.4.1.2 for more details on the class.

5.9.3.1 Group ‘has-a’ list of Members.

5.9.4 Quantity - See section SDD 5.4.1.11 for more details on the class.

5.9.5 Ingredient - See section SDD 5.4.1.7 for more details on the class.

5.9.5.1 Ingredient ‘has-a’ Quantity

5.9.6 ShoppingListItem - See section SDD 5.4.1.6 for more details on the class.

5.9.6.1 ShoppingListItem ‘has-a’ Ingredient

5.9.7 ShoppingList - See section SDD 5.4.1.5 for more details on the class.

5.9.7.1 ShoppingList ‘has-a’ list of ShoppingListItems

5.9.8 Nutrient - See section SDD 5.4.1.10 for more details on the class.

5.9.8.1 Nutrient ‘has-a’ Quantity

5.9.9 NutritionFacts - See section SDD 5.4.1.11 for more details on the class.

5.9.9.1 NutritionFacts ‘has-a’ List of Nutrient

5.9.10 Rating - See section SDD 5.4.1.4 for more details on the class.

5.9.10.1 Rating ‘has-a’ User

5.9.11 Recipe - See section SDD 5.4.1.3 for more details on the class.

5.9.11.1 Recipe ‘has-a’ list of NutritionFacts

5.9.11.2 Recipe ‘has-a’ Rating

5.9.11.3 Recipe ‘has-a’ list of Ingredients

5.9.12 Meal - See section SDD 5.4.1.8 for more details on the class.

5.9.12.1 Meal ‘has-a’ Recipe.

5.9.13 MealPlan -See section SDD 5.4.1.8 for more details on the class.

5.9.13.1 Meal Plan ‘has-a’ List of Meals

5.10 Interaction

The interactions viewpoint is described in IEEE std 1016-2009 on page 20, section 5.10. The primary concern of the interactions viewpoint is defining the interactions among entities and the strategies employed to achieve them.

5.10.1 Sort and Filter

The diagram below titled “Interaction Search and Filter” is designated to portray the minimal interactions required of the system to perform a search and/or filter function for any of the following when exchanged accordingly: title (SRS 3.2.10.2.1 , SDD 5.2.2.11), author (SRS 3.2.10.2.2, SDD 5.2.2.12), ingredients (SRS 3.2.10.2.3, SDD 5.2.2.13), keyword (SRS 3.2.10.2.4), prep/cook time (SRS 3.2.10.2.5), tags (SRS 3.2.10.2.6, SDD 5.2.2.9), and rating (SRS 3.2.10.2.7, SDD 5.2.2.10), main ingredient (SRS 3.8.4.6, SDD 5.2.2.13), favorites (SRS 3.8.4.6, SDD 5.6.1.5), ratings (SRS 3.8.4.6, SDD 5.2.2.10), and seasonal (SRS 3.8.4.6).

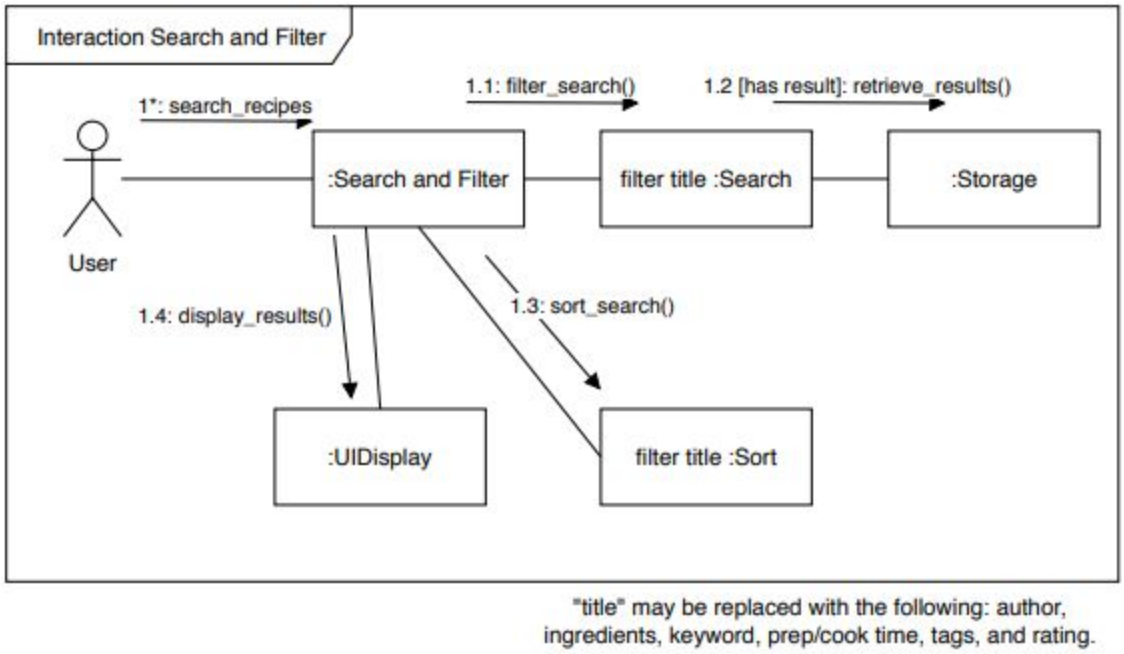


Figure 5.10.1.1. Interaction - Search and Filter

5.10.2 Database

Section 5.10.2 offers a high level view of interactions between the user and the database.

5.10.2.1 Insert to Database

Interaction: Insert to Database is represented below (SRS 3.1.3.2.1.1, SDD 5.2.2.1, SDD 5.14.7). This diagram represents server database storage, which allows the user to insert (or

POST when a web app) information to a database.

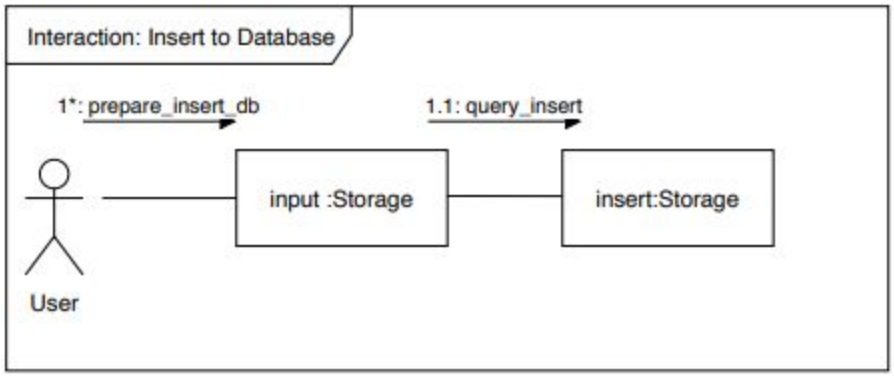


Figure 5.10.2.1.1. Interaction - Insert to Database

5.10.2.2 Select from Database

Interaction: Select from Database is represented below (SRS 3.1.3.2.1.2, SDD 5.14.7). This diagram represents server database retrieval, which has the function of returning stored recipes from the database that match or contains the keyword passed into the function (SDD 5.2.2.8).

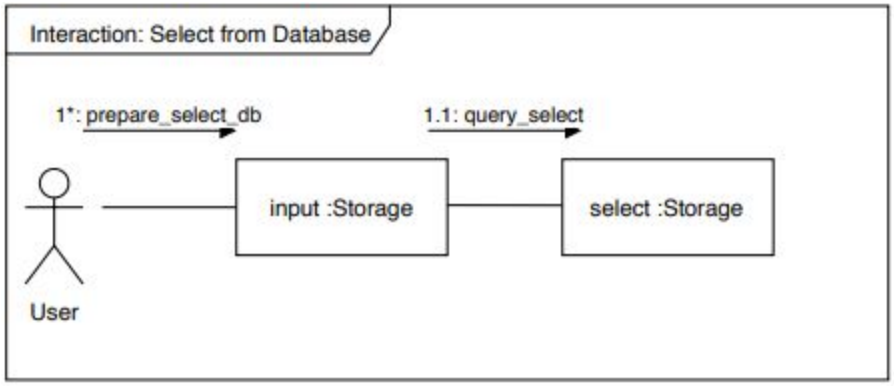


Figure 5.10.2.2.1. Interaction - Select from Database

5.10.2.3 Update Database

Interaction: Update Database is represented below (SRS 3.1.3.2.1.3, SDD 5.14.7). This diagram represents the modifiable server database, which consists of potentially multiple functions which are to call update queries on existing database rows to modify a recipe and/or user data (SDD 5.2.2.1).

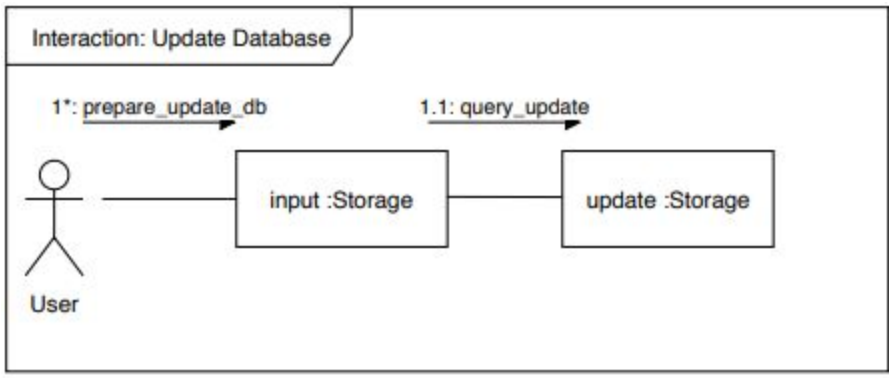


Figure 5.10.2.3.1. Interaction - Update Database

5.10.3 Users and Groups

Section 5.10.3 demonstrates interactions between various entities pertaining to users, groups (SDD 5.11.2.10), group permissions, and logging in.

5.10.3.1 Follow User

Interaction: Follow User is represented below (SRS 3.2.2.1, SDD 5.2.2.49). This diagram represents the function user/follow user, which uses a method of the user class and allows a user to “follow” another user through their id.

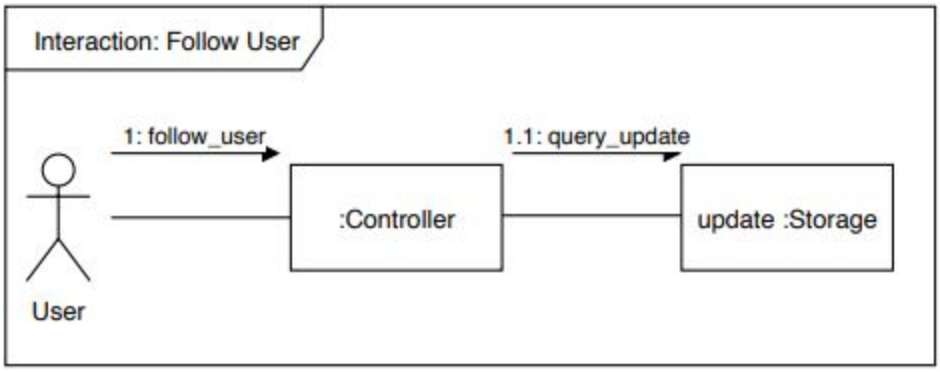


Figure 5.10.3.1.1. Interaction - Follow User

5.10.3.2 Unfollow User

Interaction: Unfollow User is represented below (SRS 3.2.2.1, SDD 5.2.2.50). This diagram represents the function user/unfollow user, which uses a method of the user class and allows a user to “unfollow” another user via their id..

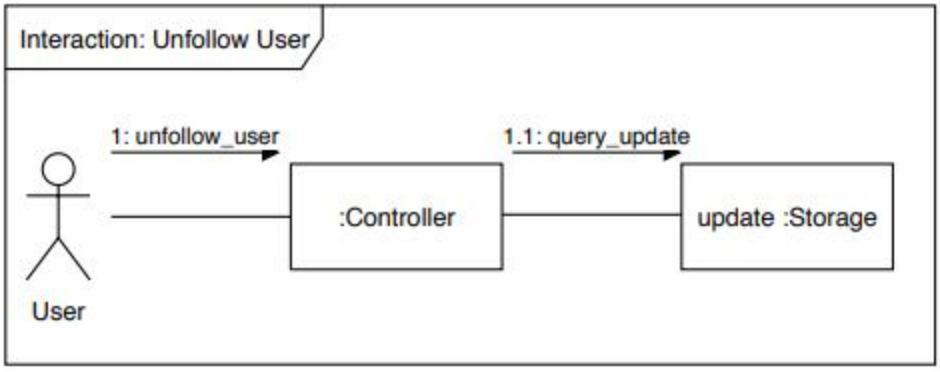


Figure 5.10.3.2.1. Interaction - Unfollow User

5.10.3.3 Create Group

Interaction: Create Group is represented below (SRS 3.2.2.3, SDD 5.2.2.39). This diagram represents the function create group, which allows a user to create a new group.

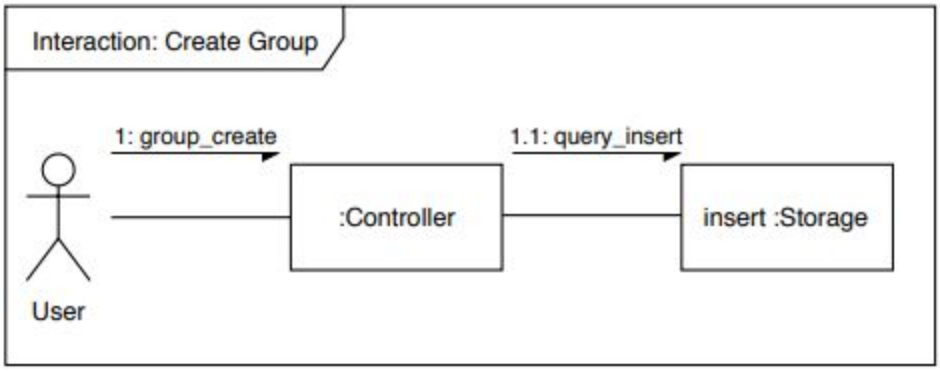


Figure 5.10.3.3.1. Interaction - Create Group

5.10.3.4 Add to Group

Interaction: Add to Group is represented below (SRS 3.2.2.3.1, SDD 5.2.2.40). This diagram represents the add user function of group, which allows the admin of the group to add or remove a given user from the group.

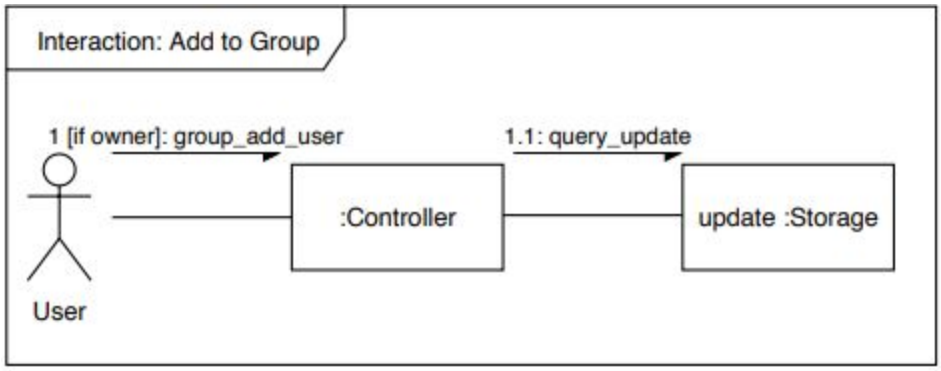


Figure 5.10.3.4.1. Interaction - Add to Group

5.10.3.5 Remove from Group

Interaction: Remove from Group is represented below (SRS 3.2.2.3.1, SDD 5.2.2.41). This diagram represents the remove user function of group, which allows the admin of the group to add or remove a given user from the group.

