Software Engineering 14:332:452

Group #9

Full Report #3: ChefPal

Submission Date: 05/03/2021

GitHub

Team Members:

Shahir Ghani
Malena Bashar
Dymytriy Zyunkin
Daniel Samojlik
Amanda Phan

Michael Fong
Malak Khalifa
Aswathy Aji
Azim Khan
Nirav Patel

Table of Contents

Individual Contributions Breakdowns

	Shahir	Malena	Nirav	Aswathy	Dymytriy	Michael	Azim	Amanda	Malak	Daniel
Summary of Changes	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Customer Statements of Requirements	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Glossary of Terms	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
System Requirements	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Functional Requirements Specification	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Effort Estimation using Use Case Points	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Domain Analysis	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Interaction Diagrams	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Class Diagram and Interface Specification	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Systems Architecture and System Design	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Algorithms and Data Structures	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
User Interface Design and Implementation	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Design of Test	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
History of Work, Current Status, and Future Work	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
References	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Project Management	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%

Summary of Changes

- Added and removed terms in Glossary of Terms
- Updated the Fully Dressed Descriptions for Use Cases
- Removed some Casual Descriptions for Use Cases to Eliminate Redundancy
- Edit and Update the Use Case Diagram
- Added descriptions to Traceability Matrix in Domain Analysis
- Added Design Patterns and OCL in Class Diagram and Interface Specification
- Replace preliminary design with final UI design in User Interface Design and Implementation
- Updated tests of use cases in Design of Tests
- Added Project Management section for throughout the entity of project

1. Customer Statement of Requirements

a. Prospective Customer Perspectives

College Students:

Living at college is hard. Not only am I in a different environment from the one I've grown up in, but at the same time I have to manage my life by myself. I have to deal with the constant stress of classes, exams, and making sure all my assignments are completed and submitted on time. This means that I have to stay on top of all my work, while somehow managing my time and money efficiently. Of course, saving money means that I can't get a meal plan, simply because it costs way too much money per meal. According to the USDA, the average person spends about $163 to $367 per month. Meanwhile, with the meal plan, I'm spending $500 a month! It would save far more money by cooking meals for myself. That also means I have to stop eating out and ordering takeout all the time, but I don't want to get stuck simply making boxed mac & cheese in my microwave. Unfortunately, grocery stores in my area are limited, so I rarely have the ingredients needed to cook quality recipes at home.

It would be so much more convenient for me if there was something out there that could help me find recipes that I could make at home that only uses the limited amount of ingredients that I have. It would definitely be a more healthier option than ordering pizza every night, and it would also be a more financially safer option.

To solve my problems, I can use ChefPal. ChefPal would provide me with recipes based on the ingredients that I select in the app that I have on me right now. It would also provide me with recipes that require the same ingredients I have, but only require a few additional ingredients to make. Since the app automatically prioritizes showing me recipes with the least amount of extra ingredients to purchase, it's easy to find recipes that are financially feasible for me. Plus, I can save these recipes so that I can make them for a later time, which not only saves me money, but also time later on as well! Now, I have more time to focus on classes because ChefPal does all the hard work of matching me to healthy, affordable food options.

Amateur Home Cooks:

I just got married but I am not a good cook. I've never had to make food because when I lived with my parents, they did all the cooking. Plus, when I lived alone, I would just purchase takeout because whenever I did try to cook, I would go out of my way to buy too many new grocery items needed for a recipe and then end up cooking it wrong, ultimately wasting my money. Then I would have all these leftover ingredients that I didn't know how to put to use. My husband works long hours and I want to be able to make food for the both of us, but I don't know

how to be resourceful with the ingredients I purchase. An additional challenge is that my husband is vegan, so it is even harder for me to figure out what food to make for both of us, while using the same ingredients at home. As a new couple, we want to be able to not only save money when providing meals for ourselves and future kids, but also ensure our family is eating healthy and that we fulfill each person's dietary needs.

ChefPal would be the most useful solution to my problems. Especially by using the feature in the app that filters by dietary needs, I can browse recipes that are vegan, which works perfectly for my husband. Then, I can change the dietary needs so I can find recipes that include meat, which I personally enjoy eating. Plus, this fixes my problem of using ingredients wisely, so I don't end up wasting ingredients I've already bought. The app will find recipes that use ingredients already in our home, saving money for my family. Even if I don't have every necessary ingredient, ChefPal will provide recipes that require the least amount of additional ingredients to make a fulfilling dish. The amazing fact that I don't need to buy new ingredients can help me when practicing how to cook well with the food in my fridge that I already have and not waste extra money on grocery shopping. ChefPal will be the best companion to have that will help me provide healthy and delicious meals for my family.

Soup Kitchen Managers:

I am the manager of a local soup kitchen in my town and it is my responsibility to make sure I can provide food to the low income citizens in my community. We get ingredients from local grocery stores that are at the lowest price and also ingredients that we receive from donations. Sale items vary week to week based on nearby grocery stores so it is always a challenge of figuring out what to make for our less fortunate consumers. We also get our stock in bulk so when our volunteers use recipes, they need to find ones that will have serving sizes in large quantities for our large low income population. I have been a manager for many years here and have formed a bond with our usuals who need food. I believe that everyone has the right to eat tasty, healthy food, regardless of their economic status. I want to uplift our community by providing them with a variety of food options to ease their existing money related hardships.

ChefPal would be very useful for me as a manager to keep track of what groceries we already have in the soup kitchen because of how the app saves all the ingredients. The fact that this app can also show a recipe's measurement information and serving sizes will really help us tailor our cooking based on the amount of people we are trying to serve. Plus, we can create more meals in bulk for our community. The feature in ChefPal that provides information on the nearest grocery stores would be a great help in finding any missing ingredients quickly, since we can filter grocery stores from closest to furthest. As a soup kitchen and non-profit organization, it is really important we save money, so the purpose of using only the ingredients we have on us will help us financially. Since this app shows us a variety of recipes using the same ingredients

we currently have, we will be able to meet our goal of providing people in need with a variety of delicious foods. I can't wait to use ChefPal because it will make a significant positive impact in my lower income community.

Last-Minute Party Planners:

I love throwing impromptu parties for my friends and family but figuring out what to make can be tough. I do these so often that I don't want to end up repeating recipes and boring my invitees. Another problem I also have is when we decide to throw surprise parties after work, I don't have time to go out grocery shopping after work in time for me to be able to gather the ingredients that I need for the parties, so I have to figure out what to make pronto. It's never assured what could be in my fridge and sometimes I don't know what I could possibly make with the random ingredients I do have. When I am stumped on what to make, I end up having to buy takeout. This poses a lot of problems in their own way. Not only is it expensive, but I find it embarrassing because I am passionate about cooking and don't want to serve my party attendees food they could just go out and buy themselves.

How could ChefPal help me? ChefPal would be very helpful for me when I have to throw a party coming home right from work because it would provide recipes that only utilize ingredients that I already have. The variety of recipes ChefPal can offer with the same ingredients can also help me serve a variety of dishes to keep my parties lively. Internet recipes are so scattered and disorganized but this app has optional filters to even show recipes for the type of dish I am looking for (ex: snack, meal, dessert etc.). ChefPal would be a lifesaver and I can't wait to flex to my friends and family about how resourceful and tasty my cooking can be.

Fast-Paced Lifestyle Workers:

Due to the job that I have, I lead a very busy lifestyle. From the second I wake up, my day is a complete blur. Every moment I spend not doing work is a moment wasted, so I don't have any time to lose. I barely have any free time, and by the time I come home from work I am completely exhausted. Most of the time, I just end up buying a hot dog or a burger from a fast food restaurant to sustain myself throughout the day, but in the long term, not only am I spending more money on buying food than I want to, it is also very unhealthy.

Putting the time into searching through countless recipe databases online is an enormous waste that ChefPal eliminates by giving me clear and concise instructions for recipes I can cook and what are the nearest grocery stores near me that sell all of the missing ingredients. In addition, since everything is online these days, this app would be much more useful for me to

choose recipes I am interested in during my daily train ride to work with the “Save Recipe” feature it has, as well as being free on the app store unlike if I had to go buy a recipe book. With all the responsibilities I have related to work, this saves me time that could have been wasted trying to look for a feasible recipe once I am back from work. ChefPal takes the guessing and searching out of planning my dinners and gives me more time to enjoy the cooking process.

b. <u>Decomposition into Sub-problems</u>

- **Problem 1**: Meal planning is a challenging issue for anyone, both financially and time-wise.
    - Ordering in or taking out food on a consistent basis can add up to become pricey on an individual’s wallet.
    - Purchasing new ingredients in bulk is expensive.
    - Some people don’t have enough time to regularly buy new ingredients and plan for different recipes.

- **Problem 2:** Logistics of grocery shopping planning while balancing other responsibilities.
    - Time management can be very hard, especially for people with a busy work schedule, such as a stock broker, or ER doctors/nurses. Their work doesn’t give them enough time to plan out their meals and physically purchase the necessary groceries for the few recipes they do know how to make.

- **Problem 3:** It is a tedious process to find recipes with one’s matching ingredients.
    - Recipe books are often wordy and can cater to a particular cuisine or have specific ingredients required, which makes them not an optimal choice for everyday cooking, which may call for a variety of things.
    - Internet recipe resources are scattered, unorganized, and filled with pop-ups and advertisements.
    - The cost of buying physical copies of recipe books is high.
    - Trying to find recipes that include only the ingredients that you currently have, if not the majority of ingredients you currently have, is very hard to find manually.

2. Glossary of Terms

- **Bugify Issue Tracker -** a simple PHP issue tracker, designed to offer powerful bug tracking capabilities in an easy to use system.
- **Cuisine** - type or style of cooking and food, particularly relevant to a particular region or country.
- **Cloud Firestore -** Cloud Firestore is a cloud-hosted, NoSQL database that your iOS, Android, and web apps can access directly via native SDKs.
- **Dart** - programming language designed for client development; an object-oriented , class-based, garbage-collected language with C-style syntax.
- **Dietary restrictions -** constraints on a users food intake due to allergies/diabetes, religious reasons, or preference.
- **Figma** - a graphic editing tool to help design the user interface of our mobile application. Is compatible for Android and IOS.
- **Firebase** - a Backend-as-a-Service app development platform that allows for hosted backend services; will be used for user registration and authentication for this purpose
- **Flutter** - Google's UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase
- **Google Places API** - service that returns information about places using HTTP requests. Places are defined within this API as establishments, geographic locations, or prominent points of interest.
- **Ingredients** - any food, spice, or substance that is used and combined in order to form a certain dish.
- **Recipe** - a set of instructions on how to create a certain dish; usually includes a list of ingredients required for the dish.
- **Spoonacular API** - an API that allows you to access a vast collection of recipes and their attributes, such as their ingredients and type of cuisine to allow recipe searching and filtering specifications defined by the developer implementing the API.

3. System Requirements

a. Enumerated Functional Requirements

Note: Priority 5 is more vital than Priority 1

Identifier:	Priority:	Requirement:
REQ-1	5	As a user, I should be able to login using my unique username and password.
REQ-2	4	I should then be able to check off my dietary restrictions (vegetarian, vegan, keto, etc.) to save towards my profile.
REQ-3	3	As a user, I should be able to set a distance preference for grocery stores.
REQ-4	5	As a user, I should also be able to input my current available ingredients.
REQ-5	5	I should be able to see a list of recipes, prioritizing the ones that require the least amount of missing ingredients. IE: starting with recipes that have zero extra missing ingredients, so they use all the ingredients I have in my possession.
REQ-6	4	I should be able to view recipes that are within my dietary restrictions that I have previously specified in the profile setup.
REQ-7	3	As a user, I should now be able to filter recipes based on cuisine.
REQ-8	3	I should next have the ability to filter recipes based on type of meal (breakfast, dinner, snack, etc.).
REQ-9	3	I should be able to filter recipes based on preparation time.
REQ-10	3	I should be able to filter out recipes based on calorie and macronutrient count.
REQ-11	4	Now that I've seen a list of options, I should be able to save any recipe into a different tab organized into customized lists.
REQ-12	5	I get to see a display of all the recipe's required ingredients, preparation time, serving size, calorie count, and list of steps to make the meal upon clicking on a recipe.

REQ-13	5	When I click on a missing ingredient in a recipe I'm viewing, I should be redirected to a generated list of grocery stores that sells the missing ingredient
REQ-14	3	I should be able to filter grocery stores based on proximity.
REQ-15	5	I should have access to my personal profile where I can change login information, dietary preferences, and my distance preference for grocery stores.
REQ-16	2	If there are any bugs or issues, I should be able to report them through a "user report" feature.
REQ-17	3	I should be able to search all the recipes in the database based on a keyword I enter.

b. Enumerated Nonfunctional Requirements

Identifier:	Priority:	Requirement:
REQ-18	4	The application should be compatible with IOS and Android OS.
REQ-19	4	The application should be able to handle 500 daily users and can be scaled as needed.
REQ-20	5	The application should keep user information private by securely storing passwords and other sensitive information.
REQ-21	4	As a system, all stored and collected data should be secure.
REQ-22	3	The application layout should be aesthetically appealing and minimalistic with clutter free UI.
REQ-23	4	The application should operate at 60Hz or120 Hz for supported devices.
REQ-24	3	The application should be easy and simple to download and navigate for nearly any person who has access to a smartphone and internet.
REQ-25	3	As a system, user requests and issues should be checked daily and addressed as soon as possible.
REQ-26	3	As a system, unit testing should be done after any changes to

		the system and before any version updates releases.

c. User Interface Requirements

Identifier:	Priority:	Requirement:	Graphic:
REQ-27	5	The sign up page for a new user.	
REQ-28	5	The login screen for a returning user.	