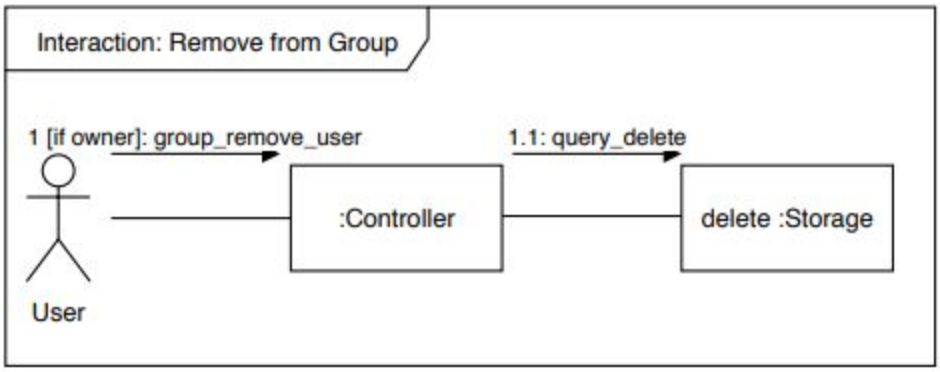


Figure 5.10.3.6.1. Interaction - Remove from Group

5.10.3.6 Change Group User Permission

Interaction: Change Group User Permission is represented below (SRS 3.2.2.3.2, SRS 3.2.11.1.4). This diagram represents the function update user permissions from group, which allows the admin of the group to change individual group member read and/or write permissions.

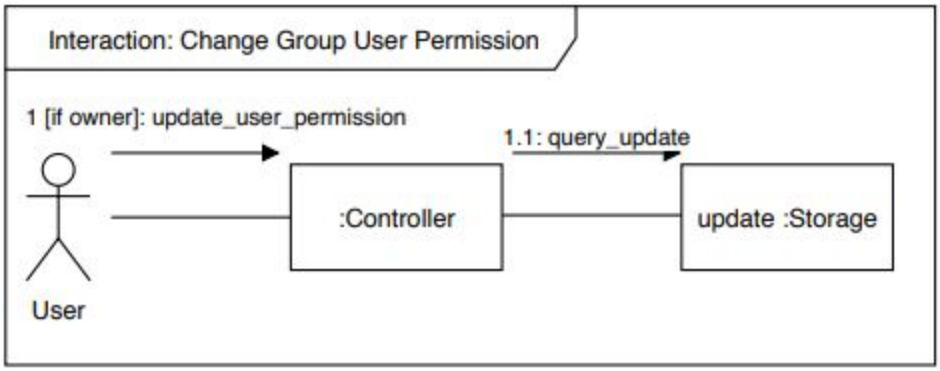


Figure 5.10.3.6.1. Interaction - Change Group User Permission

5.10.3.7 User Sign-in Session

Interaction: User Sign-in Session is represented below (SRS 3.7.2.1, SRS 3.4.2, SRS 3.7.2.1). This diagram represents the user connecting with their session and thus being able to access their associated objects and permissions, this may also be referred to as the login process for

the user. This occurs through Google (SDD 5.2.2.44) which uses Firebase (SDD 5.6.1.3) as their authentication tool.

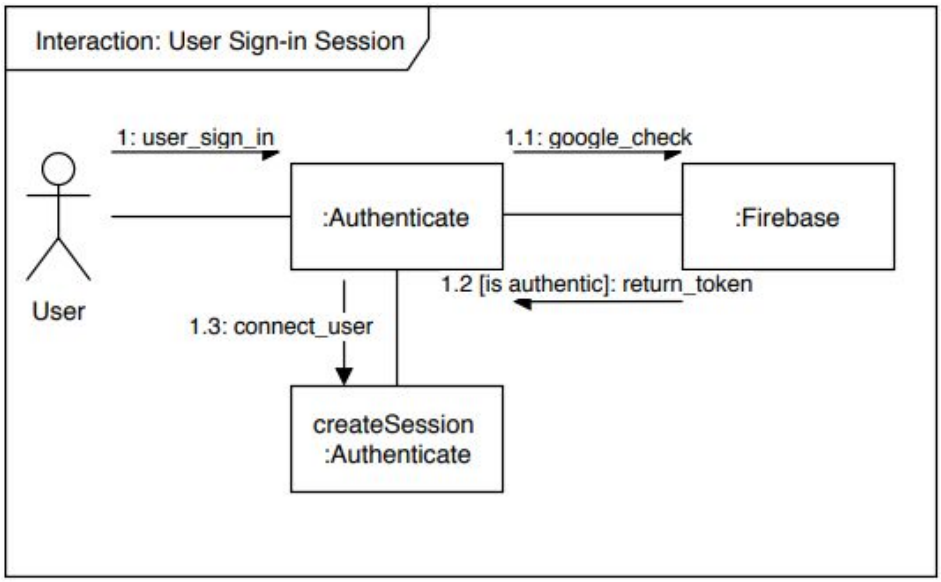


Figure 5.10.3.7.1. Interaction - User Sign-in Session

5.10.3.8 Notification

Interaction: Notification is represented below (SRS 3.7.1.2, SDD 5.11.1). This diagram represents the interactions which occur when a user attempts to upload a recipe to the database. The end of this interaction results in a pop-up notification created using HTML, that informs the user if the recipe was uploaded successfully or not.

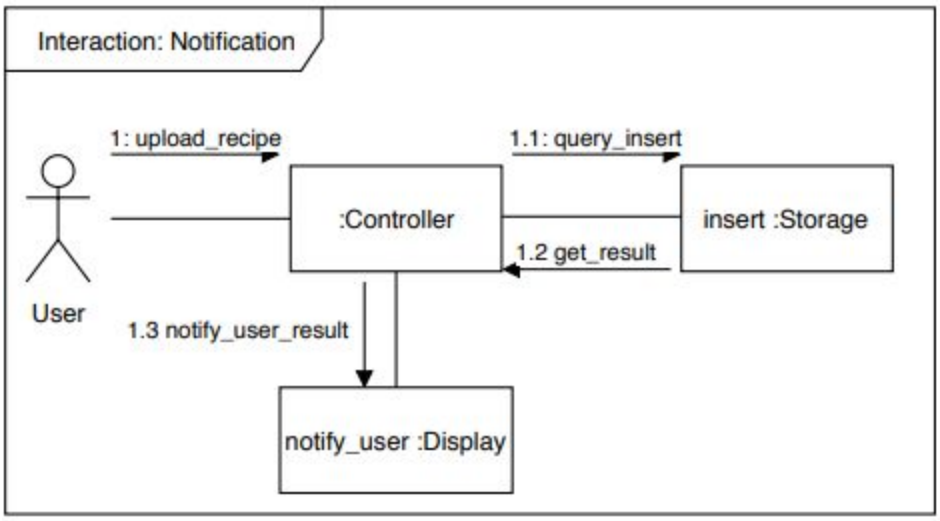


Figure 5.10.3.8.1. Interaction - Notification

5.10.4 Recipe Functionality

This section demonstrates interactions between recipe functionality entities such as saving, deleting, updating and so on. Recipe functionality as pertaining to this section refers to the functions a recipe class may perform.

5.10.4.1 Delete

Interaction: Delete is represented below (SRS 3.2.13.2, SRS 3.2.5.1). This diagram represents the interactions which occur when a user deletes an ingredient from a recipe.

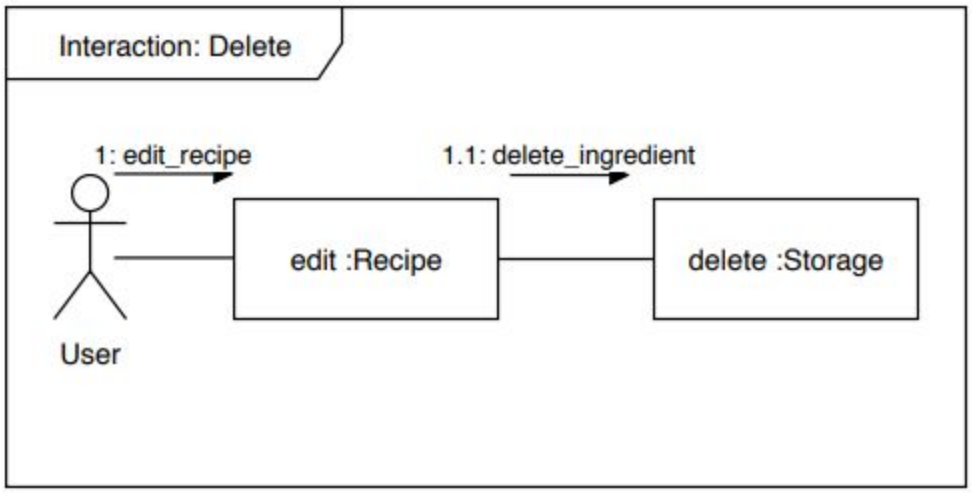


Figure 5.10.4.1.1. Interaction - Delete

5.10.4.2 Update

Interaction: Update is represented below (SRS 3.2.5.1). This diagram represents the interactions which occur when a user updates a recipe (SDD 5.6.1.6.2).

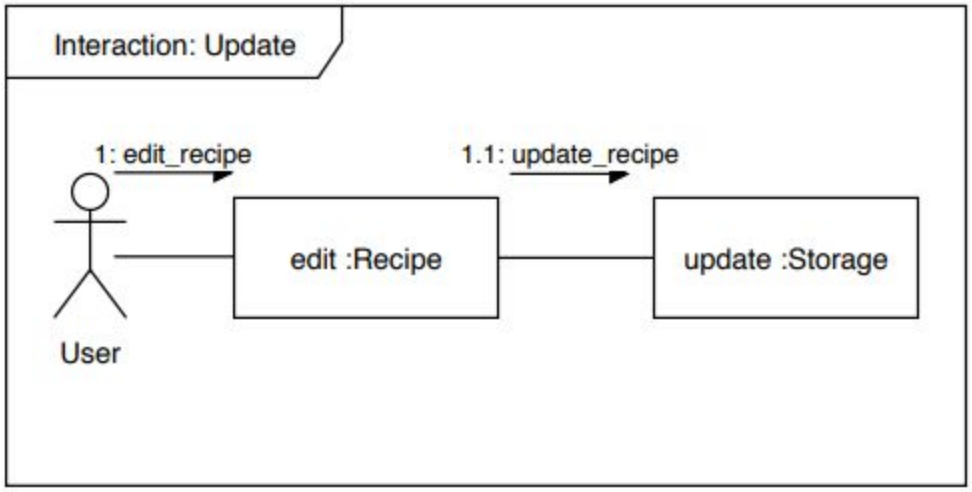


Figure 5.10.4.2.1. Interaction - Update

5.10.4.3 Notes Management

Interaction: Notes Management is represented below (SRS 3.2.11.4.1). This diagram represents the interactions which occur when a user selects an option to edit a note. In this diagram, “update_note” and “edit” (SDD 5.2.2.34) may be replaced with “create_note” and “create” (SDD 5.2.2.32), or “delete_note” and “delete” (SDD 5.2.2.35) respectively.

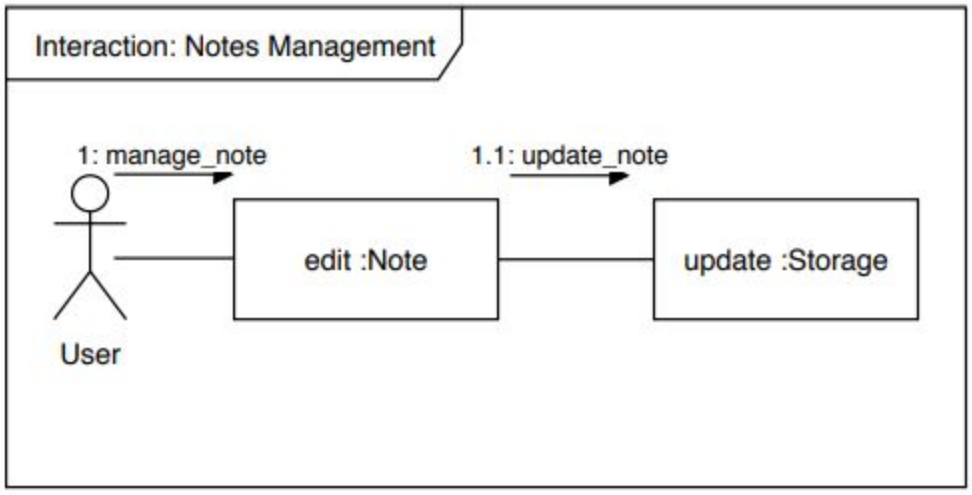


Figure 5.10.4.3.1. Interaction - Notes Management

5.10.4.4 Recipe Uploader

Interaction: Recipe Uploader is represented below (SRS 3.7.1.1, SDD 5.8.7). This diagram represents the interactions which occur when a user uploads a recipe from a third party site or an API.

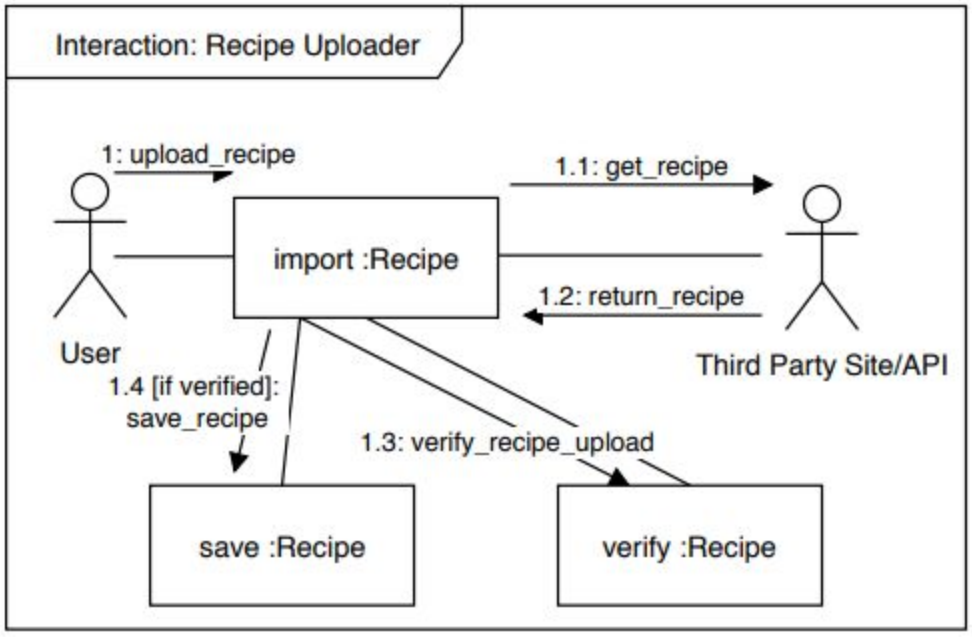


Figure 5.10.4.4.1. Interaction - Recipe Uploader

5.10.4.5 Add Meal Plan to Calendar

Interaction: Add Meal Plan to Calendar is represented below (SRS 3.2.12.3, SDD 5.5.1.8). This diagram represents the interactions which occur when a user adds to their calendar a meal plan.

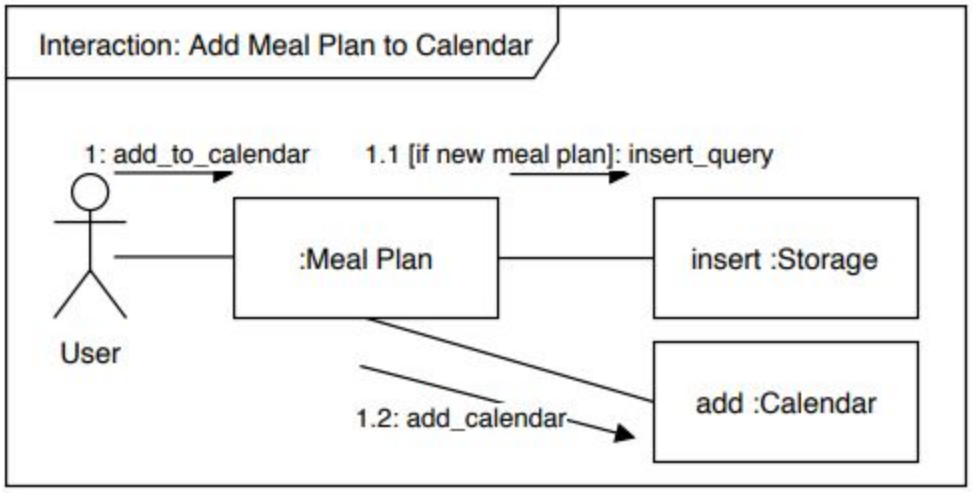


Figure 5.10.4.5.1. Interaction - Add Meal Plan to Calendar

5.10.4.6 Delete Ingredient

Interaction: Delete Ingredient is represented below (SRS 3.2.13.2). This diagram represents the interactions which occur when a user deletes an ingredient from their shopping list (SDD 5.2.2.36, SDD 5.2.2.37, SDD 5.2.2.38, SDD 5.6.1.6.4).

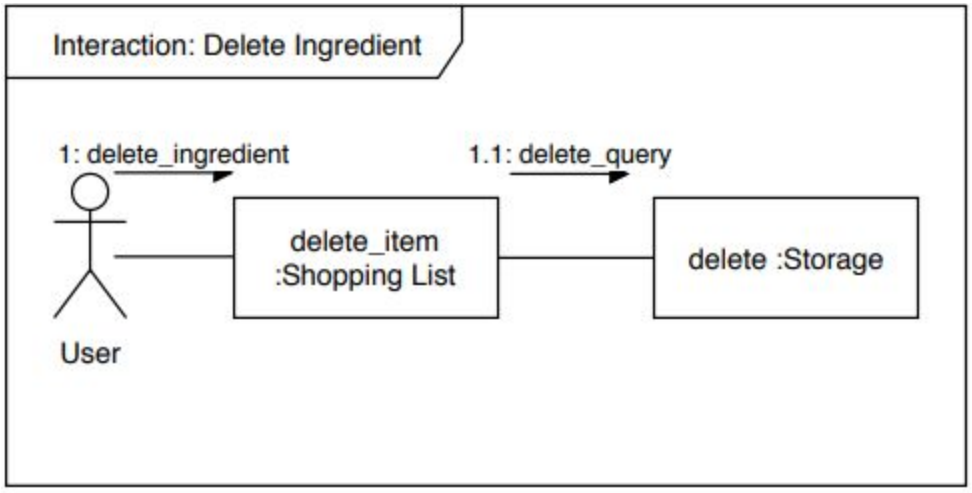


Figure 5.10.4.6.1. Interaction - Delete Ingredient

5.10.4.7 Retrieve Ingredients

Interaction: Retrieve Ingredients is represented below (SRS 3.2.13.1). This diagram represents the interactions which occur when a user retrieves a list of ingredients from a shopping list (SDD 5.2.2.36, SDD 5.6.1.6.4).

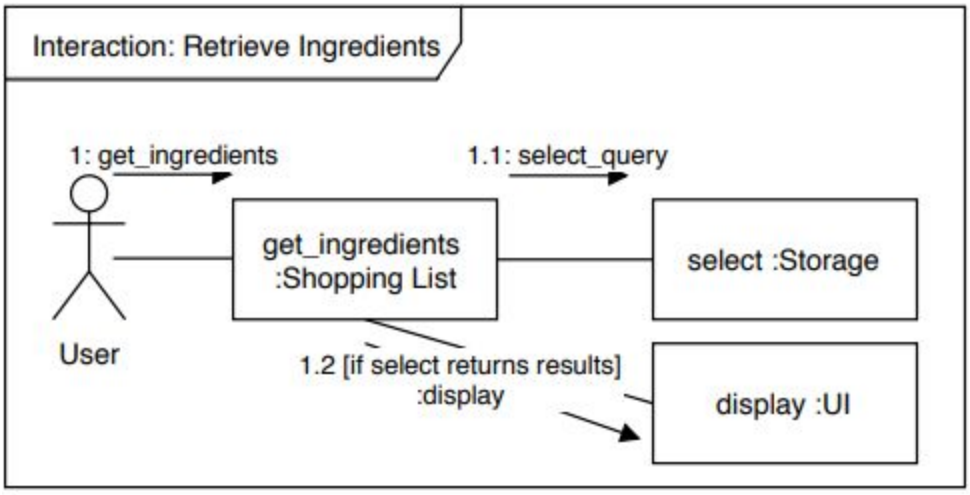


Figure 5.10.4.7.1. Interaction - retrieve Ingredients

5.10.4.8 Edit Recipe Nutritional Information

Interaction: Edit Recipe (SDD 5.11.1) Nutritional Information is represented below (SRS 3.2.14.1). This diagram represents the interactions which occur when a user edits the nutrition information of an existing recipe.

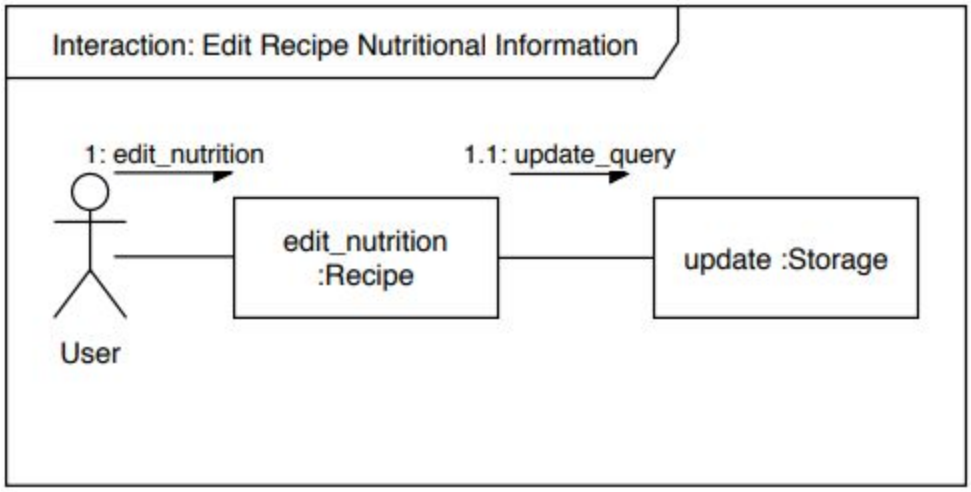


Figure 5.10.4.8.1. Interaction - Edit Recipe Nutritional Information

5.10.4.9 Get Recipe Nutritional Information

Interaction: Get Recipe Nutritional Information is represented below (SRS 3.2.14.1). This diagram represents the interactions which occur in order for the nutrition (SDD 5.4.1.9, SDD 5.4.1.10) of a recipe to become populated.

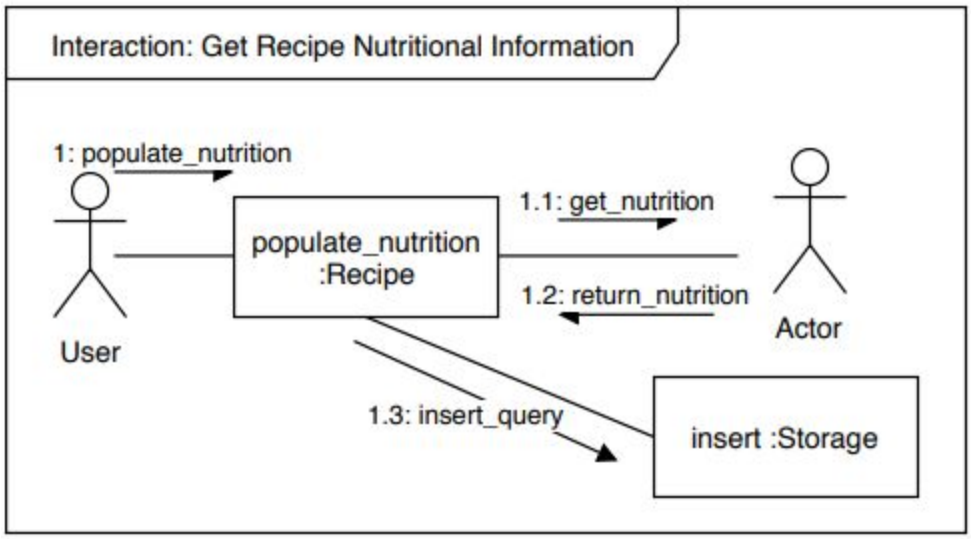


Figure 5.10.4.9.1. Interaction - Get Recipe Nutritional Information

5.10.5 User Interface (UI)

This section demonstrates interactions between the user and entities associated with the user interface (SDD 5.8). The interface may be referenced in more detail in 5.8.

5.10.5.1 Search Bar Functionality

Interaction: Search Bar Functionality is represented below (SRS 3.1.1.1.1, SRS 3.1.1.1.1.2). This diagram represents the interactions which occur when a user places input into the search bar (SDD 5.8.1.4). If get_quick_key returns true from the browser, that indicates that the user allows input to use their browser's autofill functionality. If it returns false, then that indicates standard user input.

#bookmark=id.8axmmqm3tfmk

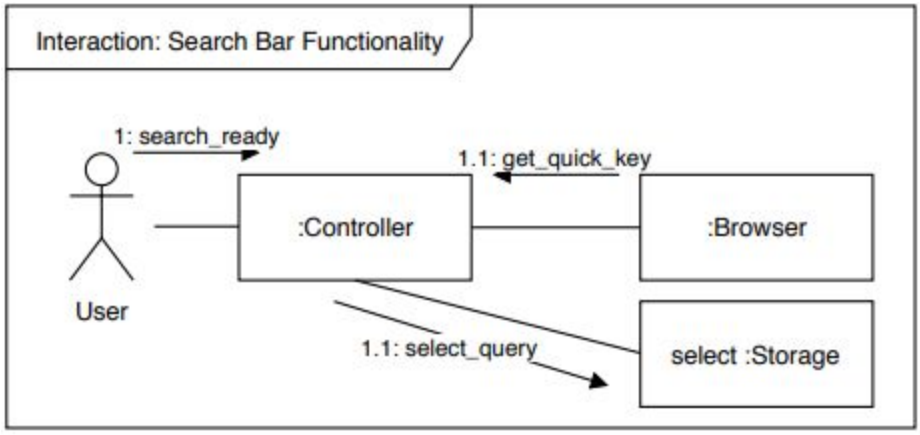


Figure 5.10.5.1.1. Interaction - Search Bar Functionality

5.10.5.2 Share Button

Interaction: Share Button is represented below (SRS 3.8.3.1, SRS 3.2.4.1). This diagram represents the interactions which occur when a user shares a recipe (SDD 5.5.1.1, SDD 5.2.2.25).

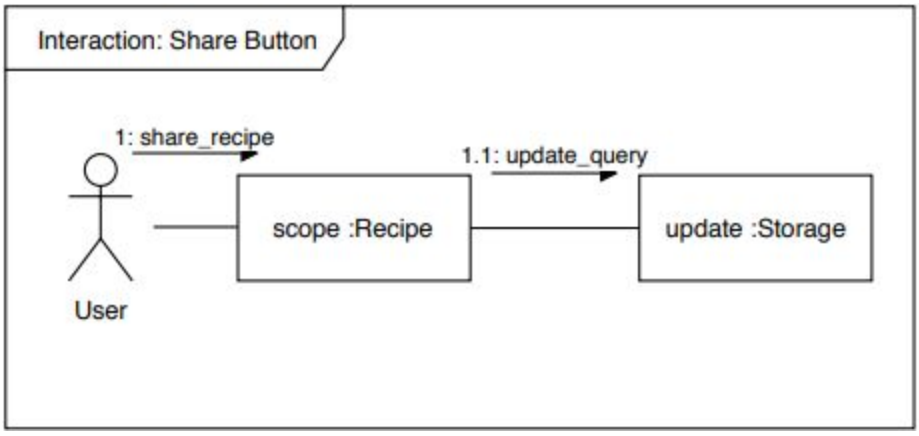


Figure 5.10.5.2.1. Interaction - Share Button

5.10.5.3 Edit Recipe Button

The interaction of the Edit Recipe (SDD 5.2.2.16) Button is represented in section 5.10.4.2 and (SRS 3.8.3.2).

5.10.6 Get Recipe by Title

This section demonstrates the interactions between a user and a search recipe function through the main interface.

Interaction: Get Recipe by Title is represented below (SRS 3.3.7, SRS 3.2.10.2.1, SDD 5.2.2.14). This diagram represents the interactions which occur when a user searches for a recipe using the search bar. Other filter options that this diagram could represent aside from title include: author (SRS 3.2.10.2.2, SDD 5.2.2.5), ingredients (SRS 3.2.10.2.3, SDD 5.2.2.6), keyword (SRS 3.2.10.2.4, SDD 5.2.2.7), prep/cook time (SRS 3.2.10.2.5), tags (SRS 3.2.10.2.6, SDD 5.2.2.9), rating (SRS 3.2.10.2.7, SDD 5.2.2.10), main ingredients (SRS 3.8.4.6, SDD 5.2.2.13), favorites (SRS 3.8.4.6), ratings (SRS 3.8.4.6), and seasonal (SRS 3.8.4.6).