REQ-29	4	If the user is new, then there is a list of dietary preferences to check off from.	
REQ-30	3	If the user is new, then there is a preferred distance range for grocery stores to select.	

REQ-31	5	A list of available ingredients, which are categorized by dairy, vegetables, meat, etc., to check off from in the "Basket" tab.	
REQ-32	5	A list of recipe titles you can scroll through in the "Search" tab. There are filters included at the top, such as cuisines, calorie and macronutrient count, preparation time, and type of meal (breakfast, snack, lunch, dinner, dessert, etc). A given recipe that a user clicks on will have a heart next to it to save the recipe.	

REQ-33	5	When a user clicks on a recipe to view, the recipe includes its required ingredients, preparation time, serving size, calorie count, and numerated steps to make the meal.	
REQ-34	4	If there is an ingredient in a given recipe that a user doesn't own, it will be noticeably clickable and will redirect to grocery store locations that offer that ingredient.	

REQ-35	3	Generated list of grocery stores with filters at the top, such as price and proximity, appear when the missing ingredient is clicked.	
REQ-36	2	"Profile" tab is where a user can change login information, dietary preferences, and distance preferences for grocery stores.	

REQ-37	2	“My Likes” tab where the liked recipes are saved in an “All Likes” list. The user’s customized lists also appear here with an option to create more customized lists.	 

4. Functional Requirements Specification

a. Stakeholders

Stakeholders refer to any person or legal entity who has a vested interest in the success of our system and application. A good way to organize these stakeholders is to categorize them into primary and secondary stakeholders. For our system, our primary stakeholders are college students, home cooks, fast-paced lifestyle people, and soup kitchen managers. They are all people who have a demand for quickly prepared home-cooked meals, and all face various external constraints. The college students and soup kitchen managers are largely limited by the products available to them as well as the cost of production. Home cooks and people with a fast-paced lifestyle, however, are more so limited by the time they have to go grocery shopping. Our secondary stakeholders are people who want to advertise their own recipes from their websites, companies that want to advertise their product, and advertising agencies who want to promote produce and grocery stores. Listing their produce and recipes on our application will greatly increase the consumer exposure to their goods. Internal stakeholders will include the project team and funders.

b. Actors and Goals

Actors are any people who interact with our application, both from the supply and demand side. The actors involved in this application and their respective goals are broken down into two groups - initiating users and participating users. Initiating users are actively interacting with the application and take advantage of the entire spectrum of the application's functionality. Participating users are all the entities that allow the application to work and provide the back-end functionality.

i. Initiating Actors

Actors	Actor's Goals	Use Case
User	Sign up for the app and create a profile. Add information and dietary restrictions to their profile.	Registration
User	Login to app and view profile.	Login
User	Edit ingredients available. Through this tab, a user can select or	Ingredient selection

	deselect what ingredients he or she has.	
User	While searching for recipes, users can filter to their likings and what they want to cook (sweet, savory, etc.).	Filtering ingredient generated recipes
User	Search for recipes on the entire database via a user inputted keyword	Recipe search
User	A user can save any recipe for future use.	Save recipes
User	A user can see a list of all saved recipes, as well as the customized lists that certain saved recipes can go into.	View saved recipes
User	A button next to a missing ingredient in a given recipe will allow the user to locate where they can purchase the recipe's required ingredient. It will redirect the user a list of grocery stores in the preferred radius that will sell the missing ingredient.	Grocery store locator
User	User can report any errors or problems they have faced throughout their process of selecting and finding recipes	Bug and error reporting
User	Users have access to this feature to change any profile information, location and dietary restriction info.	View/ Change Profile

ii. Participating Actors

Actors	Roles
Firebase	Will handle all user registration and authentication

Spoonacular API	Will take in all the users ingredients & filters and process their search.
Google Places API	Will be used to search for nearby grocery stores upon request by the user.
Bugify Issue Tracker	Will handle user bug reports.

c. Use Cases

i. Causal Description

UC-2: Primary Preference Selection

The user will be prompted to enter in their primary preferences, such as any dietary preferences or restrictions, as well as enter their location and a radius that they will be willing to go grocery shopping in.

UC-8: Save Recipes

After searching for recipes, the user will be able to add recipes that he or she likes to a custom list, which the user can create. The user will have a page of the app dedicated to recipe lists, which he or she has saved and will allow for more lists to be created.

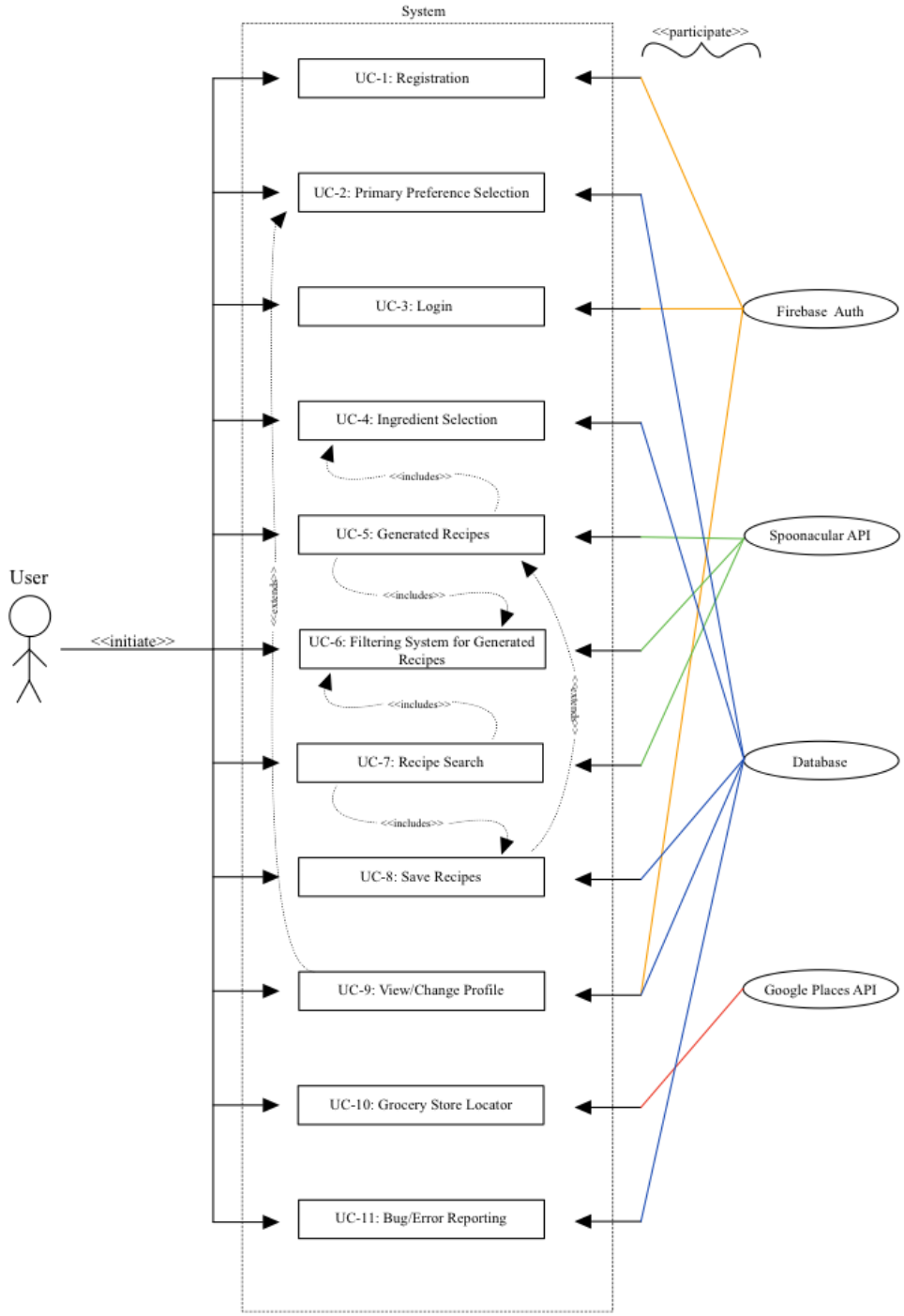
UC-9: View/Change Profile

The user will be able to change their login information, edit their dietary preferences and food intolerances. The user will also be able to alter their address in order to change the list of available grocery stores.

UC-11: Bug/Error Reporting

The user will be able to report any bug or error that occurs. This way, the developers can be aware of any problems that the user is facing and help approach them via a hotfix.

ii. Use Case Diagram



In the use case diagram, the relationships between the user, the use cases, the database, and the APIs used are displayed. All of the use cases are initiated due to the user's actions. In addition, some use cases will rely on other use cases in order to be functional and provide the best user

experience. Use cases will then interact and communicate with the database and APIs in order to provide the user an output.

iii. Traceability Matrix

The Traceability Matrix points to which use cases fulfill each requirement, and scales them to account for the priority of each use case.

Req't	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11
REQ-1	5	✔		✔								
REQ-2	4		✔							✔		
REQ-3	3	✔									✔	
REQ-4	5				✔							
REQ-5	5					✔						
REQ-6	4					✔		✔				
REQ-7	3						✔					
REQ-8	3						✔					
REQ-9	3						✔					
REQ-10	3						✔					
REQ-11	4								✔			
REQ-12	5							✔				

REQ-13	5										✔	
REQ-14	3										✔	
REQ-15	5									✔		
REQ-16	2											✔
REQ-17	3							✔				
REQ-18	4											
REQ-19	4											
REQ-20	5	✔		✔						✔		
REQ-21	4									✔		
REQ-22	3											
REQ-23	4											
REQ-24	3											
REQ-25	3											✔
REQ-26	3											
REQ-27	5	✔										
REQ-28	5			✔								
REQ-29	4		✔									

REQ-30	3	✔									✔	
REQ-31	5				✔							
REQ-32	5					✔	✔					
REQ-33	5							✔				
REQ-34	4										✔	
REQ-35	3										✔	
REQ-36	2									✔		
REQ-37	2								✔			
Total Weight	-	21	8	15	10	14	17	17	6	20	21	5

iv. Fully-Dressed Descriptions

UC-1: Registration
Related Requirement: REQ-1, REQ-3, REQ-20, REQ-27, REQ-30
Initiating Actor: User
Actor's Goal: To create a new account with a username and password
Participating Actors: Firebase
Pre-Conditions: This is the first time the user is using and operating the app
Post-Conditions: The user can now select their dietary preferences and their username and password will be saved to the database
Minimal Guarantees: The registration fails if the password doesn't meet the requirements, or if the user already has an account, or if another user has the same username.
Success Guarantees: The new user is stored into a database.
Flow of event for Success Scenario: ● → **User** selects the "Username" text box and enters in their preferred username. ● ← **Application** verifies that the username is unique. ● → **User** selects the "Password" text box and enters in their preferred password. ● ← **Application** navigates the user to personalized their home screen.
Flow of Events for Alternate Scenario: ● → **User** selects the "Username" text box and enters in their preferred username. ● ← **Application** finds the same username has already been used for another account. ● → **User** selects the "Password" text box and enters in their preferred password. ● ← **Application** alerts the user that the login credentials have already been taken and prompts them to pick a different username and password.

UC-2: Primary Preference Selection
Related Requirement: REQ-2, REQ-29
Initiating Actor: User
Actor's Goal: To add dietary information,restrictions, and distance preference to their profile
Participating Actors: Firebase
Pre-Conditions: The user has been registered in the system and login into the system using his/her authentication credentials.
Post-Conditions: The user can now get recipe suggestions based on the preferences and grocery locations.
Minimal Guarantees: The registration fails if the password doesn't meet the requirements, or if the user already has an account, or if another user has the same username.
Success Guarantees: The user preferences are stored in the database and the user's profile
Flow of event for Success Scenario: ● → **User** checks the different boxes that align with their preferences ● → **User** inputs their distance preference ● ← **Application** stores the preferences into the user's information ● → **User** confirms all the selections ● ← **Application** personalizes the user's suggested recipe based on the selections
Flow of Events for Alternate Scenario: ● → **User** has no preference and skips the selections ● ← **Application** alerts the user that a minimum of one box must be checked

UC-3: Log in
Related Requirement: REQ-1, REQ-20, REQ-28
Initiating Actor: User
Actor's Goal: To regain access into their ChefPal account
Participating Actors: Firebase
Pre-Conditions: The user has been registered in the system and log in into the system using his/her authentication credentials.
Post-Conditions: The user can gain access back to their account and begin searching for recipes based off of ingredients available to them
Minimal Guarantees: The log-in fails if the password does not match the username, or if the entered username is not in the database.
Success Guarantees: The user is logged into the system and is able to interact with the app.
Flow of event for Success Scenario: ● → **User** selects "Username" text box and enters in their personal username. ● → **User** selects the "Password" text box and enters in their personal password associated with that username. ● ← **Application** validates user authentication credentials against stored hash and prompts the user to their home screen.
Flow of Events for Alternate Scenario (Login Error): ● → **User** selects "Username" text box and enters in their personal username. ● → **User** selects the "Password" text box and enters in their personal password associated with that username. ● ← **Application** alerts the user that their login credentials do not match records and prompts them to re-enter login and password.