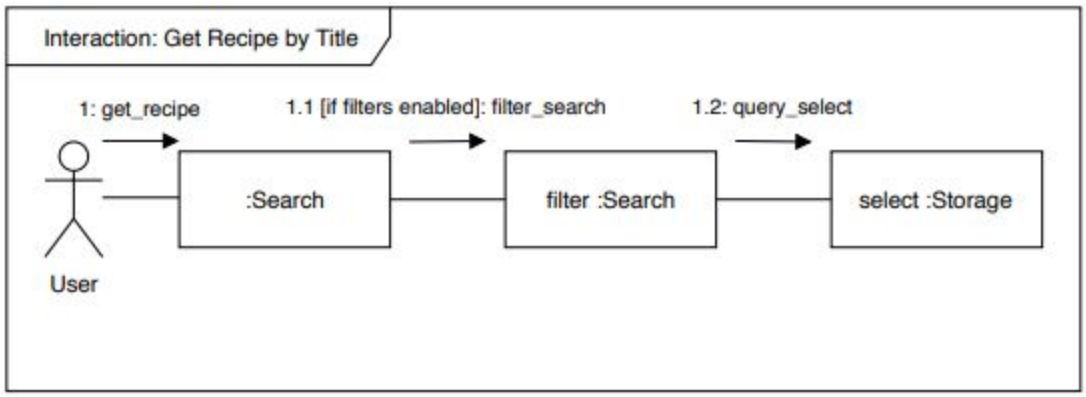


Figure 5.10.6.1. Interaction - Get Recipe by Title

5.11 State Dynamics

The state dynamic viewpoint is described in IEEE Std 1016-2009 on page 21, section 5.11. This document uses the UML activity diagram design language to illustrate the application's state dynamics.

5.11.1 Design Entities

Login Page	SRS: 3.7.2
Primary Navigation	SRS: 3.1.1
View/Edit Recipe	SRS: 3.2.5
User Application Settings	SRS: 3.1.3.2.1.2
Personal Recipe Library	SRS: 3.1.3.2.1.1
Group/Shared Recipe Library	SRS: 3.1.1.1.2
Public Recipe Library	SRS: 3.1.1.2.1.1
Menu Planner	SRS: 3.2.12
Shopping List	SRS: 3.2.13
Groups	SRS: 3.2.2

5.11.1.1 Outline Of Entities And Their Activities

This list contains the outline of possible activities within the Cookbook application and the possible transition between each activity. Starting from the initial opening of the web application where the user will login and proceeds through the list of activities that they can choose from.

1. The user opens the web app in a web browser.
2. **Login**: If the user has not logged in recently or at all, the login page will load and will be presented with the option to login or sign up.
3. If they select sign up, a new view will load and will prompt the user for necessary information. Once the information is entered and the sign up button is selected, a new account will be created for the user and the main page of the application will load (5).
4. If the user has already logged in recently, the main page of the application will load (5).
5. **Main Page**: the recipes section is the main page where the user can select from the navigation: recipes(6), shopping list(20), meal plan(21), add recipe(25), groups(30), and

search(35). From the hamburger menu, more options are available: user account(36), preferences(37), notifications(38), app rating(39), and log out(40).

6. **Recipes**: from the recipes section, the user will have two options: personal recipes(7), and public recipes(16). Upon selecting either, corresponding recipes will be loaded onto the page.
7. **Personal Recipes**: from here, the user will have three options: enable multi-select(8), select a single recipe to view(9), or search/filter recipes.
8. If the user enables multi-select, they can then select the recipes they desire then select one of the three options: delete selected, add recipes to personal meal plan, or add recipes to a group meal plan.
9. **View Personal Recipe**: from here the user has many options: edit the recipe(10), add the recipe to a meal plan(14), share/export the recipe(15), open a new tab to video link, or add an ingredient to the shopping list.
10. **Edit Recipe**: from here, the user can: delete the recipe(11), toggle the recipe private or public, edit recipe information fields, adjust the recipe portion, cancel edits(12), or save edits(13).
11. If the user deletes the recipe, they will be directed back to their personal recipes(7).
12. If the user cancels any edits, the changes will not be saved and the recipe will be re-loaded(9).
13. If the user saves any edits, the changes will be checked by the moderator, saved if they are good, and the recipe will be re-loaded(9).
14. If the user chooses to add the recipe to a meal plan, they will be prompted to choose to add it to personal meal plan or a group meal plan. Once selected, the meal plan section will be shown so that the user can select what day to add it to. The user will then be redirected back to the recipe(9).
15. If the user chooses to share or export the recipe, a prompt will appear asking to share or export. If share is selected, a link to the recipe will be generated. If export is selected, the recipe will be converted into a PDF document and prompt the user for further action.
16. **Public Recipes**: from here, the user will have three options: enable multi-select(17), select a single recipe to view(18), or search/filter recipes.
17. If the user enables multi-select, they can then select the recipes they desire then select one of the three options: add recipes to personal library, add recipes to personal meal plan, or add recipes to a group meal plan.
18. **View Public Recipe**: from here the user has many options: add the recipe to a meal plan(14), share/export the recipe(15), open a new tab to video link, or rate the recipe(19).
19. If the user chooses to rate the recipe, they will be prompted to select 1-5 stars and leave an option comment. When submitting a comment, the text will be checked in the moderator and saved if it passes.
20. **Shopping List**: from here, the user can do the following: add a shopping item, edit a shopping item, or delete a shopping item.

21. **Menu Planner**: from here, the user can do the following: generate a shopping list from the planned menu(22), export the plan to the calendar(23), or edit the menu plan(24).
22. If the users chooses to generate a shopping list from the planned menu, a prompt to confirm will appear. If the user confirms, all of the ingredients will be aggregated and added to the shopping list.
23. If the user chooses to export the plan to their calendar, a prompt to confirm the correct Google account will appear. After confirming, the data will be prepared and sent through the Google calendar API to the user's calendar.
24. If the user chooses to edit the menu plan, they will be given the option to add or remove a recipe item. If the user is the admin of the menu plan, they will also be given the option to add or remove people who can access the menu plan.
25. **Add Recipe**: from here, the user will be prompted with two options: import a recipe(26) or create a new recipe(27).
26. **Import Online Recipe**: from here, the user will be prompted to input a url to a webpage that has a recipe. After the user confirms, the recipe import API will grab information and populate a new recipe with the information. Then the user can choose to edit the information(27).
27. **Create New Recipe**: from here, the user can: cancel(28), toggle the recipe private or public, edit recipe information fields, adjust the recipe portion, or save the recipe(29).
28. If the user cancels the recipe, they will be directed back to the prompt(25).
29. If the user saves the recipe, the information will be checked by the moderator, saved if they are good, and the saved recipe will be loaded(9).
30. **Groups**: from here, the user will be presented with all of the groups they are a part of. After selecting a group, the user can: view the group's recipes(33) or show group members(34).
31. **Group Recipes**: from here, the user will have three options: enable multi-select(32), select a single recipe to view(33), or search/filter recipes.
32. If the user enables multi-select, they can then select the recipes they desire then select one of the three options: remove selected, add recipes to personal meal plan, or add recipes to a group meal plan.
33. **View Group Recipe**: from here the user has many options: add the recipe to a meal plan(14), share/export the recipe(15), open a new tab to video link, or add an ingredient to the shopping list.
34. **View Group Members**: from here the user can: see all group members, and if they are an admin, add or remove group members and update user permissions.
35. **Search**: from search, the user can filter recipes by personal, group, and public recipes, as well as title, tags, ratings, author, and ingredients.
36. **User Account**: from here, the user can change their username, password, or profile image.
37. **Preferences**: from here, the user can choose a color theme for the app.
38. **Notifications**: from here, the user can see notifications on latest reviews on their public apps, people following them, and being added to groups.

39. **App Rating**: from here, the user can leave feedback on the app.
40. **Logout**: if the user chooses to logout, they will be redirected to the login page(2).

5.11.2 State Dynamics Views

In order to make the activity diagrams simple to understand, the app has been broken into ten different diagrams. Each diagram represents a unique part of the application.

5.11.2.1 Login Page

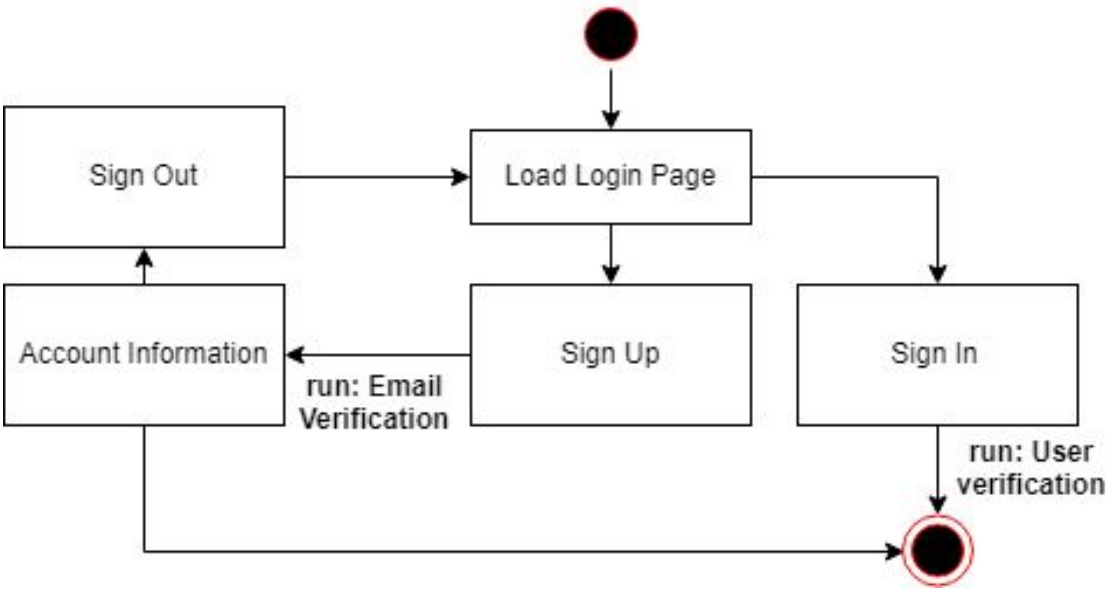


Figure 1. Login Page - The Login Page show the general functionality of how the user can either sign up to start using the application or login and be sent to the Main Page. (see 5.8.2.2 and 5.2.2.43-45)

5.11.2.1.1 Design Concerns

SRS	3.2.1.1, 3.7.2.1
SDD	5.8.2.2, 5.2.2.43-45

5.11.2.2 Primary Navigation

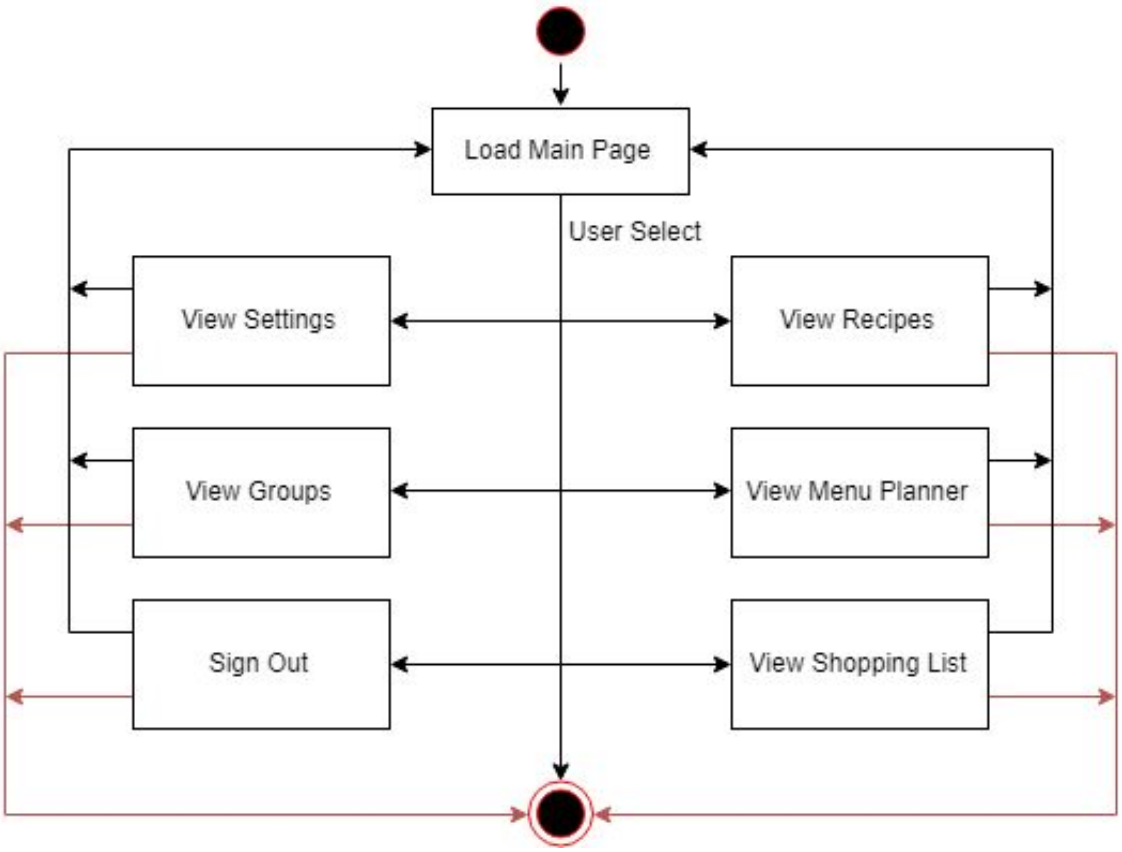


Figure 2. Primary Navigation - the primary navigation is how the user will navigate between the primary sections of the application by means of a navigation bar (see 5.8.2.1.1)