UC-4: Ingredient Selection
Related Requirement: REQ-4, REQ-31
Initiating Actor: User
Participating Actors: Firebase Database
Actor's Goal: Having the user enter in the ingredients that they currently have in their possession
Pre-Conditions: The user has been registered in the system and log in into the system using his/her authentication credentials.
Post-Conditions: Entering any preference regarding their meal such as meal type or cuisine of meal that they would like
Minimal Guarantees: If no ingredients are selected, then no recipes will be generated.
Success Guarantees: User will be able to view available recipes based off of any of the selected ingredients
Flow of event for Success Scenario: ● → **User** selects available ingredients from the provided menu. ● ← **Application** will show that ingredient has been selected ● → **User** selects the "Search" button
Flow of Events for Alternate Scenario: ● → **User** selects ingredient ● ← **Application** does not successfully show the ingredient has been selected ● → **User** issues a bug report

UC-5: Generated Recipes
Related Requirement: REQ-5, REQ-6, REQ-32
Initiating Actor: User
Actor's Goal: To select and filter out recipes based on their preset dietary restrictions and current ingredients, as well as other filters
Pre-Conditions: User inputted their own preferences and dietary needs in order to be able to generate a list of recipes based on the selected ingredients
Post-Conditions: Recipes will be pulled pertaining to user's selected preferences and current ingredients on hand
Minimal Guarantees: N/A (If there are no matching recipes based on the user's input, the application will suggest some other recipes based on previous history)
Success Guarantees: Users will be able to view a list of generated recipes based on their dietary preferences and selected ingredients.
Flow of event for Success Scenario: ● → **User** searches with selected ingredients ● ← **Application** will return a list of recipes that contain the most ingredients in common with the selected ingredients, where recipes lower on the list contain less amounts of common ingredients.
Flow of Events for Alternate Scenario: ● → **User** searches with selected ingredients ● ← **Application** does not successfully display results ● → **User** issues a bug report

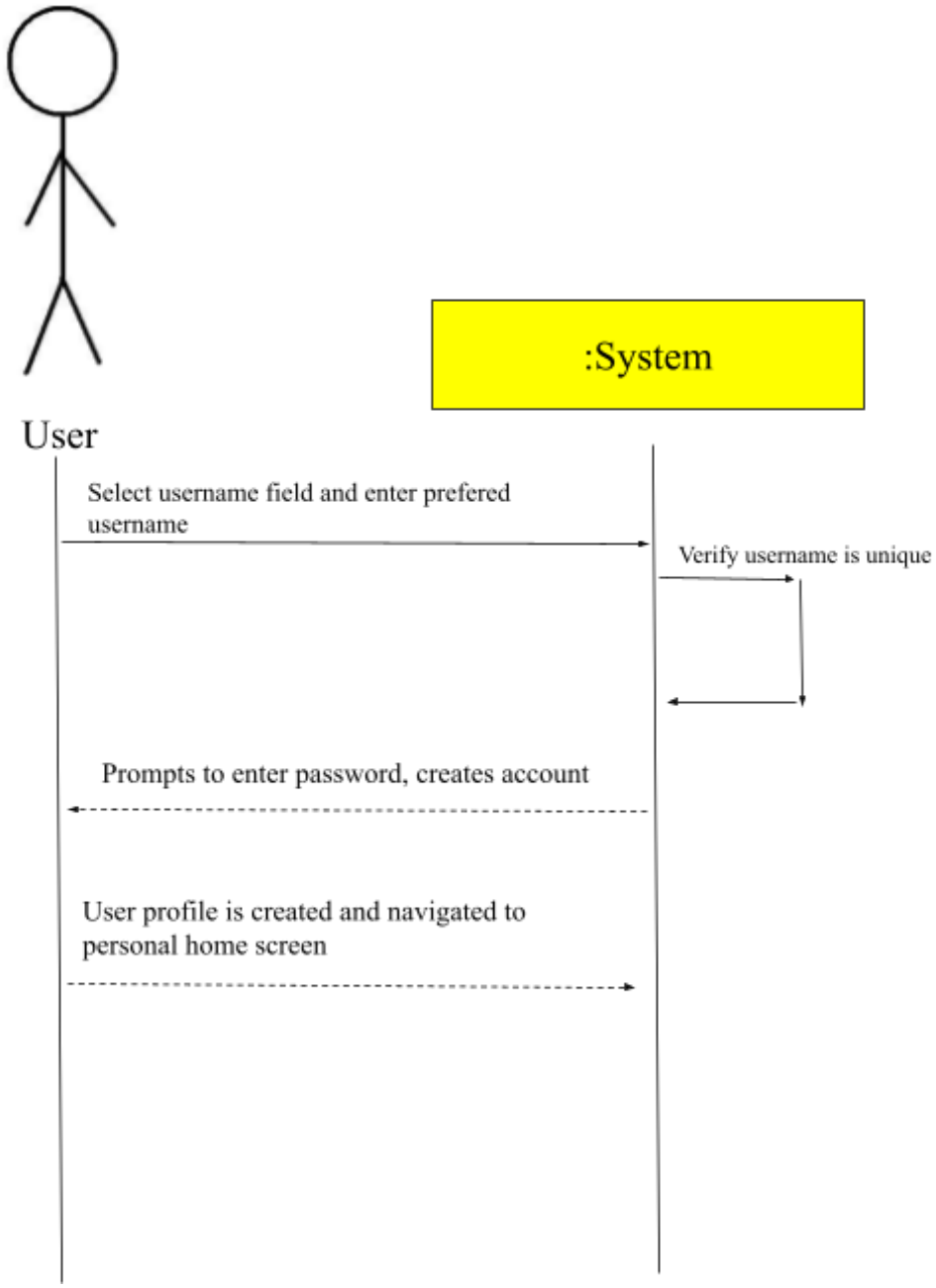
UC-6: Filtering Generated Recipes
Related Requirement: REQ-7, REQ-8, REQ-9, REQ-10, REQ-32
Initiating Actor: User
Actor's Goal: User will be able to filter through more categories, such as meal type, cuisine, prep time, calorie count, etc., to help generate more specific recipes.
Pre-Conditions: The application have generated recipes that correspond the user's ingredients
Post-Conditions: User gets to narrow down the results given by choosing what type of meal they want, which cuisine they prefer, whether they have a calorie count specifications, etc…
Minimal Guarantees: User will get generated recipes that do not include extra filtration.
Success Guarantees: User will get more filterized recipes based off of their selections.
Flow of event for Success Scenario: ● ← **Application** generated recipes based on users dietary restrictions and ingredients ● → **User** inputs preference filters such as type of meal, type of cuisine, prep time, calories ● ← **Application** will regenerate recipes based on the new filters
Flow of Events for Alternate Scenario: 1. Display Error a. → **User** filters with generated recipes b. ← **Application** does not successfully display results c. → **User** issues a bug report

UC-7: Recipe Search
Related Requirement: REQ-6, REQ-12, REQ-17, REQ-33
Initiating Actor: User
Actor's Goal: Allowing the user to look up any recipe based off of their search preferences Ex: User searches Shrimp Fried Rice
Pre-Conditions: User selects their dietary restriction and inputs available ingredients and preferences.
Post-Conditions: User will be able to see all available recipes for searched recipe
Minimal Guarantees: No recipes will output
Success Guarantees: User will be able to view recipes base on their search
Flow of event for Success Scenario: ● → **User** enters in a recipe to be searched with given keywords into a search bar ● ← **Application** successfully displays results ● → **User** scrolls through the result
Flow of Events for Alternate Scenario: ● → **User** searches with given keywords ● ← **Application** does not have any recipes related to that keyword search, or no matches occur ● ← **Application** does not not output any recipes

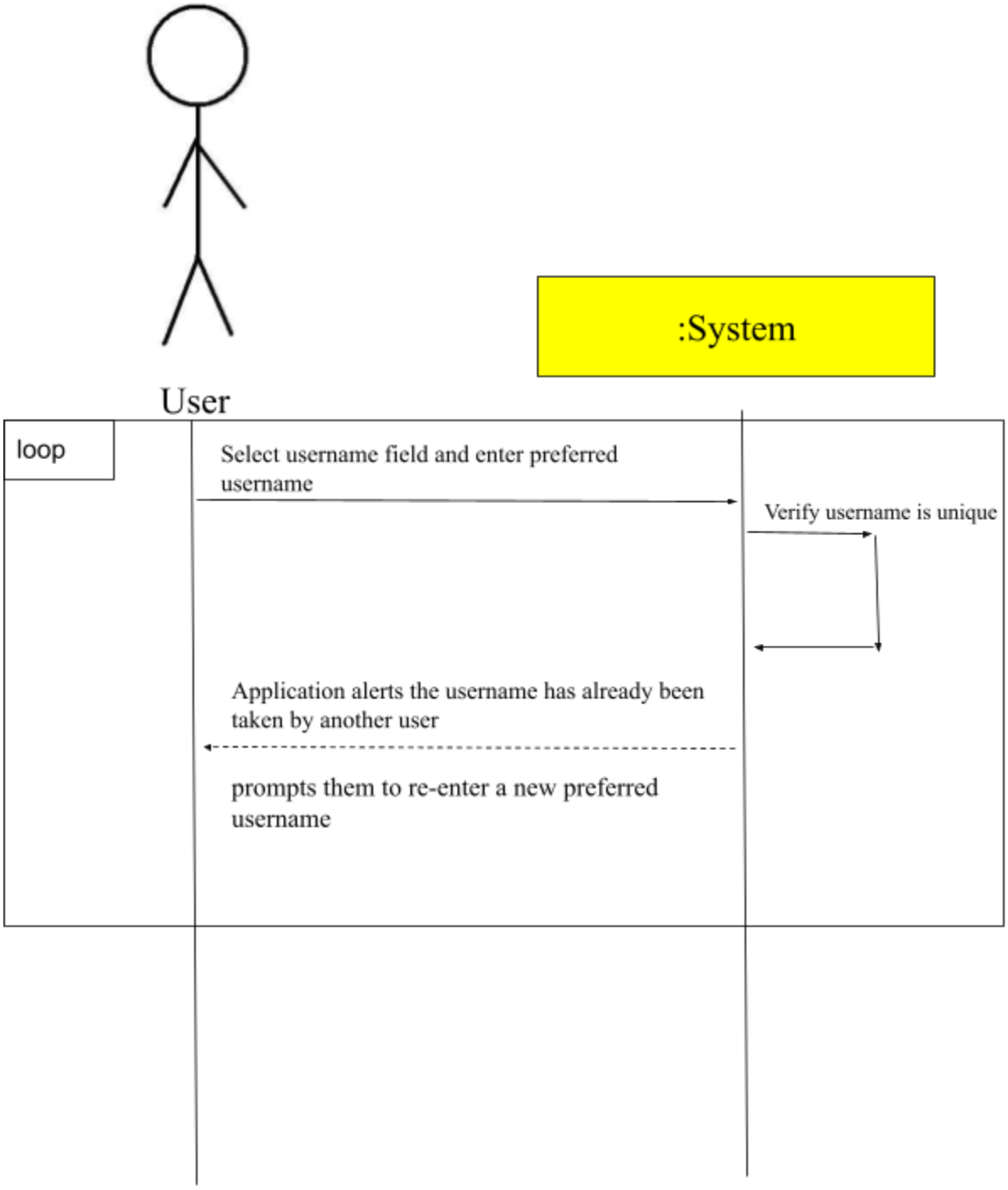
UC-10: Grocery Store Locator
Related Requirement: REQ-3, REQ-13, REQ-14, REQ-30, REQ-34, REQ-35
Initiating Actor: User
Actor's Goal: Locating a store that has the missing ingredients available to the user
Pre-Conditions: User has identified all of the ingredients that they currently have and given the fact that they have minimal ingredients, they need to go to the grocery store. User has also indicated their current location and the distance they are willing to travel for grocery shopping.
Post-Conditions: The application gives the user the name and address of the grocery store nearest to their location that has the necessary ingredients for the selected recipe.
Minimal Guarantees: No grocery store is available in the requested area.
Success Guarantees: The app will display nearby grocery stores with items that the user will need for their chosen recipe.
Flow of event for Success Scenario: ● → **User** selects a recipe ● → **User** presses find missing ingredients button if there are any missing ingredients ● ← **Application** successfully displays grocery store list. ● → **User** views the results
Flow → **User** selects a recipe ● → **User** presses find missing ingredients button if there are any missing ingredients ● ← **Application** prompts user to enable location services ● → **User** accepts the app's use of location services. ● ← **Application** successfully displays grocery store list. ● → **User** views the results