5.11.2.2.1 Design Concerns

SRS	3.1.1, 3.2.1.1
SDD	5.8.2.1.1

5.11.2.3 View/Edit Recipe

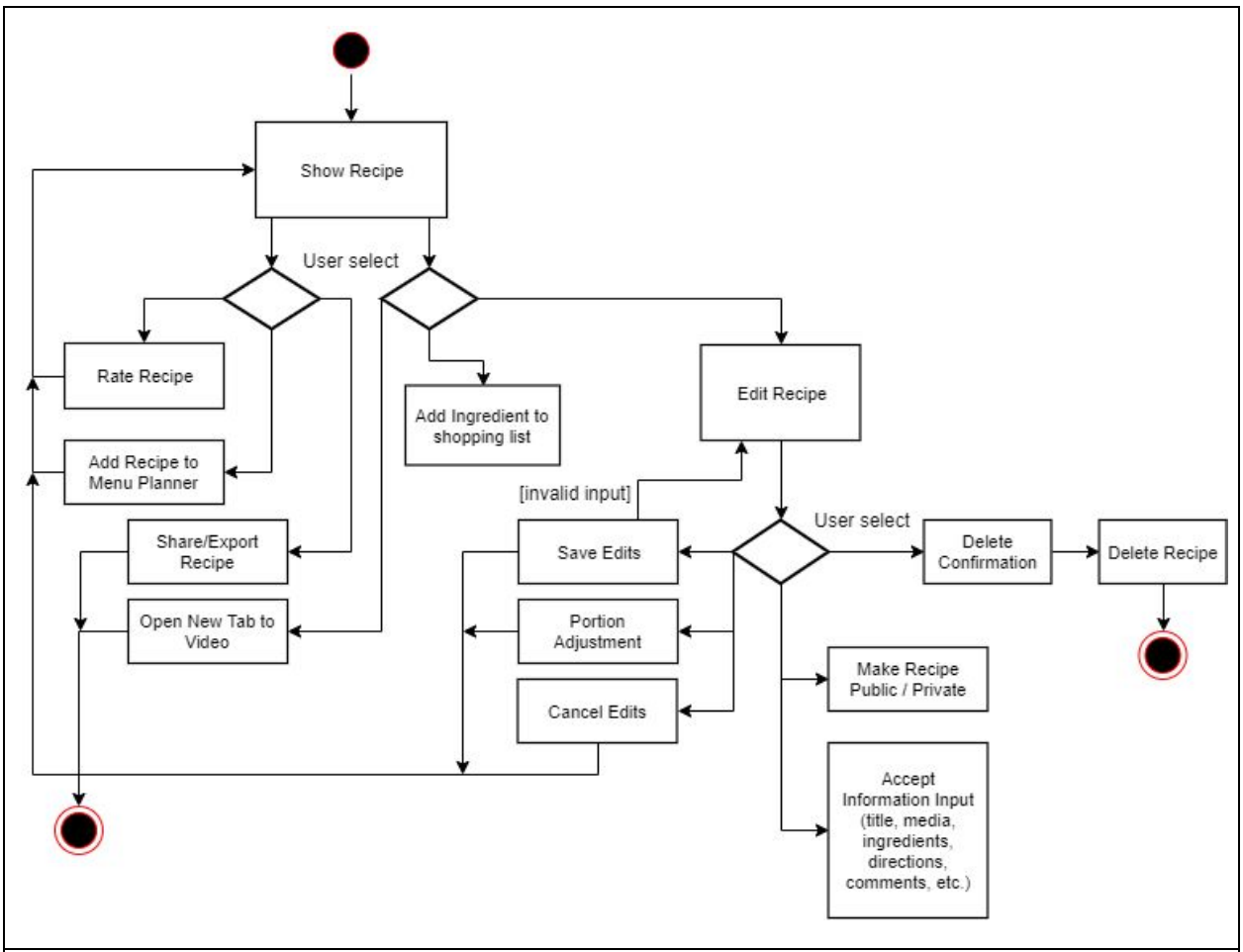


Figure 3. View/Edit Recipe - this diagram shows what functions a user can do when viewing and editing a thier recipe. (see 3.2.3, 3.2.4 and 3.2.5)

5.11.2.3.1 Design Concerns

SRS	3.2.5
SDD	3.2.3 , 3.2.4 , 3.2.5

5.11.2.4 User Application Settings

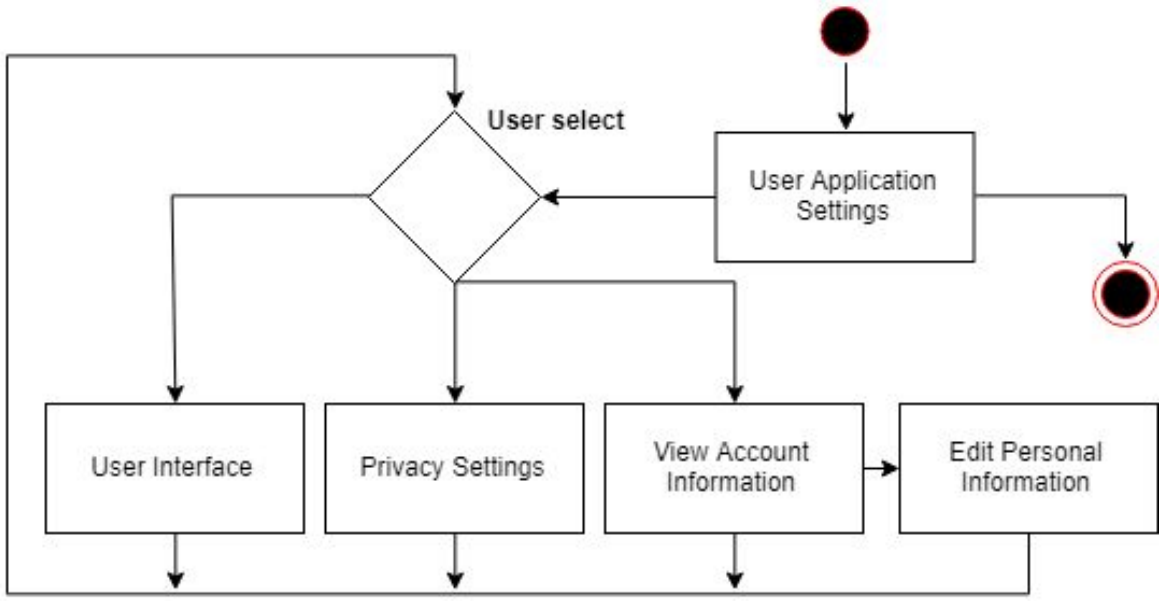


Figure 4. User Application Settings - This diagram shows what settings and personal information a user can access or change as well as the pathway they will be able to move throughout the different settings pages. (see 5.6.1.3)

5.11.2.4.1 Design Concerns

SRS	3.1.3.2.1.2
SDD	5.6.1.3

5.11.2.5 Personal Recipe Library

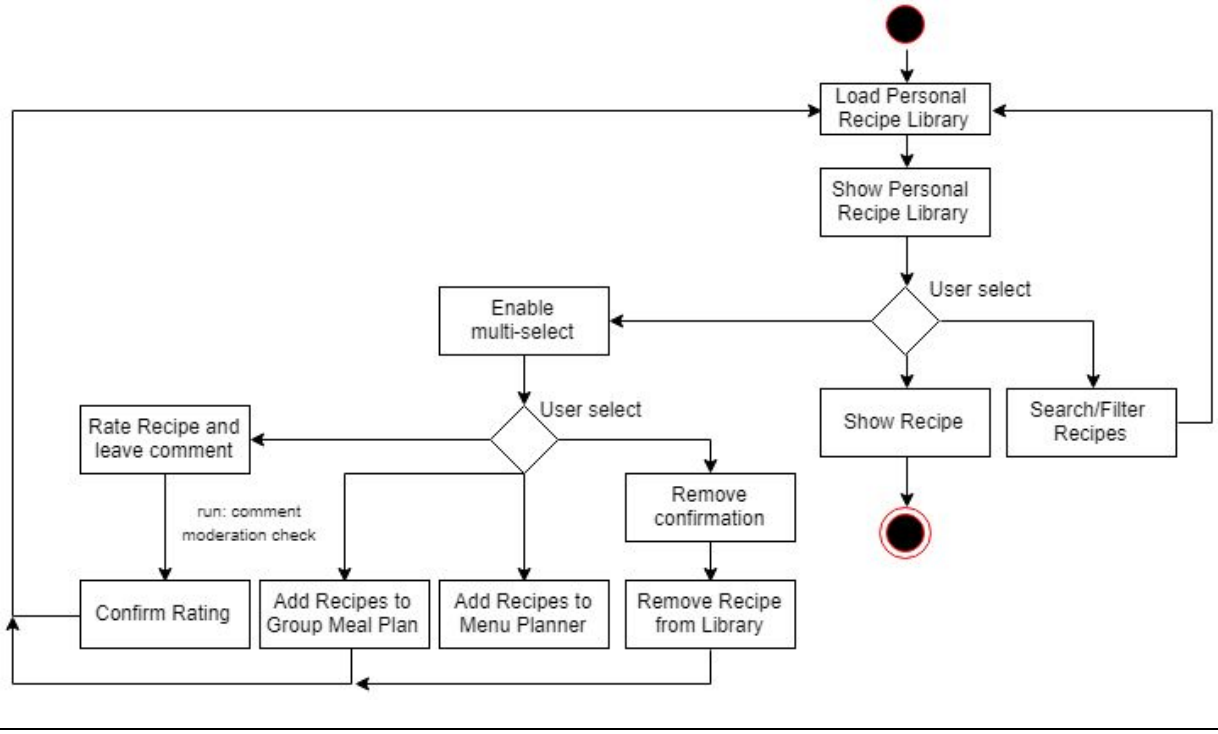


Figure 5. Personal Recipe Library - The above diagram is a visual representation of a users saved personal recipes. This also shows the flow of viewing their library to making edits and then leaving back to the main page or another such as Menu Planner, Group / Shared Recipe Library, etc. (see 5.2.2)

5.11.2.5.1 Design Concerns

SRS	3.1.3.2.1.1
SDD	5.2.2

5.11.2.6 Group/ Shared Recipe Library

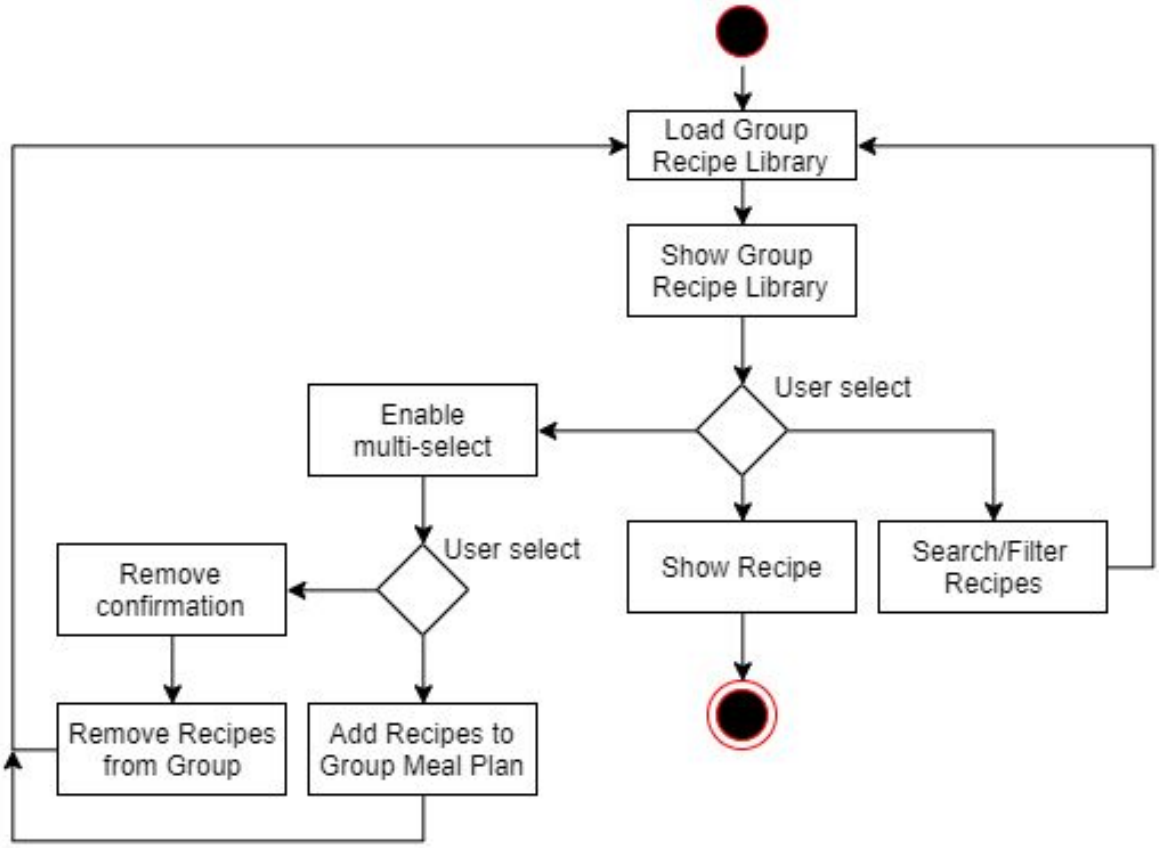


Figure 6. Group/Shared Recipe Library - Above diagram gives contains a representation of a users group or shared recipe library. It functions very similar to their personal recipe library except it will affect more than a single user. (see 5.2.2 and 5.6.1.4)

5.11.2.6.1 Design Concerns

SRS	3.1.1.1.2
SDD	5.2.2, 5.6.1.4

5.11.2.7 Public Recipe Library

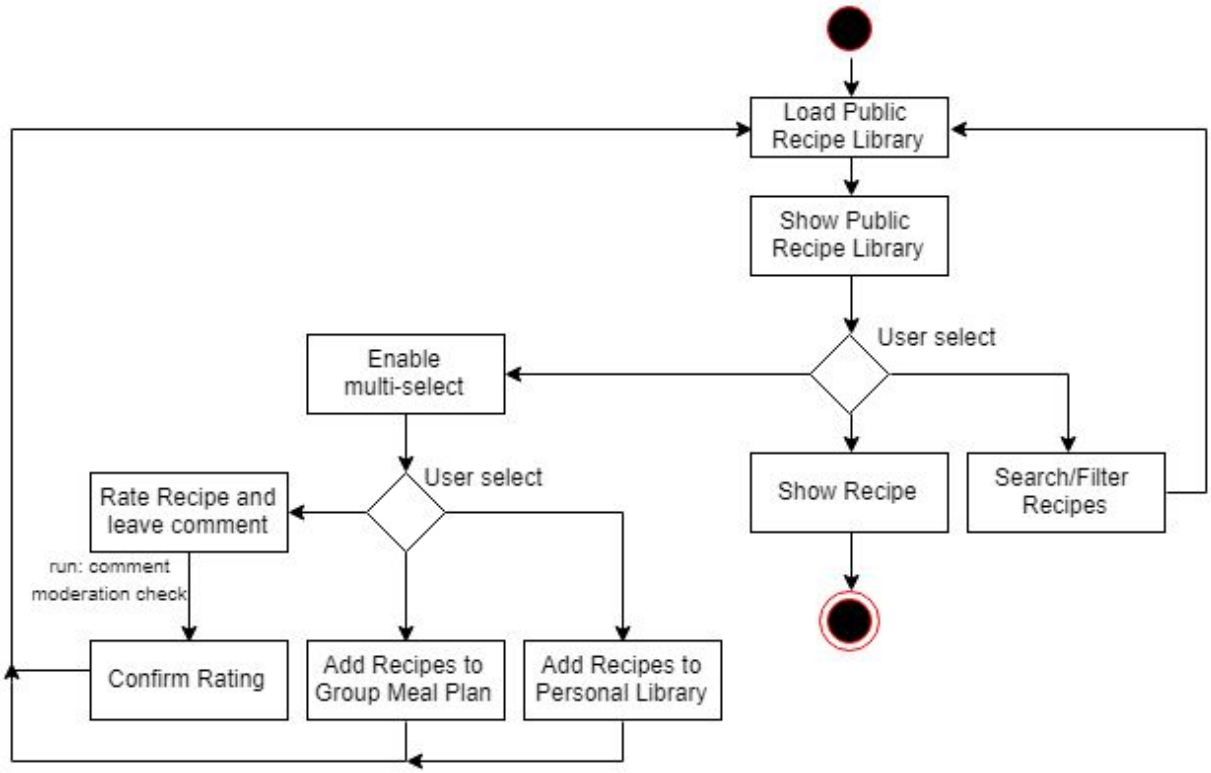


Figure 7. Public Recipe Library - The above diagram displays the options a user can choose from and the pathway a user can take when viewing public recipies stored on the database. (see 5.2.2 and 5.6.1.5)