d. System Sequence Diagrams

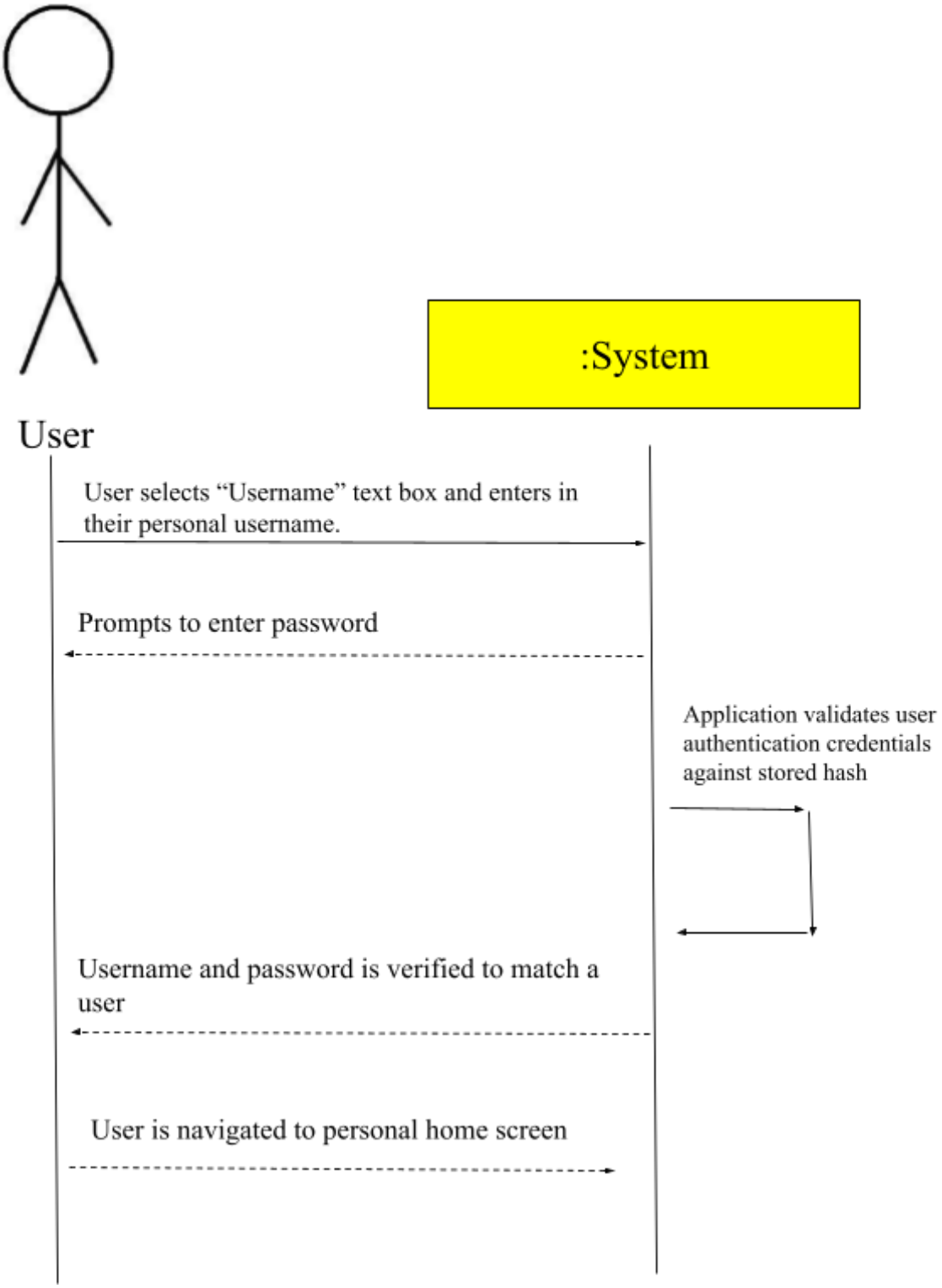
UC - 1 Sign- Up Main Success Scenario



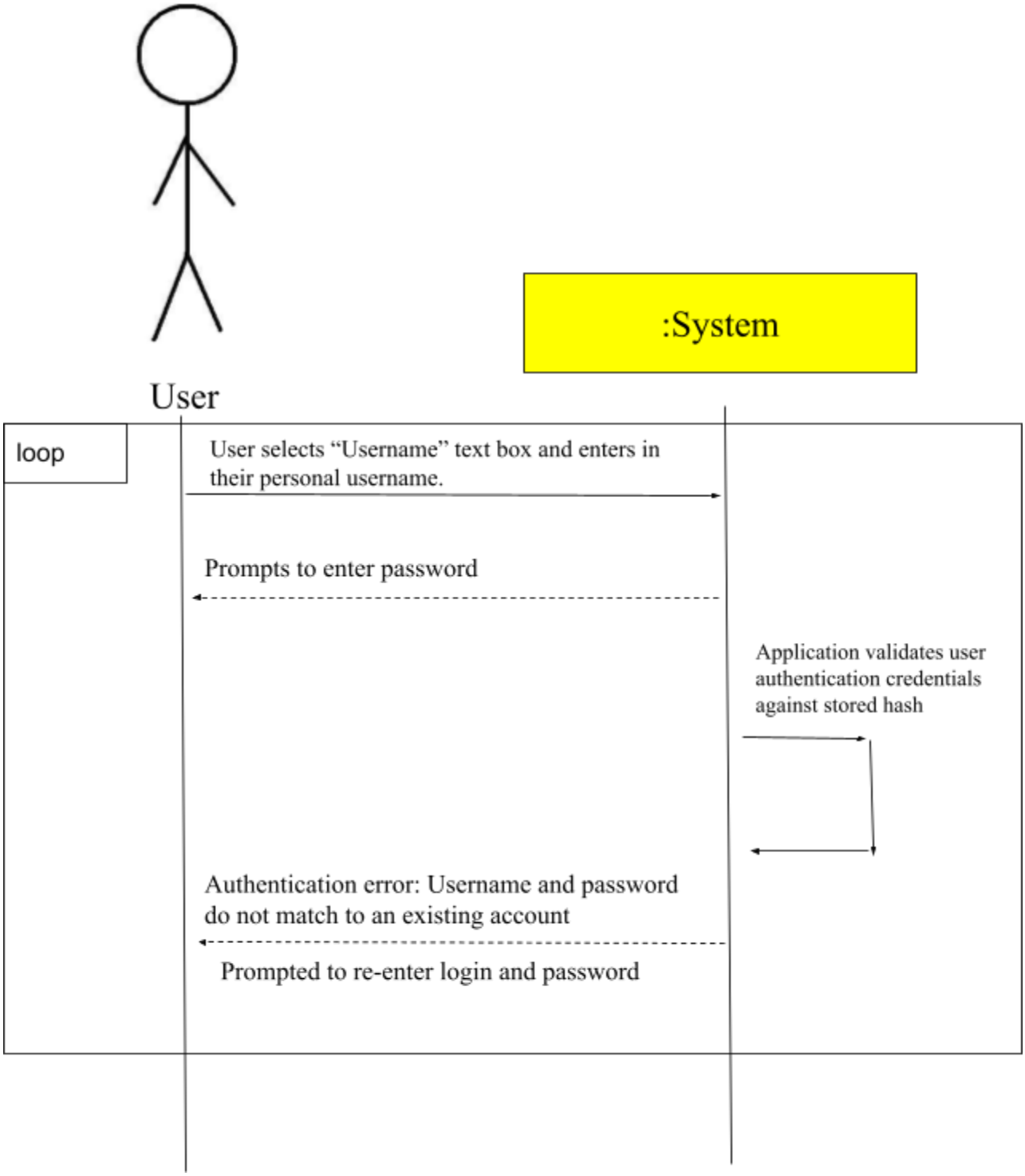
UC - 1 Sign-Up Alternate Success Scenario