5.11.2.7.1 Design Concerns

SRS	3.1.1.2.1.1
SDD	5.2.2, 5.6.1.5

5.11.2.8 Menu Planner

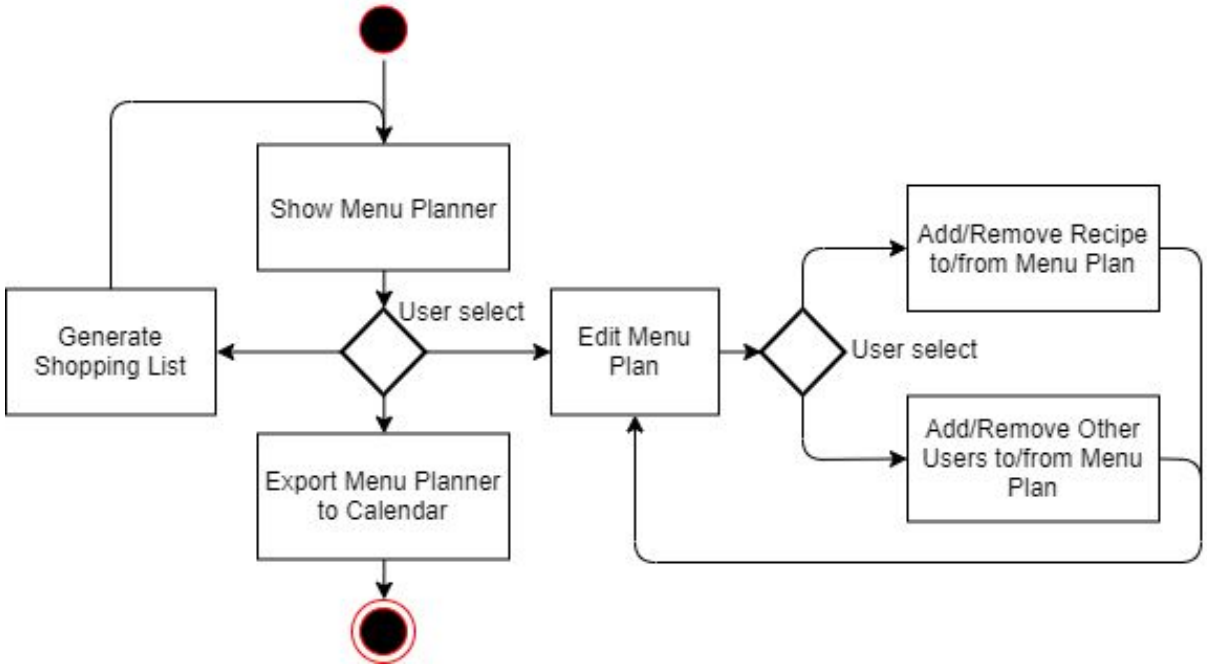


Figure 8. Menu Planner - In this diagram a user can generate a shopping list, export their menu planner to a calendar, edit the menu planner by adding or removing recipes or users as desired. (see 5.6.1.6)

5.11.2.8.1 Design Concerns

SRS	3.2.12, 3.5.4.3
SDD	5.6.1.6

5.11.2.9 Shopping List

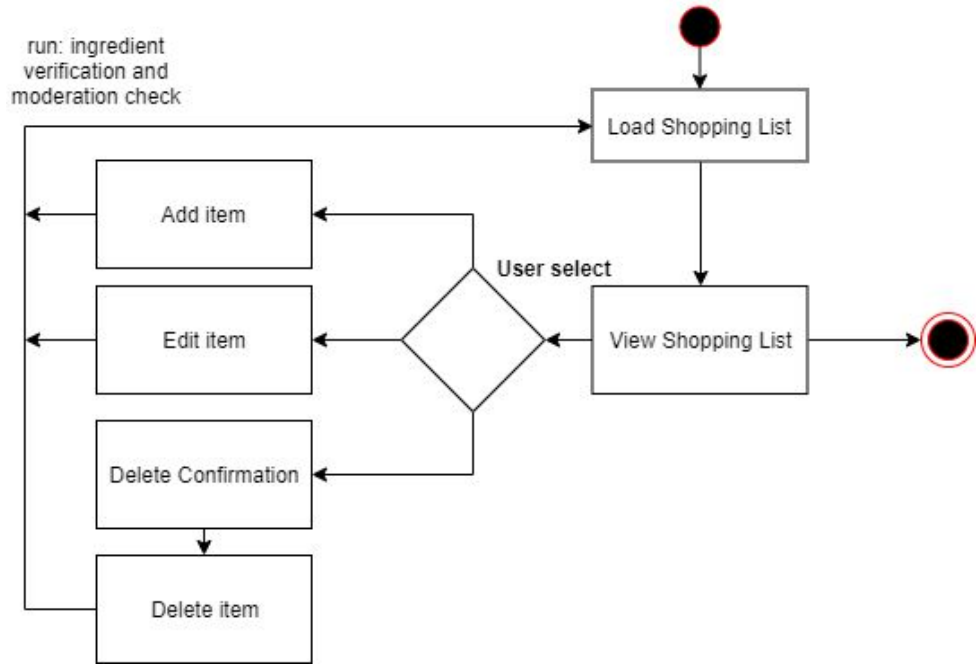


Figure 9. Shopping List - This diagram represents the users shopping list. Within their shopping list they are able to add, remove, and edit these items. (see 5.8.11)

5.11.2.9.1 Design Concerns

SRS	3.2.13, 3.5.4.2
SDD	5.2.2.36 -38, 5.6.1.6.4, 5.8.11

5.11.2.10 Groups

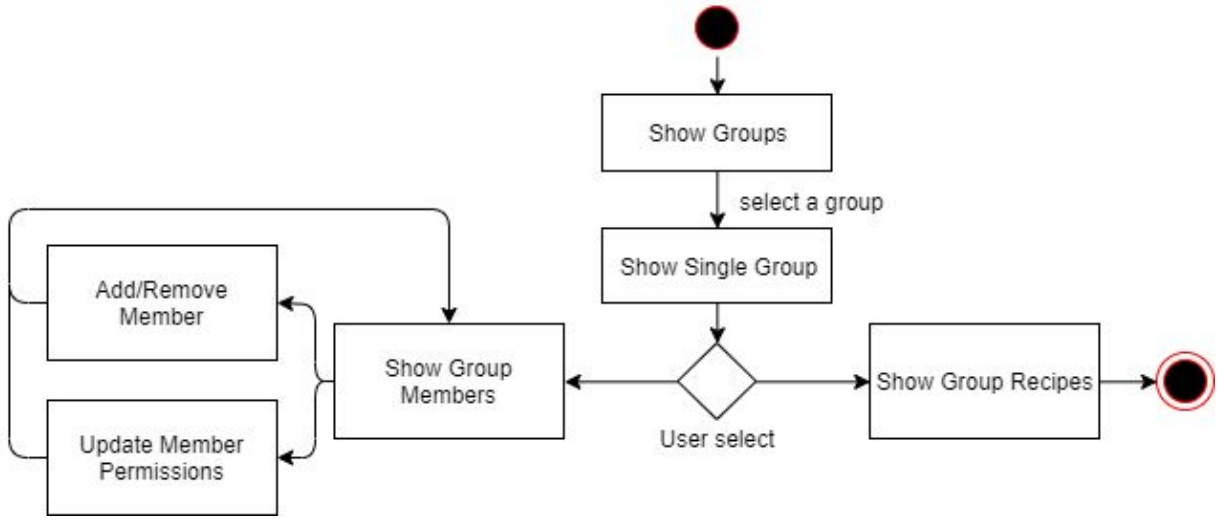


Figure 10. Groups - This diagram pertains to a users group and how they can interact within the groups settings. Here they are able to view a list of members and if they have required permissions can make changes to their group and recipes. (see 5.6.1.4 and 5.8.6)

5.11.2.10.1 Design Concerns

SRS	3.2.2, 3.5.4.4
SDD	5.6.1.4, 5.8.6

5.12 Algorithm

The algorithm viewpoint is described in IEEE Std 1016-2009 on page 21, section 5.12. The purpose of the algorithm viewpoint is to provide a detailed design description of operations and the internal details and logic of each design entity. This will include pseudo SQL statements that serve as pseudo-code into how the database is queried.

5.12.1 Structured Query Language

While the SQL language would allow many more queries to be written against the database schema, this is a reasonably complete catalog of queries. These SQL statements have not been implemented in MariaDB using its exact syntax, thus serve as pseud-code for the structured query language.

5.12.1.1 Data Definition Language

This is the set of queries used to define the database schema. These are the minimum queries believed to be necessary for the application, though further constraints might need to be imposed on data definitions, or additional indexes created, to optimize performance and ensure data integrity.

5.12.1.1.1 Create Users Table

```
CREATE TABLE users(
  id           INTEGER PRIMARY KEY,
  firebaseId   TEXT UNIQUE NOT NULL, -- firebase must provide some sort of unique id;
                                     -- cache it so that we don't have to parse json
                                     -- for every query to lookup a user
  authJson     TEXT UNIQUE NOT NULL, -- no two people should have identical auth data
  email        TEXT, -- cached version of email from authJson if available
  fullName     TEXT, -- cached version of full name from authJson if available
  nickName     TEXT, -- cached version of nick name from authJson if available
  imageUrl     TEXT  -- cached version of image url from authJson if available
);
```

(SDD 5.6.1)

5.12.1.1.2 Create Groups Table

```
CREATE TABLE groups(
  id           INTEGER PRIMARY KEY,
  ownerId      INTEGER REFERENCES users(id),
  name         TEXT,
  UNIQUE (ownerId, name) -- each group created by an owner must have a unique
                         -- name, but the name can be reused by other owners
);
```

(SDD 5.6.1)

5.12.1.1.3 Create Groups to Users Join Table

```
CREATE TABLE groups2users(
  groupId      INTEGER REFERENCES groups(id),
  userId       INTEGER REFERENCES users(id),
  permission   INTEGER, -- could be enumeration or bits
  UNIQUE (groupId, userId) -- user should only appear in a group once
);
```

(SDD 5.6.1)

5.12.1.1.4 Create Ingredients Table

```
CREATE TABLE ingredients(
  id           INTEGER PRIMARY KEY,
  name         TEXT UNIQUE,
  nutrientJson TEXT
);
```

(SDD 5.6.1)

5.12.1.1.5 Create Quantities Table

```
CREATE TABLE quantities(
  id           INTEGER PRIMARY KEY,
  qtyText      TEXT,   -- some are not numeric: "a" (aka 1), half, 1/2
  qtyNumeric   NUMERIC -- numeric value if possible for scaling purposes
);
```

(SDD 5.6.1)

5.12.1.1.6 Create Units Table

```
CREATE TABLE units(
  id           INTEGER PRIMARY KEY,
  unitText     TEXT -- could be anything: imperial, metric, sm/med/lg, pinch
);
```

(SDD 5.6.1)

5.12.1.1.7 Create Ingredient Tuples Join Table

```
CREATE TABLE ingTuples(
  id           INTEGER PRIMARY KEY,
  qtyId        INTEGER REFERENCES quantities(id),
  unitId       INTEGER REFERENCES units(id),
  ingId        INTEGER REFERENCES ingredient(id),
  UNIQUE (qtyId, unitId, ingId)
);
```

(SDD 5.6.1)

5.12.1.1.8 Create Ingredient Lists Table

```
CREATE TABLE ingLists(
  id           INTEGER PRIMARY KEY
);
```

(SDD 5.6.1)

5.12.1.1.9 Create Ingredient Lists to Ingredient Tuples Join Table

```
CREATE TABLE ingLists2ingTuples(
  ingListId    INTEGER REFERENCES ingLists(id),
  ingTupleId   INTEGER REFERENCES ingTuples(id),
  seq          INTEGER,
  UNIQUE (ingListId, ingTupleId), -- each tuple should only appear once in list
  UNIQUE (ingListId, seq) -- each sequence should only appear once in a list
);
```

(SDD 5.6.1)

5.12.1.1.10 Create Recipes Table

```
CREATE TABLE recipes(
  id           INTEGER PRIMARY KEY,
  ownerId      INTEGER REFERENCES users(id),
  title        TEXT,
  author       TEXT, -- could come from someone not registered in our system
  prepTime     INTEGER, -- in seconds if anything needs that level of precision
  equipment    TEXT,
  instructions TEXT,
  servingsLow  INTEGER,
  servingsHigh INTEGER,
  ingListId    INTEGER REFERENCES ingLists(id),
  privacy      INTEGER,
  mainIngId    INTEGER REFERENCES ingredients(id) -- NULL if no main ing
);
```

(SDD 5.6.1)

5.12.1.1.11 Create Recipe Media Table

```
CREATE TABLE recipeMedia(
  recipeId     INTEGER REFERENCES recipes(id),
  mediaType    TEXT,
  url          TEXT,
  seq          INTEGER,
  UNIQUE (recipeId, url),
  UNIQUE (recipeId, seq)
);
```

(SDD 5.6.1)

5.12.1.1.12 Create Recipe Lists Table

```
CREATE TABLE recipeLists(
  id           INTEGER PRIMARY KEY,
  ownerId      INTEGER REFERENCES users(id),
  name         TEXT,
  UNIQUE (ownerId, name) -- each recipe list created by an owner must have a
                         -- unique name, but the name can be reused by other
                         -- owners
);
```

(SDD 5.6.1)

5.12.1.1.13 Create Recipe Lists to Recipes Join Table

```
CREATE TABLE recipeLists2recipes(
  recipeListId INTEGER REFERENCES recipeLists(id),
  recipeId     INTEGER REFERENCES recipes(id),
  UNIQUE (recipeListId, recipeId) -- recipe should only appear in a list once
);
```

(SDD 5.6.1)

5.12.1.1.14 Create Users to Recipe Lists Join Table

```
CREATE TABLE users2recipeLists(
  userId       INTEGER REFERENCES users(id),
  recipeListId INTEGER REFERENCES recipeLists(id),
  UNIQUE (userId, recipeListId)
);
```

(SDD 5.6.1)

5.12.1.1.15 Create Groups to Recipe Lists Join Table

```
CREATE TABLE groups2recipeLists(
  groupId      INTEGER REFERENCES groups(id),
  recipeListId INTEGER REFERENCES recipeLists(id),
  UNIQUE (groupId, recipeListId)
);
```

(SDD 5.6.1)

5.12.1.1.16 Create Ratings Table

```
CREATE TABLE ratings(
  recipeId INTEGER REFERENCES recipes(id),
  authorId INTEGER REFERENCES users(id),
  rating   INTEGER,
  comment  TEXT,
  UNIQUE (recipeId, authorId)
);
```

(SDD 5.6.1)

5.12.1.1.17 Create Meal Plans Table

```
CREATE TABLE mealPlans(
  id           INTEGER PRIMARY KEY,
  ownerId      INTEGER REFERENCES users(id),
  groupId      INTEGER REFERENCES groups(id),      -- NULL if not for group
  whenIso      TEXT,                               -- 'YYYY-MM-DDTHH:MM:SSZ'
  description  TEXT,
  recipeListId INTEGER REFERENCES recipeLists(id), -- only one per meal plan
  UNIQUE (ownerId, whenIso)
);
```

(SDD 5.6.1)

5.12.1.1.18 Create Shopping List Items Table

```
CREATE TABLE mealPlans(
  id           INTEGER PRIMARY KEY,
  ownerId      INTEGER REFERENCES users(id),
  groupId      INTEGER REFERENCES groups(id),      -- NULL if not for group
  whenIso      TEXT,                               -- 'YYYY-MM-DDTHH:MM:SSZ'
  description  TEXT,
  recipeListId INTEGER REFERENCES recipeLists(id), -- only one per meal plan
  UNIQUE (ownerId, whenIso)
);
```

(SDD 5.6.1)

5.12.1.2 Data Query Language

Select statements are defined here for the useful queries of individual and joined tables. Although these queries will work for almost all data access needs, they can be modified as needed in the implementation to avoid returning extraneous information or to simplify the query for useful data access scenarios. These optimization scenarios are left to the discretion of the implementation team.