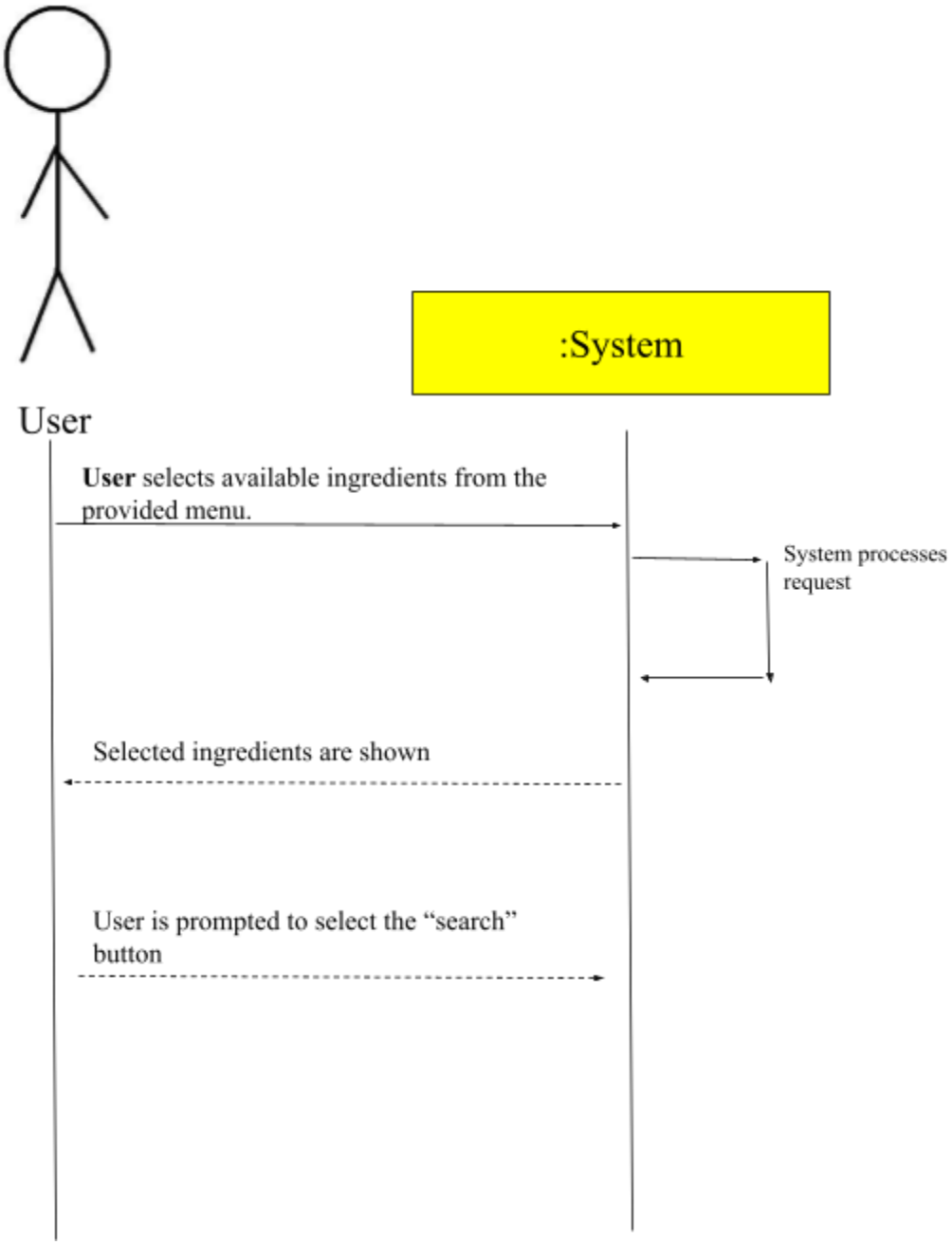
UC - 3 Login Main Success Scenario



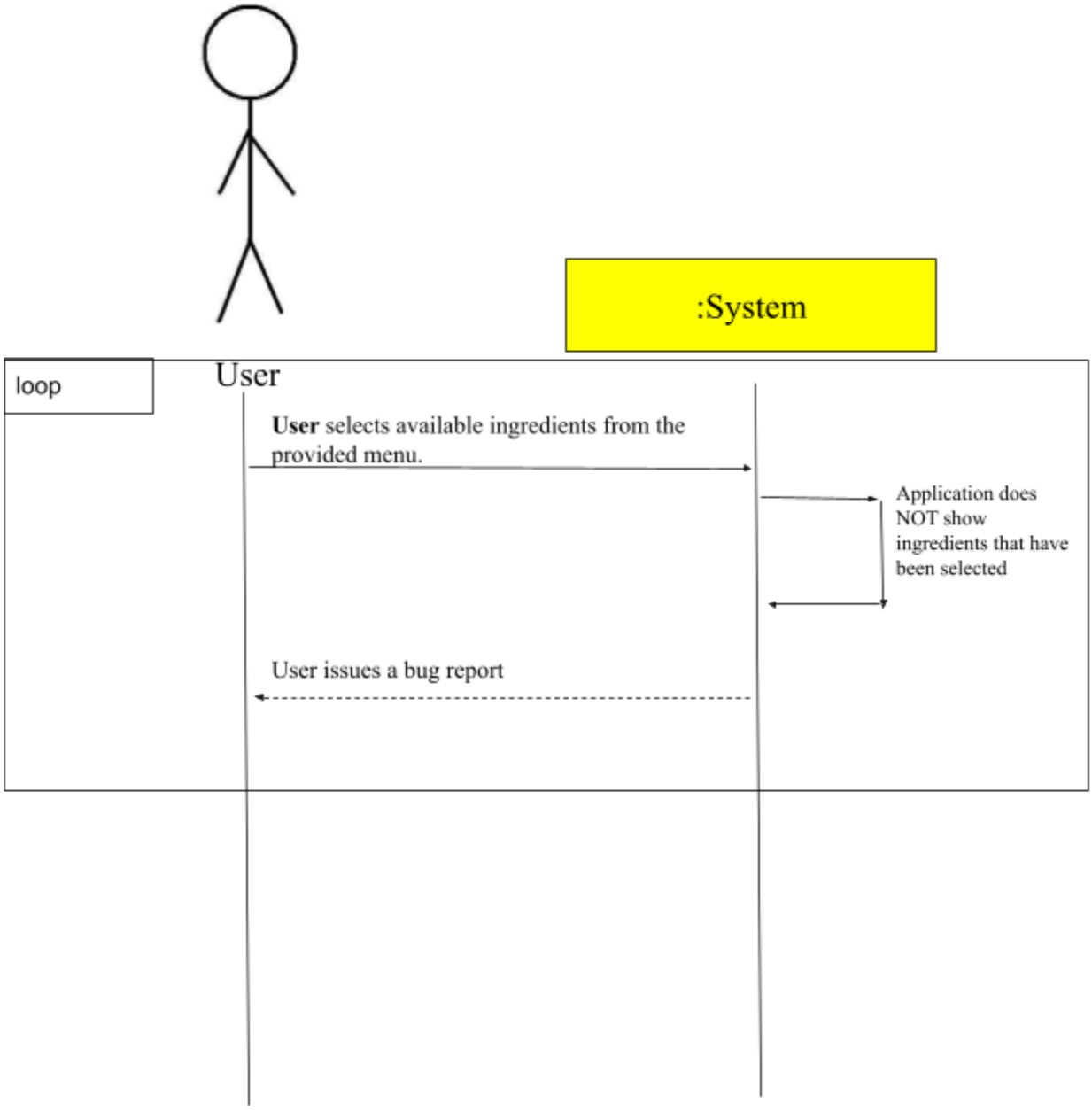
UC - 3 Login Alternate Success Scenario



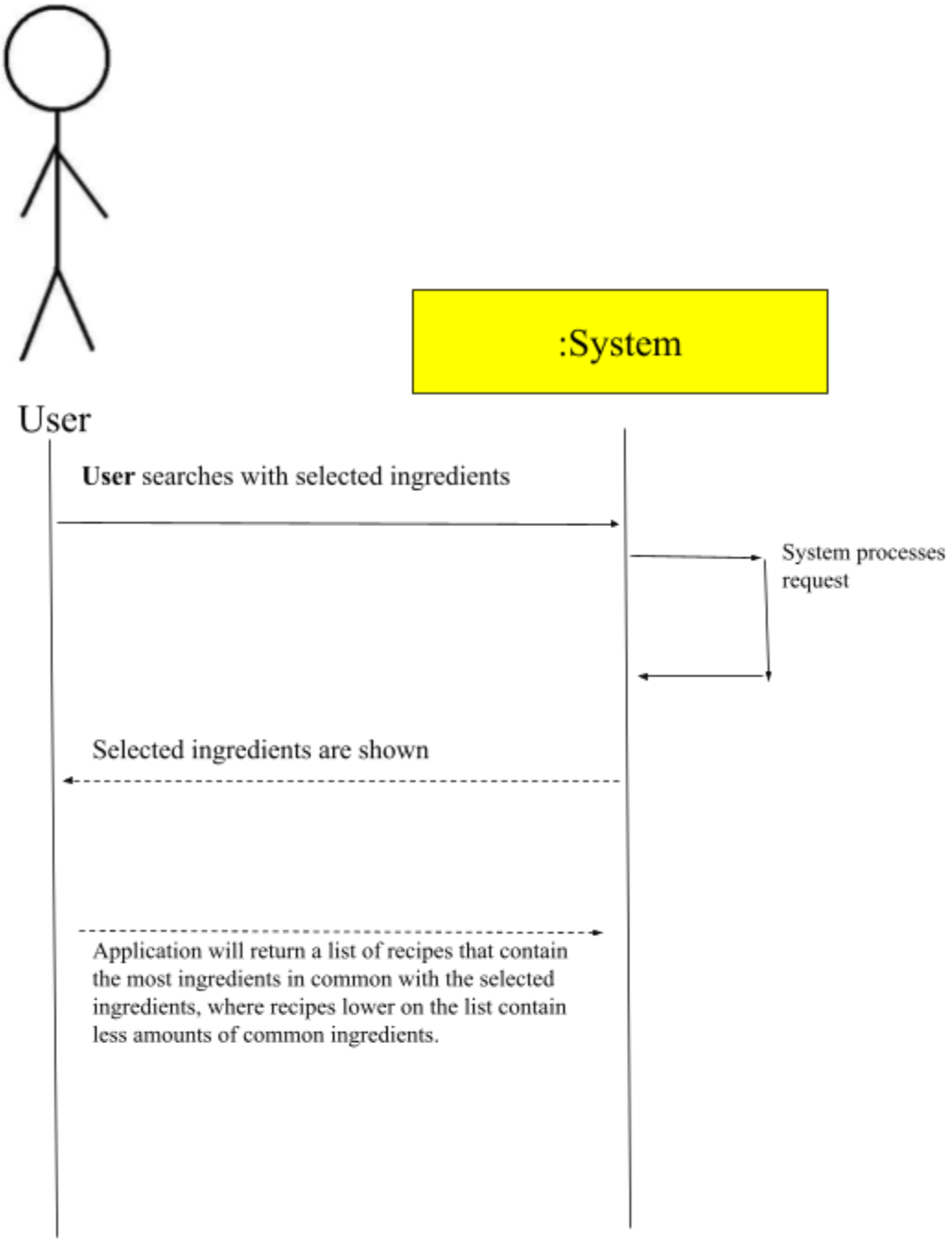
UC - 4 Ingredient Selection Main Success Scenario



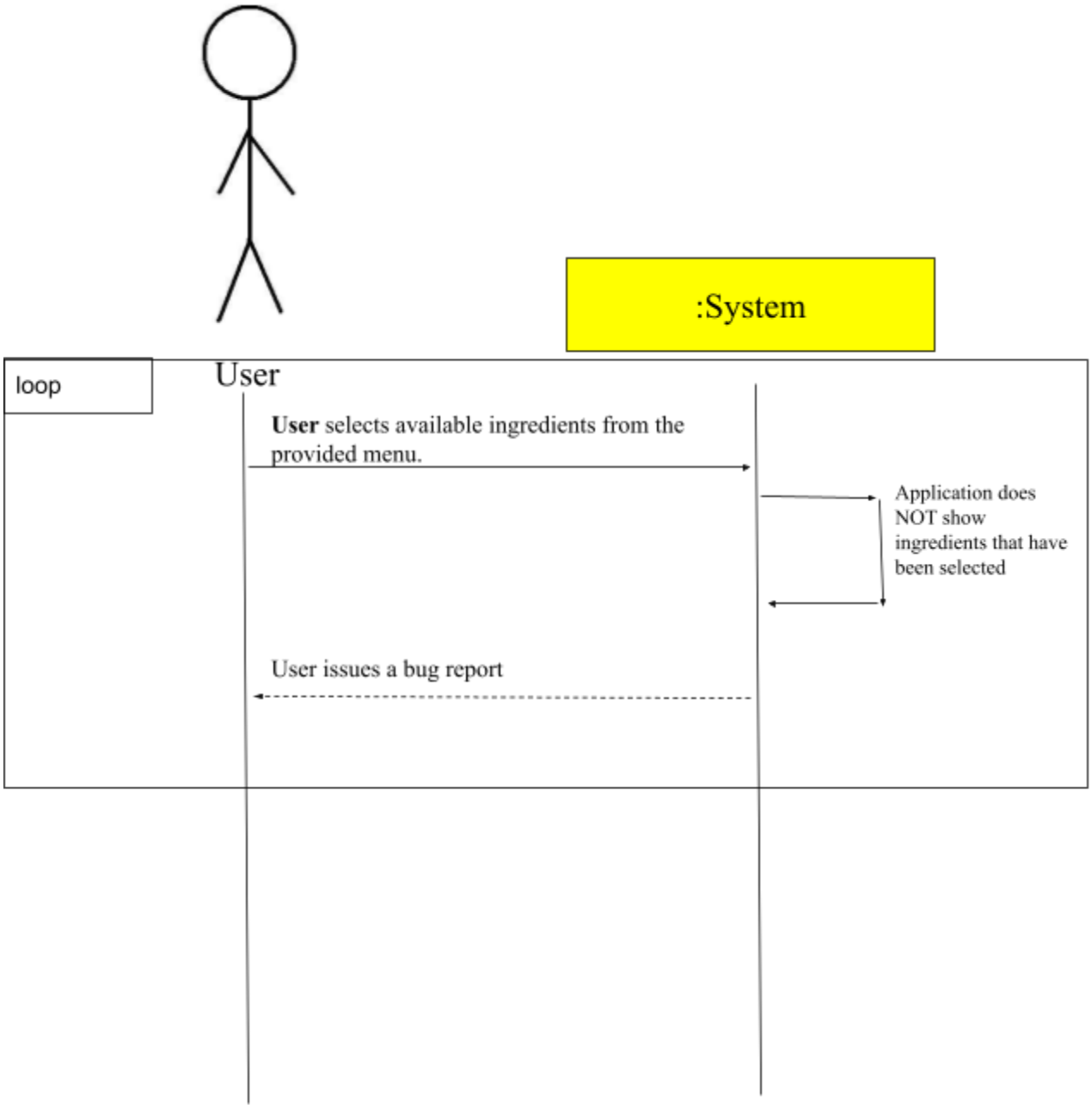
UC - 4 Ingredient Selection Alternate Success Scenario