5.12.1.2.1 Select Users

```
SELECT id, firebaseId, authJson, email, fullName, nickName, imageUrl
  FROM users
 WHERE id = :id OR firebaseId = :firebaseId;
```

(SDD 5.6.1)

5.12.1.2.2 Select Groups

```
SELECT id, ownerId, name
  FROM groups
 WHERE id = :id OR ownerId = :ownerI);
```

(SDD 5.6.1)

5.12.1.2.3 Select Group Members

```
SELECT userId, permission, firebaseId, authJson, email, fullName, nickName, imageUrl
  FROM groups2users INNER JOIN
       users ON groups2users.userId = users.id
 WHERE groupId = :groupId;
```

(SDD 5.6.1)

5.12.1.2.4 Select Ingredients

```
SELECT id, name, nutrientJson
  FROM ingredients
 WHERE id = :id OR name LIKE :pattern;
```

(SDD 5.6.1)

5.12.1.2.5 Select Quantities

```
SELECT id, qtyText, qtyNumeric
  FROM quantities
 WHERE id = :id OR qtyText LIKE :pattern OR qtyNumeric = :qtyNumeric;
```

(SDD 5.6.1)

5.12.1.2.6 Select Units

```
SELECT id, unitText
  FROM units
 WHERE id = :id OR unitText LIKE :pattern;
```

(SDD 5.6.1)

5.12.1.2.7 Select Ingredient Tuples

```
SELECT t.id,
       q.id, q.qtyText, q.qtyNumeric,
       u.id, u.unitText,
       i.id, i.name, i.nutrientJson
  FROM ingTuples AS t INNER JOIN
       quantities AS q ON q.id = t.qtyId INNER JOIN
       units AS u ON u.id = t.unitId INNER JOIN
       ingredients AS i ON i.id = t.ingId
 WHERE t.id = :id OR (q.id = :qtyId AND u.id = :unitId AND i.id = :ingId);
```

(SDD 5.6.1)

5.12.1.2.8 Select Ingredient List

```
   SELECT l.id, l.seq, l.ingTupleId,
          q.id, q.qtyText, q.qtyNumeric,
          u.id, u.unitText,
          i.id, i.name, i.nutrientJson
     FROM ingList AS l INNER JOIN
          ingTuples AS t ON t.id = l.ingTupleId INNER JOIN
          quantities AS q ON q.id = t.qtyId INNER JOIN
          units AS u ON u.id = t.unitId INNER JOIN
          ingredients AS i ON i.id = t.ingId
    WHERE l.id = :id
 ORDER BY l.seq;
```

(SDD 5.6.1)

5.12.1.2.9 Select Recipes

```
SELECT r.id, ownerId, fullName, title, author, prepTime,
       equipment, instructions, servingsLow, servingsHigh,
       ingListId, privacy, mainIngId, i.name,
       AVG(SELECT rating FROM ratings WHERE recipeId = r.id)
  FROM recipes AS r INNER JOIN
       users AS u ON r.ownerId = u.id INNER JOIN
       ingredients AS i ON r.mainIngId = i.id
 WHERE r.id = :id OR
       ownerId = :ownerId OR
       fullName = :fullName OR
       title = :title OR
       author = :author OR
       (:prepLow <= prepTime AND prepTime <= :prepHigh) OR
       equipment LIKE :equipPattern OR
       instructions LIKE :instPattern OR
       (servingsLow <= :servings AND :servings <= servingsHigh) OR
       i.name LIKE :ingPattern;
```

(SDD 5.6.1)

5.12.1.2.10 Select Recipe Media

```
SELECT seq, url
  FROM recipeMedia
 WHERE recipeId = :recipeId
ORDER BY seq;
```

(SDD 5.6.1)

5.12.1.2.11 Select Recipe Lists

```
SELECT id, ownerId, name
  FROM recipeLists
 WHERE id = :id OR ownerId = :ownerId OR name LIKE :pattern;
```

(SDD 5.6.1)

5.12.1.2.12 Select Recipes in Recipe List

```
SELECT r.id, ownerId, fullName, title, author, prepTime,
       equipment, instructions, servingsLow, servingsHigh,
       ingListId, privacy, mainIngId, i.name,
       AVG(SELECT rating FROM ratings WHERE recipeId = r.id)
  FROM recipes AS r INNER JOIN
       users AS u ON r.ownerId = u.id INNER JOIN
       ingredients AS i ON r.mainIngId = i.id INNER JOIN
       recipeLists2recipes AS j ON j.recipeId = r.id
 WHERE j.recipeListId = :recipeListId;
```

(SDD 5.6.1)

5.12.1.2.13 Select User's Recipe Lists

```
SELECT l.id, l.ownerId, l.name
  FROM recipeLists AS l INNER JOIN
       users2recipeLists AS j ON j.recipeListId = l.id
 WHERE userId = :userId;
```

(SDD 5.6.1)

5.12.1.2.14 Select Group's Recipe Lists

```
SELECT l.id, l.ownerId, l.name
  FROM recipeLists AS l INNER JOIN
       groups2recipeLists AS j ON j.recipeListId = l.id
 WHERE groupId = :groupId;
```

(SDD 5.6.1)

5.12.1.2.15 Select Recipe(s) Ratings

```
SELECT recipeId, authorId, rating, comment
  FROM recipeRatings
 WHERE recipeId = :recipeId;
```

(SDD 5.6.1)

5.12.1.2.16 Select Meal Plans

```
SELECT id, ownerId, groupId, whenIso, description, recipeListId
  FROM mealPlans
 WHERE (ownerId = :ownerId OR groupId = :groupId) AND
       (:whenLow IS NULL OR :whenLow <= whenIso) AND
       (:whenHigh IS NULL OR whenIso <= :whenHigh);
```

(SDD 5.6.1)

5.12.1.2.17 Select Meal Plan Shopping List

```
SELECT li.mealPlanId, li.ingTupleId, li.acquired,
       q.id, q.qtyText, q.qtyNumeric,
       u.id, u.unitText,
       i.id, i.name, i.nutrientJson
  FROM shoppingListItems AS li INNER JOIN
       mealPlans AS mp ON mp.id = li.mealPlanId INNER JOIN
       ingTuples AS t ON t.id = li.ingTupleId INNER JOIN
       quantities AS q ON q.id = t.qtyId INNER JOIN
       units AS u ON u.id = t.unitId INNER JOIN
       ingredients AS i ON i.id = t.ingId
 WHERE li.mealPlanId = :mealPlanId;
```

(SDD 5.6.1)

5.13 Resources

The resources viewpoint is described in IEEE Std 1016-2009 on page 22, section 5.13. The purpose of the Resource viewpoint is to model the characteristics and utilization of resources in a design subject.

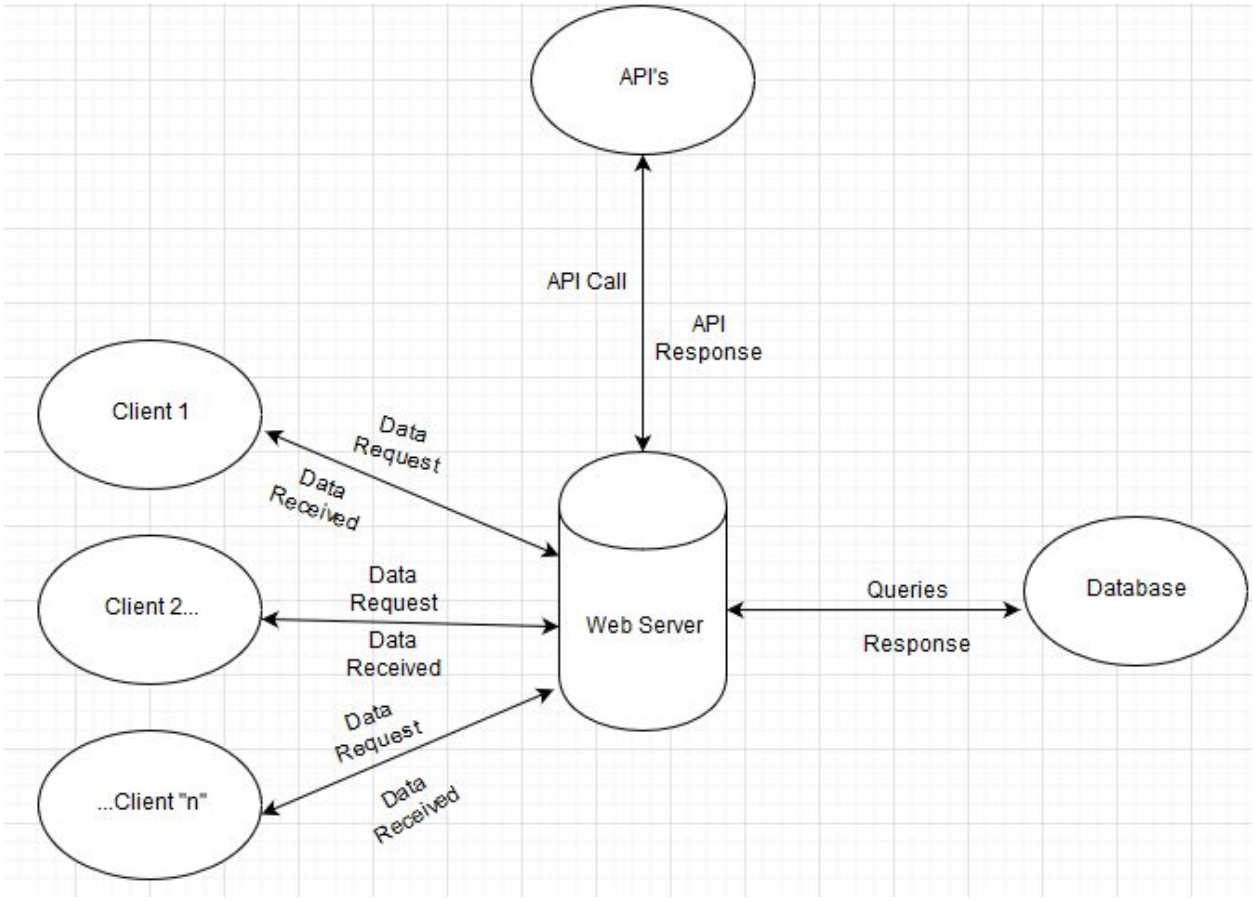


Figure 5.13.1 Flow of Resources **-** *See below for description*

This chart is a basic representation of the flow of the resources that will need to be accessed in order for the app to run properly.

5.13.1 Time Constraints Diagrams

5.13.1.1 Saving Time Diagram

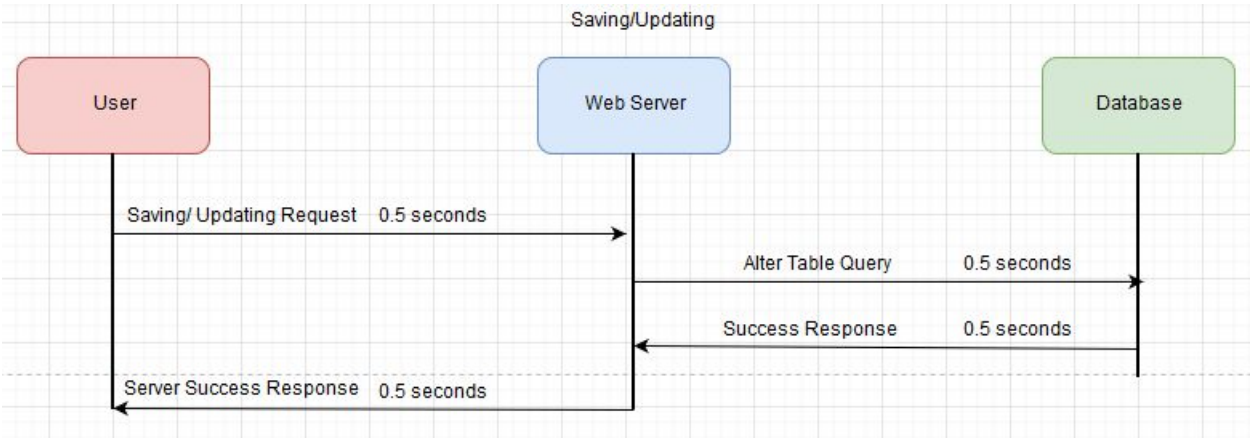


Figure 5.13.2 See below for description

The Saving Time UML Diagram represents the 2 second time requirement stated in section "3.4.6" of the SRS. This diagram breaks down the time requirement into the different stages of the action from start to finish of a successful attempt. See SDD sections 5.10.2.1, 5.10.2.3, & [Algorithms]

5.13.1.2 Searching Time Diagram

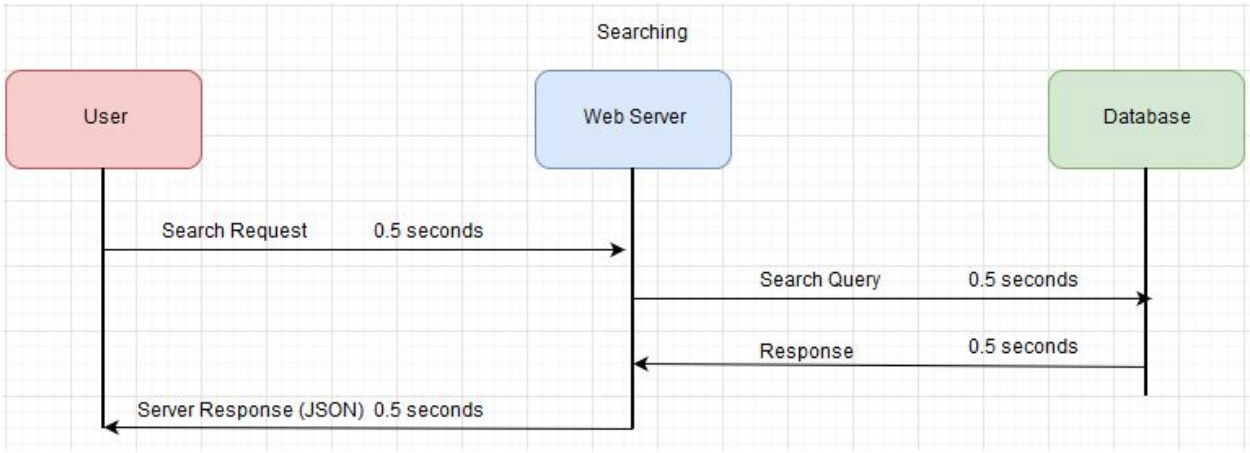


Figure 5.13.3 See below for description

The Searching Time UML Diagram represents the 2 second time requirement stated in section "3.4.4" of the SRS. This diagram breaks down the time requirement into the different stages of the action from start to finish of a successful attempt. See SDD sections 5.10.1 & 5.12.1.2.9

5.13.1.3 Loading Time Diagram

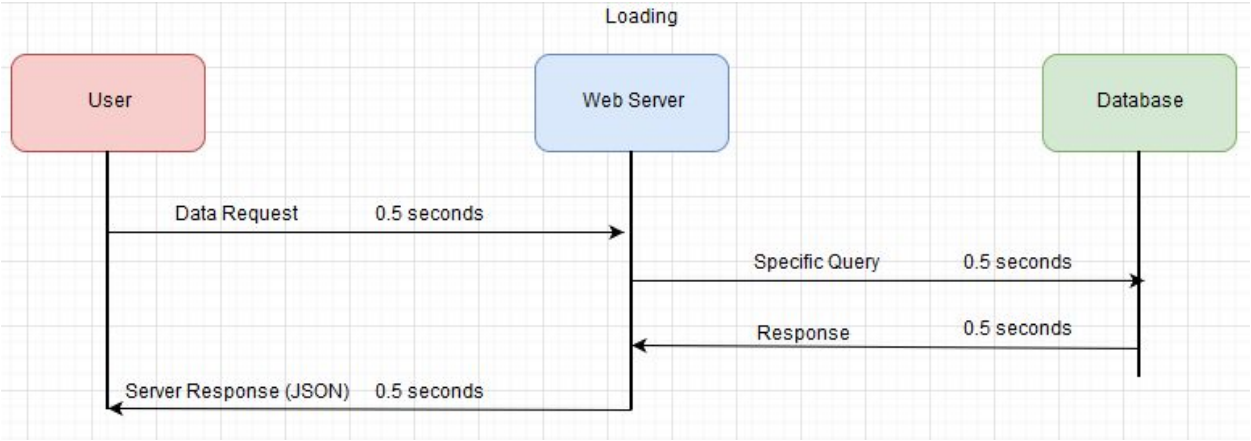


Figure 5.13.4 See below for description

The Loading Time UML Diagram represents the 2 second time requirement stated in section "3.4.5" of the SRS. This diagram breaks down the time requirement into the different stages of the action from start to finish of a successful attempt. See SDD sections 5.10.2.2& 5.12.1.2.9

5.13.1.4 Sharing Time Diagram

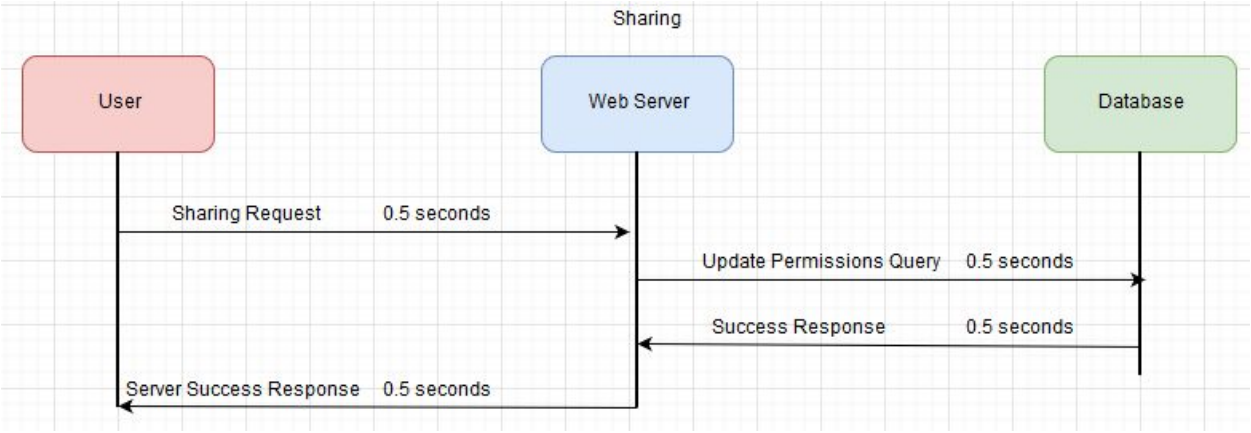


Figure 5.13.5 See below for description

The Sharing Time UML Diagram represents the 2 second time requirement stated in section "3.4.7" of the SRS. This diagram breaks down the time requirement into the different stages of the action from start to finish of a successful attempt. See SDD sections 5.10.2.3 & [Algorithms]

5.14 Design Rationale

Certain design decisions were made in advance of the creation of this document:

1. NearlyFreeSpeech.net shall be used for the application hosting platform (SRS 3.6.4.5).
2. Structured Query Language shall be used for application data management (SRS 1.3.1.1).
3. Python shall be used for application server side programming (SRS 3.1.3.2.1.1).
4. Javascript shall be used for application client side programming (SRS 1.3.1.4).
5. HTML5 shall be used for application structured document markup (SRS 1.3.1.4).
6. CSS shall be used for application presentation style definitions (SRS 1.3.1.4).
7. Database structure shall mirror that of application software classes (SRS 3.1.3.2.1.1).
8. Single Page Application architecture shall be used for the application (SRS 3.6.1.1).
9. The AngularJS front-end web framework shall be used for the application (SRS 3.6.1.1).
10. FirebaseUI shall be used for application user authentication (SRS 3.1.3.1.1.3).

5.14.1 Hosting Platform

There are many options available for hosting a web site, some even free. Many of those can be used to host a static website (no programming allowed on the server side, just hosting of HTML, CSS, and Javascript files that can be loaded into a site user's web browser). While much can be accomplished in this way, it is not sufficiently flexible for a modern professional web application.

Some hosting providers advertise insanely low prices with "unlimited" amounts of storage and bandwidth, but hide limitations in the fine print of their terms of service. They are shared platforms that try to cram as many customers as possible into a single web server, and if an application actually tries to use their "unlimited" offerings one can quickly discover a hosting plan has been cancelled for violation of their terms of service. Alternatively, the application might be negatively impacted by other sites on the same server, preventing users from having a quality experience.

At the other extreme one can purchase lots of expensive equipment, software licenses for databases, operating systems, development platforms, etc, and pay for dedicated bandwidth. While this provides ultimate flexibility of system and platform configuration, one can easily spend thousands or even millions of dollars before accepting a single user connection. This is untenable for a lean start up.

An alternative to buying and deploying one's own hardware and system software is using cloud based resources. Many providers exist, such as Amazon, Google, and Microsoft, and they can be had for very modest costs. By using free / open source software, much of the expense is eliminated, but one still often has to spend a lot of time setting up and configuring systems that, to be done properly, requires expertise that is beyond even technically minded people. The

basics may be simple, but security and updating considerations can quickly overwhelm the initial naive setup costs.

Between these two extremes exist shared hosting platforms that allow one to pay for resources actually used, but offload much of the system configuration and maintenance overhead to experts without having to pay their full time salaries. Heroku and NearlyFreeSpeech.net are two of many examples of such platforms. Heroku requires one to use their infrastructure in a very particular way, using the tools they provide to create applications, update applications, and so on.

NearlyFreeSpeech.net is a shared hosting platform that doesn't require much in the way of custom tool usage. It is setup to provide a shared but isolated & secure hosting environment where individual sites have minimal impact on others, and can be migrated automatically between servers when necessary to minimize the negative impact of bad actors. They provide system administration expertise, but provide little if any support for application development or deployment. This is beneficial to an organization that has application expertise but lacks the talent or resources to properly manage system configuration or administration tasks.

Because one pays for what one uses, NearlyFreeSpeech is not motivated to cancel a hosting contract if a site becomes successful and uses a larger than average share of system resources. This allows a lean startup to host a website and database for less than 10 cents per day if it has the bad luck of not attracting a single user, and only paying for things such as CPU, RAM, and bandwidth as it is consumed.

Some budget hosting providers greatly limit the programmability of their platform, perhaps only offering a language such as PHP. NearlyFreeSpeech.net provides all mainstream languages by default, so an application can be written using whatever language best suits the project. Custom processes can be run as well for more esoteric needs, though that is unlikely to be strictly necessary for this application. One could even write a server side implementation in a fully compiled language such as C++ if so inclined. Otherwise, the options provided directly by NearlyFreeSpeech.net are fairly spartan (effectively only one database option is provided vs the 14 options provided by Amazon, for example). This is intended to minimize cost, providing the least expensive functional platform possible.

The low startup cost of NearlyFreeSpeech.net is balanced by the higher than average cost for resources consumed. However, those costs are offset by the reduced system administration expense for all but the largest of websites. By utilizing standardized and/or common technologies, it will be possible to migrate to a more traditional cloud or on-premises solution when and if the application grows to a point where that makes financial sense. Until that day, NearlyFreeSpeech.net provides an affordable solution to early application needs.

5.14.2 Data Management

There are many database platforms available, from simple “pile of files” data stores, to document-centric schemaless data stores, to fully relational databases. By far the most common and portable solution is a relational database utilizing SQL (Structured Query Language) for data management and access tasks. While not all SQL-based database servers support identical dialects of SQL, they are generally close enough that migration from one to another isn’t too difficult if required.

NearlyFreeSpeech.net provides MariaDB, a SQL based engine designed to be largely compatible with its spiritual predecessor MySQL. While it would be possible to deploy some custom database solution on NearlyFreeSpeech.net, using the supported MariaDB is the most cost effective solution that would most easily transfer to another platform when and if it became necessary.

5.14.3 Server Side Programming

Many programming languages exist that can be effectively used on the server side of a web application. Some of the most common include:

1. C
2. C++
3. C#
4. Java
5. Javascript
6. PHP
7. Ruby
8. Perl
9. Python

C & C++ provide perhaps the most potential for memory and performance efficiency, but have a longer development, change, and test cycle.

C# & Java provide huge frameworks of functionality, but still require a lengthy compile, deploy, and test cycle.

Javascript is becoming more common on the server, largely through the Node.JS platform. Many find the programming techniques used to conform to its single threaded nature difficult to contemplate and use.

PHP, Ruby, and Perl have been used for many web application projects over the years but seem to be falling out of favor; most new work is apparently not targeting these languages.

Python provides the relative easy development and testing of an interpreted application (no lengthy compile, deploy, and test cycles are required). There are many libraries available for use with Python (a benefit that is similar to using C# or Java) avoiding the need to invent or find technologies (as might be necessary with C or C++). It is a rapidly growing language used in many domains, including web applications, and tools exist to pre-compile Python scripts to avoid some of the default-interpretation overhead that comes with typically non-compiled languages. Finally, Python is a free language, avoiding the expense that would potentially come with some options such as C# and Java.

5.14.4 Client Side Programming

When it comes to web application client side programming, Javascript is the linga franca of client side web programming languages; it is the one language that is understood by all commercially relevant web browsers. Thus any required client side programming shall target Javascript.

That being said, there are many ways to generate or utilize Javascript that do not necessarily entail writing Javascript directly. Many programming languages can be compiled or transpiled to Javascript, using Javascript as a sort of virtual machine language. Some of these languages include Microsoft's TypeScript, Google's Dart, and even C and C++ via emscripten. Additionally, libraries such as AngularJS allow one to take advantage of the flexibility of Javascript without having to write the entire application in Javascript. This flexibility allows an application to be written in any or many languages and still target the commercially relevant web browsers available today.

5.14.5 Structured Document Markup

While the world wide web and its associated software stack has evolved much since its first public release in 1991, one thing links the "ancient" and "modern" webs: the Hypertext Markup Language, aka HTML. It has been extended extensively over the ensuing 28 years, but is still built around the concept of open and close tags used to impart semantic meaning to fragments of text. The modern web is built around HTML5 as defined by the Web Hypertext Application Technology Working Group (WHATWG). All modern commercially relevant web browsers support HTML5 natively, and many older browsers that don't support it natively can be configured to support it through custom CSS files.

5.14.6 Presentation Style Definitions

Early versions of HTML mixed semantics and presentation in the same document. This made some aspects of web page design easier, but made using the same information in multiple formats more difficult. This wasn't a significant problem when everyone on the web had a

desktop style browser with a generally uniform screen size. The modern web must work in a number of contexts that were not contemplated by the early web. Most notably, it is desirable to have a single document that can be displayed on a number of device form factors, from large high resolution 4K displays down to tiny phone screens. For this (among other) reasons, Cascading Style Sheets, aka CSS, allows definition of style that is applied to the semantic markup in an HTML5 document.

5.14.7 Database Structure vs Program Structure

SQL serves well the purpose of managing data in the database, but it is not a good idea to sprinkle SQL through all portions of an application. For this reason, software classes will be defined to reflect the database structure in a way that can be utilized by the application and isolate the utilization of SQL to specific locations in the application source code. This reduces the opportunity for errors that would come from more SQL statements which typically can't be compile time tested, and eases maintenance costs if it becomes necessary to target a different database engine or modify the database schema.

5.14.8 Application Architecture

The Single Page Application architecture provides several benefits over the traditional Multiple Page Application architecture.

1. Speed and responsiveness: A traditional application has to endure a full server round trip for the entire web page, typically generating the page text dynamically. A single page application can download fragments of data and regenerate the page layout on the client, resulting in far less data transfer. Less data transferred generally equates to higher speed and responsiveness.
2. Caching remote data: A single page application can cache information on the client, minimizing the number of times the same data must be downloaded.
3. Linear user experience: Because the entire application is loaded up front, the single page application can provide a more linear user experience than a typical multiple page application.
4. Offline capability: A single page application can be used in an offline mode, holding on to program state to upload to a server until later when a connection becomes available.

The biggest downside to a single page application is the potentially lengthy initial load time. This is typically more than offset by the increased responsiveness of the application.

5.14.9 Application Framework

A single page application can be developed in pure HTML, CSS, and Javascript. It is more effective to use a well known, thoroughly tested, externally maintained framework to provide the core functionality. Some of the better known options are:

1. AngularJS (aka Angular 1)
2. Angular 2+
3. React
4. Vue
5. Ember

AngularJS is Javascript-based open-source front-end web framework, maintained by a community led by Google.

Angular 2+ is a Typescript-based framework by the same team responsible for AngularJS. While Typescript is a superset of Javascript that compiles down to plain Javascript, making it a viable option for the application, it does require a broader knowledge of application developers to use effectively.

React is a Javascript library for building user interfaces. It is incomplete on its own and requires using additional libraries for a complete single page application.

Vue is similar in concept to AngularJS, having been developed as a lightweight alternative framework. While respected, it does not appear to have as much community support as AngularJS.

Ember is another Javascript web framework. Like Vue, it is based on the model-view-viewmodel design pattern. It seems a more complete option that Vue, but still seems to have a smaller community.

Taking the following descriptions into account, it has been decided to use AngularJS as the framework for this application. It does not utilize an extra language, provides a simple means of developing single page applications, and is being maintained by an active community led by a well known company.

5.14.10 User Authentication

There are a number of providers of third-party authentication services, including:

1. Google Sign-In
2. Facebook Login
3. Sign in with Twitter

4. Apple
5. Microsoft
6. Yahoo

Ideally it would be nice to support as many authentication services as possible. This makes using the application more attractive to users as it is less likely they will need to create yet another account just to use the application.

To this end, the application will use FirebaseUI, a library built on top of the Firebase Authentication SDK that provides drop-in UI flows for use in web applications. This will provide access to all the listed authentication services (and more should we decide to enable them).