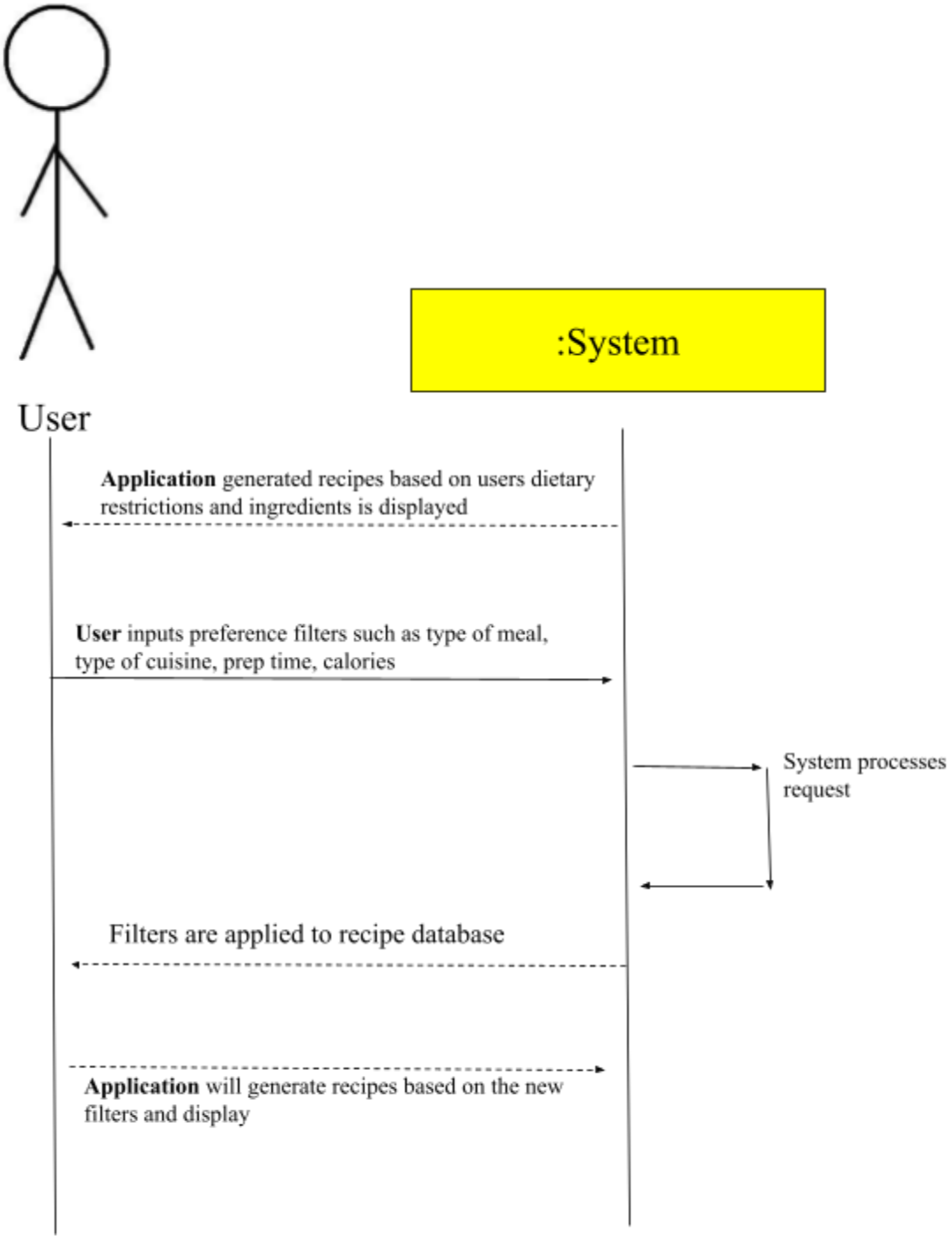
UC-5 Generated recipes Success Scenario



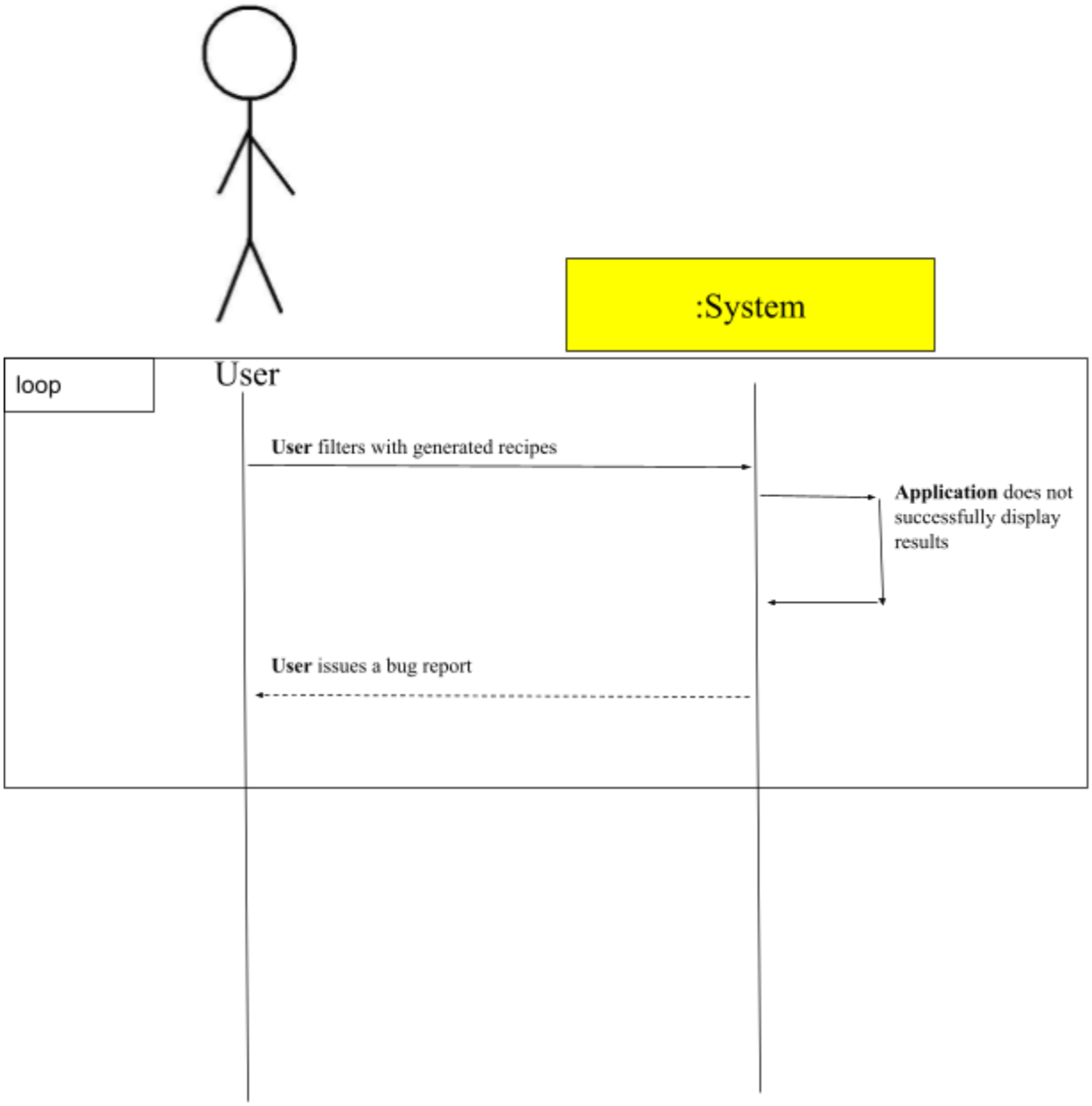
UC- 5 Generated recipes Alternate Success Scenarios



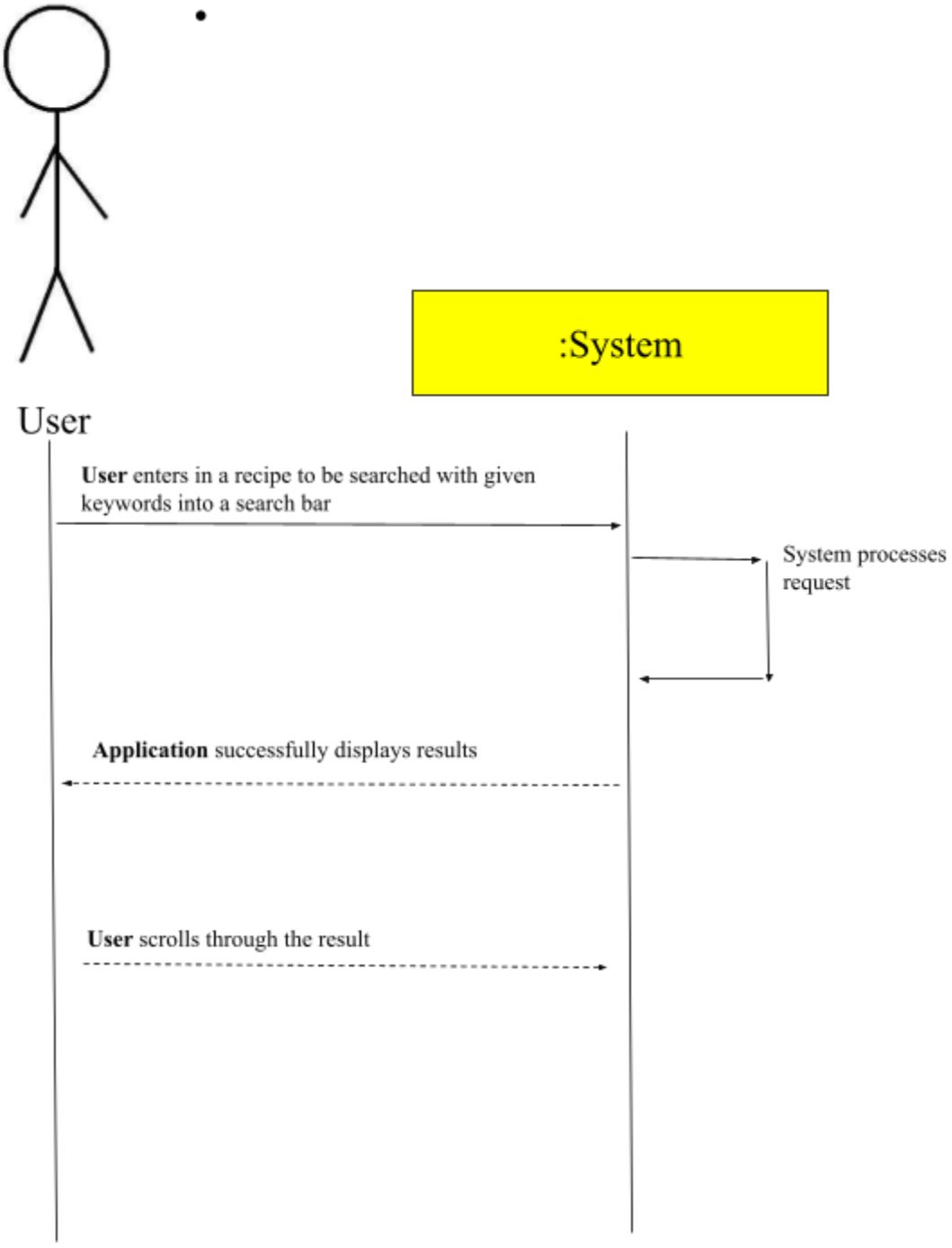
UC-6 Filtering Generated Recipes Success Scenario



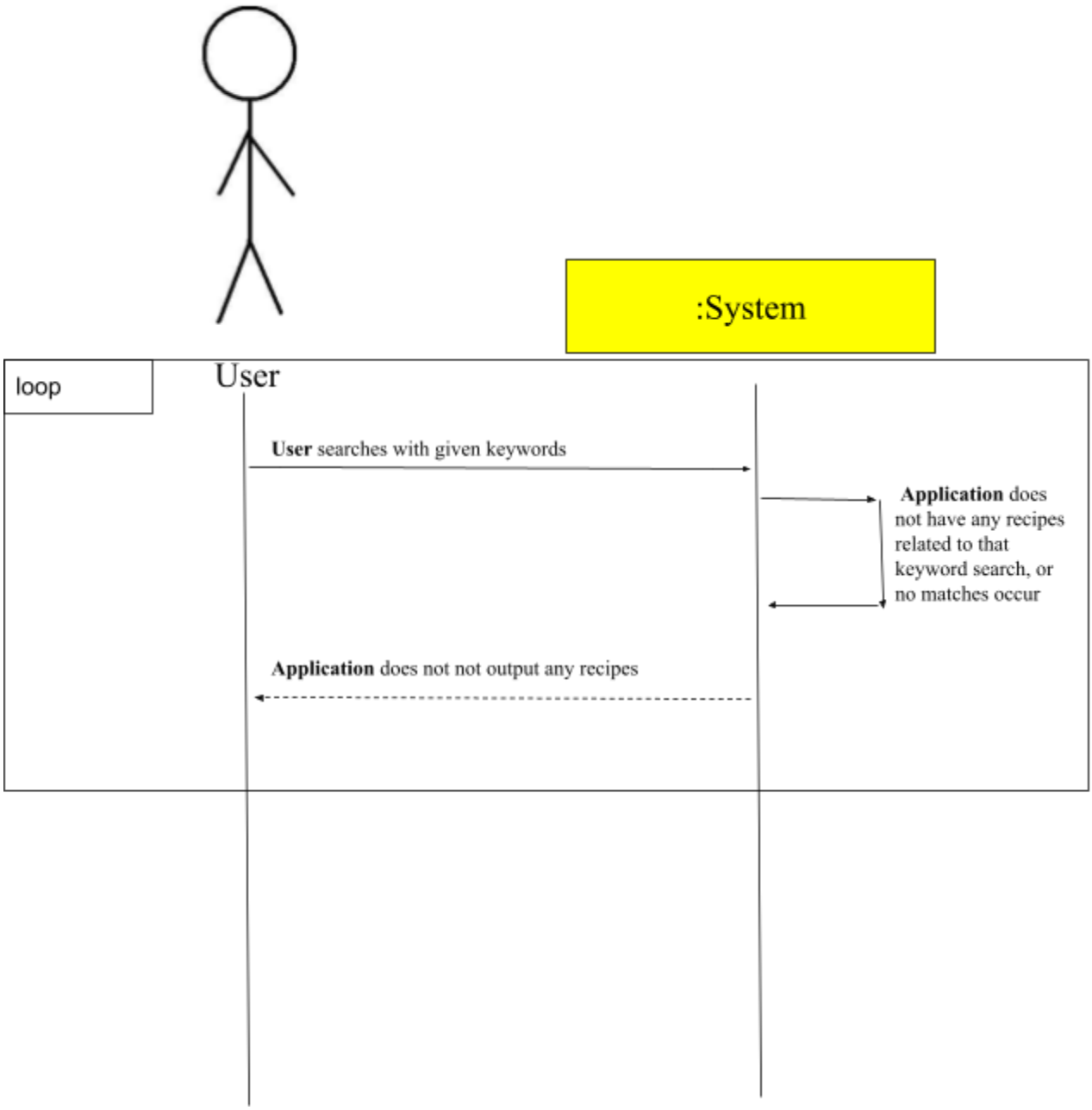
UC-6 Filtering Generated Recipes Alternate Success Scenario



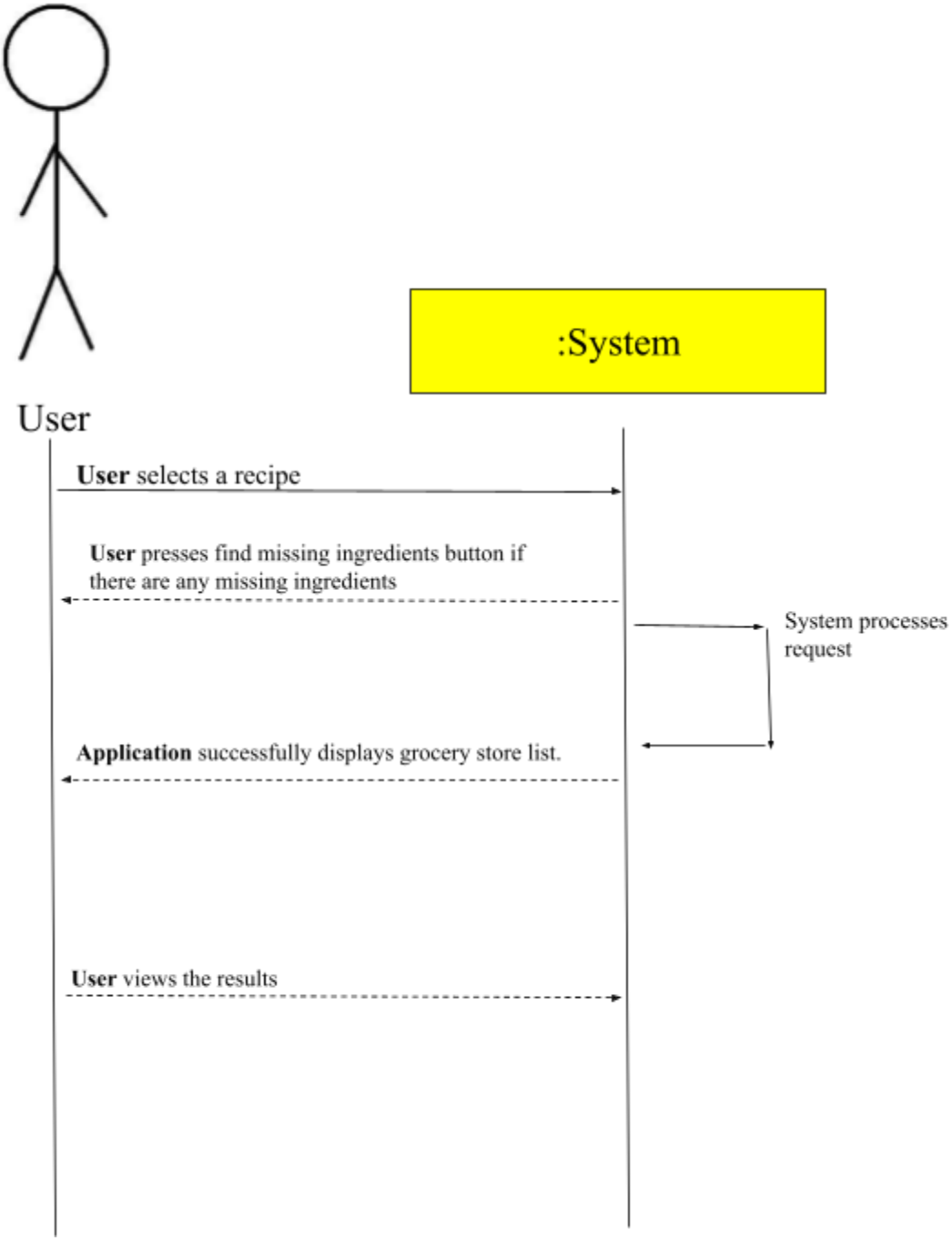
UC-7 Recipe Search Success Scenario



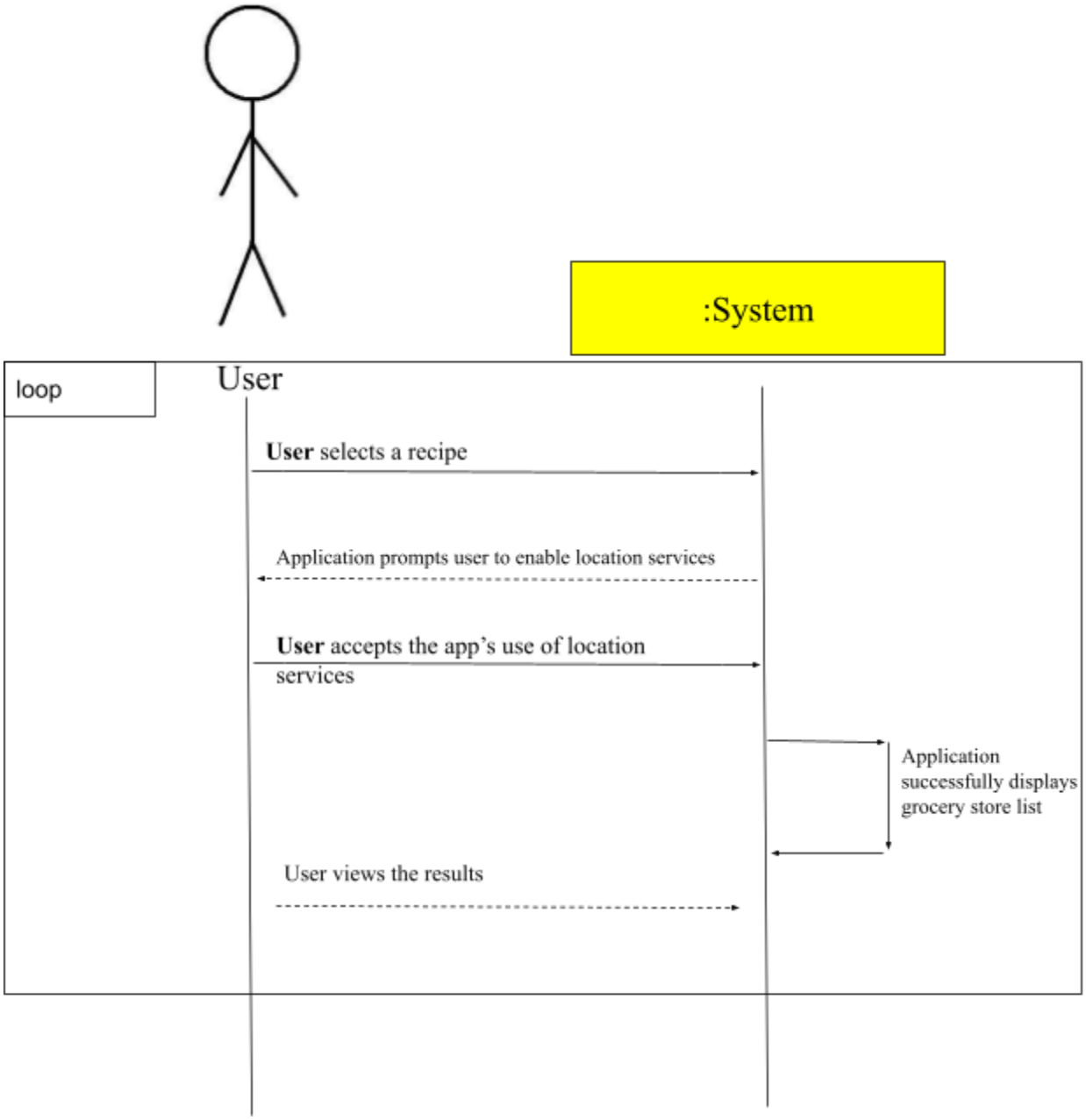
UC-7 Recipe Search Alternate Success Scenario



UC-10 Grocery Store Locator Success Scenario



UC-10 Grocery Store Locator Alternate Success Scenario



5. Effort Estimation using Use Case Points

a. User Effort Estimation

Note: N = user-interface navigation; D = clerical data entry

UC-1: Registration

- User clicks to open the application (N) → user clicks "Create New Account" (N) → user clicks for each textbox to input data for username and password (D) → user inputs data into each textbox (D) → user clicks "Create Account" once all required fields are entered (D)
- Total clicks = 1 + 1 + 1 * (# of textboxes) + (keystrokes for data entry for each textbox) + 1
- Maximum of 50% of clicks are for user-interface navigation; at least 50% of the clicks (keystrokes) will be for clerical data entry

UC-2: Primary Preference Selection

- User clicks which dietary preferences to apply (the user is directed to this screen after the user successfully creates a new account) (D) → user clicks "Submit" (D)
- User clicks slider to apply distance preference (the user is directed to this screen after the user successfully selects a dietary preference) (D) → user clicks "Submit" (D)
- Total clicks = (# of dietary preferences selected) + 1 + 1 for distance slider + 1
- 0% of clicks are for user-interface navigation; 100% of clicks are for clerical data entry

UC-3: Login

- User clicks to open the application (N) → user clicks username textbox (D) → user enters username (D) → user clicks password textbox (D) → user enters password (D) → user clicks "Log In" (D)
- Total clicks = 1 + 1 + (keystrokes for username) + 1 + (keystrokes for password) + 1
- Maximum of 5% of clicks are for user-interface navigation; at least 95% of clicks are for clerical data entry

UC-4: Ingredient Selection

- User clicks ingredients that are available (D) via:
    - Clicking drop down menu (N) → Selecting box next to ingredients in that menu to select the ingredient (D)
- Total clicks:
    - (# of drop down menus selected) + (# of ingredients within a certain drop down)
- Maximum of 50% of clicks are for user-interface navigation; at least 50% of clicks are for clerical data entry

UC-5: Generated Recipes

- User clicks on the "Search" icon at the bottom tab (N) after all ingredients are selected, to which the user is directed to a screen with the generated recipes. User can click on certain recipes to read them in detail
- Total clicks = 1, depends on if user clicks recipes to read in more detail
- 100% of clicks are for user-interface navigation; 0% of clicks are for clerical data entry

UC-6: Filtering System for Generated Recipes

- User clicks on "Filter by" (N) and is navigated to a menu with filter options → user clicks on a filter to apply (N), to which a menu of specific filters is shown → user clicks on a filter to apply (D) → user clicks "Apply Filters" (D)
- Total clicks = 1 + 1 * (# of filters user wants to apply) + 1 * (# of specific filters that user applies) + 1
- About 50% of clicks are for user-interface navigation; about 50% of clicks are for clerical data entry

UC-7: Recipe Search

- User clicks on "Search Recipe" tab (N) → user clicks on search bar (D) → user inputs keywords for recipe search (D) → user selects "Search" button (D)
- Total clicks = 1 + 1 + (keystrokes for keywords) + 1
- About 10% of clicks are for user-interface navigation; about 90% of clicks are for clerical data entry

UC-10: Grocery Store Locator

- User picks the recipe (N) → clicks "Buy Near Me" next to the desired missing ingredient (N) → user allows for location use (D) → user clicks on store (D)
- Total clicks = 1 + 1 + 1 + 1
- About 50% of clicks are for user-interface navigation; about 50% of clicks are for clerical data entry.

b. Project Size Estimation Based on Use Case Points

UCP = (UUCW + UAW) x TCF x ECF = (100 + 8) x 0.815 x 1 = **88.02**

- UUCW
  - UUCW = 4 x Simple + 5 x Average + 2 x Complex = 20 + 50 + 30 = 100

Group	Use Case	Description	Category	Weight
1	UC- 1: Registration	Allow users to create a login with a unique username and password that is stored in the database.	Simple	5
1	UC- 2: Primary Preference Selection	The user will be prompted to enter in their primary preferences, such as any dietary preferences or restrictions, as well as enter their location and a radius that they will be willing to go grocery shopping in.	Average	10
1	UC- 3: Login	The user can enter in their username and password, allowing the user to be logged into their own account with their own personalized app.	Simple	5
2	UC- 4: Ingredient Selection	The user will be able to select the ingredients they have by checking off boxes next to ingredients that are sorted into lists by food groups.	Simple	5
2	UC-5: Generated Recipes	Based on the ingredient selection, the user is presented with a generated list of recipes. These recipes are based on the dietary preferences in the user's stored profile settings and the ingredients they have selected. The recipes that require the least	Average	10

		number of extra missing ingredients appear first.		
2	UC-6: Filtering System for Generated Recipes	After seeing the generated recipes, users will have the option to further narrow down these recipes based on additional filters. These filters would include cuisine, meal preparation time, type of meal, and calories.	Average	10
3	UC-7: Recipe Search	The user will be able to search the entire recipe database, regardless if the user has the necessary ingredients to make that recipe. Once being able to access all of the recipes and search for any recipe they desire, they can still filter out their results by cuisine, meal prep, etc. However, these recipes will not be personalized to the user's ingredients, rather it is meant more for the user to explore the available recipes and see the selection at hand.	Complex	15
3	UC-8: Save Recipes	After searching for recipes, the user will be able to add recipes that he or she likes to a custom list, which the user can create. The user will have a page of the app dedicated to recipe lists, which he or she has saved and will allow for more lists to be created.	Average	10
4	UC-9: View/Change Profile	The user will be able to change their login information, edit their dietary preferences and food intolerances. The user will also be able to alter their address in order to change the list of available grocery stores.	Simple	5
4	UC-10: Grocery Store Locator	The user will be able to find stores nearby which should have the ingredients that the user is missing. This will allow the user to buy the necessary missing ingredients to make the recipe.	Complex	15
4	UC-11: Bug/Error Reporting	The user will be able to report any bug or error that occurs. This way, the developers can be aware of any problems that the user is facing and help approach them via a hotfix.	Average	10

- UAW
  - UAW= 1 x Simple + 2 x Average + 1 x Complex = 1 + 4 + 3 = 8

Actors	Roles	Complexity	Weight
Firebase	Will handle all user registration and authentication.	Simple	1
Spoonacular API	Will take in all the users ingredients and filters, and it will process their search.	Average	2
Google Places API	Will be used to search for nearby grocery stores upon request by the user.	Average	2
Bugify Issue Tracker	Will handle user bug reports.	Complex	3

- TCF
  - TCF= 0.6 + (TF/100)
  - TCF= 0.6 + (21.5/100) = 0.815

Group	Identifier	Description	Weight	Perceived Complexity	Calculated Factor
1	REQ-18	The application should be compatible with IOS and Android OS.	1	3	3
3	REQ-19	The application should be able to handle 500 daily users and can be scaled as needed.	1	3	3
2	REQ-20	The application should keep user information private by securely storing passwords and other sensitive information.	2	1	2
2	REQ-21	As a system, all stored and collected data should be secure.	2	3	6
1	REQ-22	The application layout should be aesthetically appealing and minimalistic with clutter free UI.	1	1	1
1,2,3,4	REQ-23	The application should operate at 60 Hz or 120 Hz for supported	2	1	2

		devices.			
4	REQ-24	The application should be easy and simple to download and navigate for nearly any person who has access to a smartphone and internet.	2	1	3
3	REQ-25	As a system, user requests and issues should be checked daily and addressed as soon as possible.	0.5	2	1
1	REQ-26	As a system, unit testing should be done after any changes to the system and before any version updates releases.	0.5	1	0.5
				Technical Factor Total :	21.5

6. Domain Analysis

a. Conceptual Model

i. Concept Definition

Responsibility	Type (D- Doing, K- Knowing, N- Neither)	Concept
R1: Display user registration or log in information	D	Interface
R2: Verify user's username and password entered	D	Authenticator
R3: Display distance and dietary preference selection list	D	Interface
R4: Display a list of all possible ingredients user can select	D	Interface
R5: Cross-reference database for available recipes	D	Controller
R6: Receive selected ingredients	C	Communicator
R7: Send a list of possible recipes using selected ingredients	C	Communicator
R8: Get keyword inputted in search bar to find recipes	D	Controller
R9: Display all possible recipes by keyword search	D	Interface
R10: Display filters for user to select	D	Interface
R11: Receive selected filters	C	Communicator
R12: Update list from selected filters	D	Controller
R13: Display updated list	D	Interface
R14: Store saved recipes for future use	K	Database
R15: Get user's current location	D	Controller
R16: Get nearby grocery stores with missing ingredients	D	Controller
R17: Display grocery stores and their location and hours	D	Interface
R18: Store user's login credentials	K	Database
R19: Store user's password hash	K	Database

R20: Store existing ingredients	K	Database

ii. Association Definitions

Concept Pair	Association Description	Association Name
Interface → Authenticator	Interface sends input data to Checker for verification	Check information
Interface → Controller	Interface sends user data, Controller reads input data	Read input
Controller → Interface	Controller gets requests from database and sends it to interface	Send result
Authenticator → Controller	Checker sends the validated information back to controller	Send validation message
Communicator → Interface	Communicator sends messages and interface displays it to user	Send message
Interface → Communicator	User sends in a message and communicator sends it to the system	Get/receive message
Controller ↔ Communicator	Controller gets requests from communicator and responds, communicator sends it back to system	Get a request and send an answer
Communicator ↔ Database	Communicator sends a request to database for stored info and sends response back	Send request, receive a response and send response

iii. Attribute Definitions

Responsibility	Attribute	Concept
R1: Display user registration or log in information	dispUserInfo	Interface
R2: Verify user's username and password entered	dataCheck	Authenticator
R3: Display distance and dietary preference selection list	dispPref	Interface
R4: Display a list of all possible ingredients user can select	dispIngr	Interface
R5: Cross-reference database for available recipes	getRecipes	Controller
R6: Receive selected ingredients	receiveIng	Communicator
R7: Send a list of possible recipes using selected ingredients to the user	sendList	Communicator
R8: Get keyword inputted in search bar to find recipes	getKeyword	Controller
R9: Display all possible recipes by keyword search	dispRecipes	Interface
R10: Display filters for user to select	dispFilters	Interface
R11: Receive selected filters	getFilters	Communicator
R12: Update list from selected filters	updateList	Controller
R13: Display updated list	dispList	Interface
R14: Store saved recipes for future use	saveRecipes	Database
R15: Get user's current location	getLoc	Controller
R16: Get nearby grocery stores with missing ingredients	getStores	Controller
R17: Display grocery stores and their location and hours	dispStores	Interface
R18: Store user's login credentials	saveUserCred	Database
R19: Store user's password hash	saveUserPass	Database
R20: Store user's existing ingredients	saveIngr	Database

iv. Traceability Matrix

The table below depicts how the use cases map to the domain concepts

Concept	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11
Interface	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Authenticator	✔		✔								
Controller					✔	✔	✔			✔	
Communicator		✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Database		✔	✔	✔				✔	✔		

Interface:
The interface maps to all of the Use cases, because this is primarily a user driven application. The user inputs all of the commands via the touch screen on the phone. For all of the use cases, the user is either entering in some kind of data, or navigating the screen. This is why the interface maps to each domain concept.

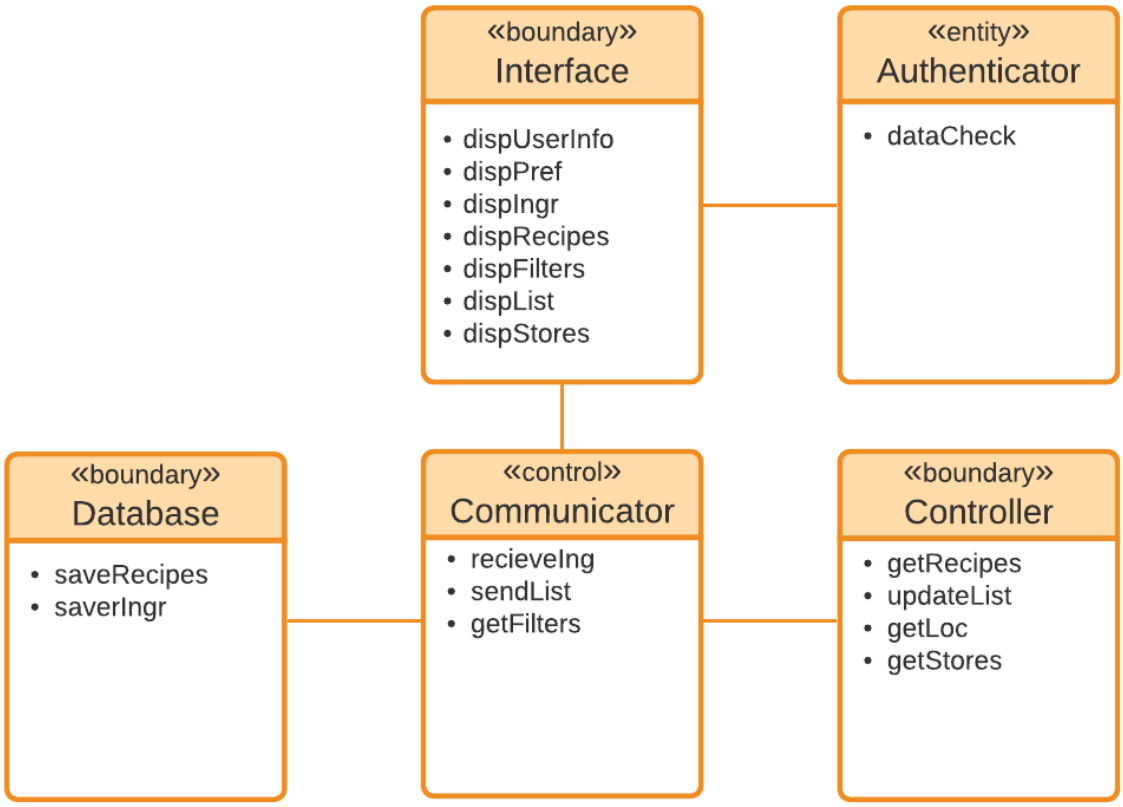
Authenticator:
The authenticator is only needed when some form of authentication is happening, mainly in UC-1 and UC-3. These 2 use cases are registration and login, respectively. This means that the username and password of the user are needed, and they need to be verified through the authenticator.

Controller:
The controller maps to UC-5 (Generated recipes), UC-6 (Filtering System for Generated Recipes), UC-7 (Recipe Search), and UC-10 (Grocery Store Locator). Essentially, the controller acts as the API request that we are sending from our application to the Spoonacular API and Google Maps API. The API calls occur in these specific use cases, which is why the Controller maps to those specific use cases.

Communicator:
The Communicator maps to almost all of the use cases except the first one. The communicator plays a key role in handling the specific requests that the user is trying to accomplish through the application. Because the Communicator handles all such requests, whether it be to access the API library through the Controller or the Database, it is present in almost every use case.

Database:
The database is meant for storing information, such as important login data or saved recipes. This domain concept maps to specific use cases that either store ingredients, store login information, and saved recipes. Because only specific use cases have information actually stored in the Firebase database, these are the use cases that map to the Database domain model.

Domain Concept Model



b. System Operation Contracts

Operation:	Registration
Use Case:	UC-1
Responsibilities:	• Use the database system to register a new user's username and password
Expectations:	• The new user is stored into the database system

Preconditions:	• The user has to be a first time user • The user doesn't have an account
Postconditions:	• The user can select their dietary preference • Username and password will be saved to database

Operation:	Primary Preference Selection
Use Case:	UC-2
Responsibilities:	• Use the database system to register a new user's dietary information, restrictions, and distance preference to their profile
Expectations:	• The user preferences are stored into the database system and to the user's profile
Preconditions:	• User is registered and can now operate the mobile app
Postconditions:	• User can now receive recipe suggestions based on preferences and ingredients

Operation:	Login
Use Case:	UC-3
Responsibilities:	• Firebase authenticates the user's login information
Expectations:	• The user logs into the system and is able to interact with the app
Preconditions:	• User can login with credentials used to sign up for mobile app
Postconditions:	• User has gained access back to their account • User can begin searching recipes and make changes to their account

Operation:	Ingredient Selection

Use Case:	UC-4
Responsibilities:	• The database system will store the user's selected ingredients so that the data can be used to view applicable recipes
Expectations:	• User will be able to view available recipes based off of any of the selected ingredients
Preconditions:	• User has full access and can operate their own account
Postconditions:	• User can enter any preference regarding meal they desire (e.g. meal type, cuisine of meal)

Operation:	Generated Recipes
Use Case:	UC-5
Responsibilities:	• The data from the database are transferred via the Spoonacular API which consolidates applicable recipes for the user to view.
Expectations:	• Users will be able to view a list of generated recipes based on their dietary preferences and selected ingredients
Preconditions:	• User has inputted preferences and dietary needs in order to generate recipes based on current ingredients
Postconditions:	• Recipes will be pulled that detail user's selected preferences and ingredients on hand

Operation:	Filtering System for Generated Recipes
Use Case:	UC-6
Responsibilities:	• Include more user preferences to better their user experience • To provide the user with more specific and detailed recipes

Expectations:	• User will receive a more specific set of recipes based on their filters
Preconditions:	• User has a generated list of recipes corresponding to their selections
Postconditions:	• User can narrow down results received even further by choosing type of meal, cuisine of meal, calorie count of meal, etc.

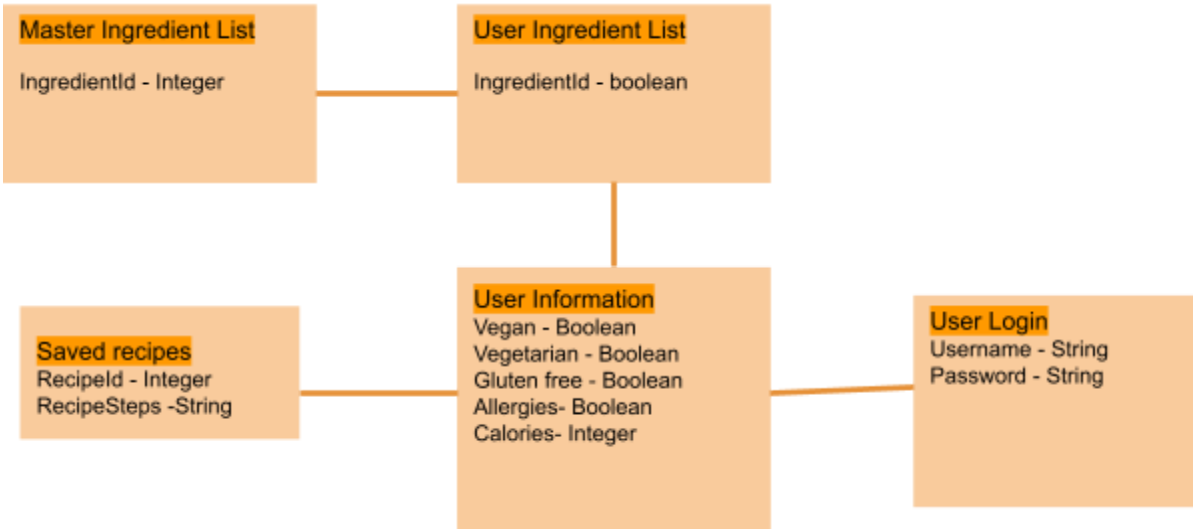
Operation:	Recipe Search
Use Case:	UC-7
Responsibilities:	• To find all recipes that are both within the user's dietary preferences and require only (if not, most of) the user's available ingredients
Expectations:	• User will be able to view recipes based on their search
Preconditions:	• User has selected dietary restrictions and preferences • User has inputted available ingredients
Postconditions:	• User is able to view all available recipes based off of their search

Operation:	Grocery Store Locator
Use Case:	UC-10
Responsibilities:	• To locate grocery stores closest to the user that have the missing ingredients available using mapping API.
Expectations:	• The app will display nearby grocery stores with items that the user will need for their chosen recipe
Preconditions:	• User has indicated current location • User has shared distance willing to travel to go grocery shopping • User has selected all current ingredients in possession
Postconditions:	• User has received name and address of grocery store

	nearest to their current location for missing ingredients of the recipe from the mobile app

c. Data Model and Persistent Data Storage

Database Schema Model



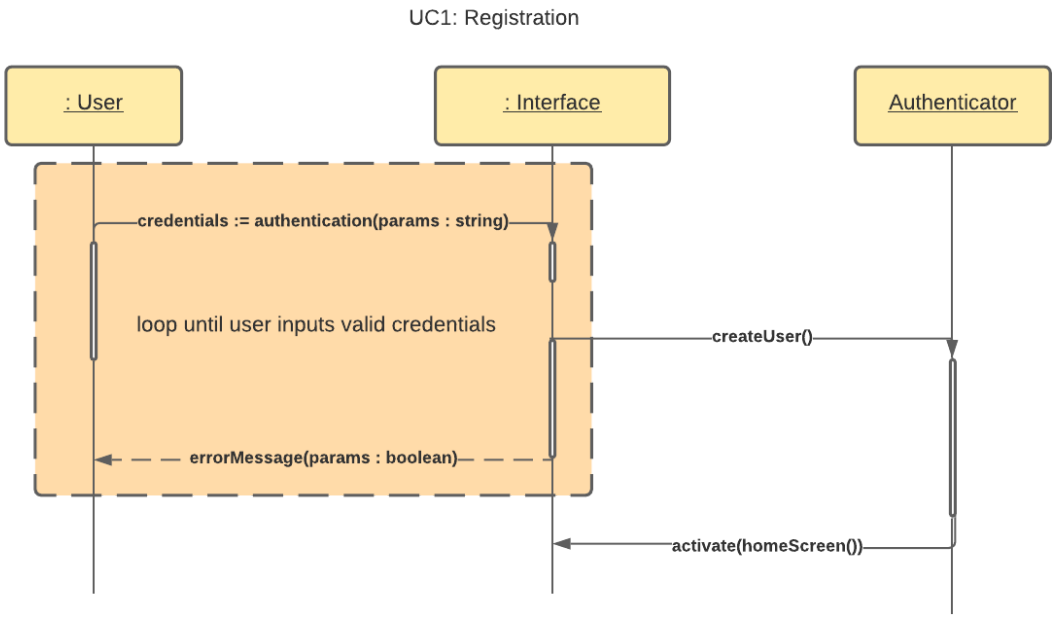
Our mobile application does need to have specific data stored within the database. It stores user information in the database, like dietary restrictions (vegan, vegetarian, gluten free, allergies, etc). It also stores the user's corresponding username and password which secure how their information is saved. In addition, it stores preferences the user might want to sort the recipes by, such as type of cuisine, meal, preparation time and serving size. The ingredients are stored in a master ingredient list as an integer, and if the user has an ingredient in the master ingredient list, it is stored in the user ingredient list as a boolean. This optimizes the storage space used by using the smallest possible data types. The resulting recipes are saved in order of most matched to the user's preferences and optimization of ingredients. Recipes are saved by "liking" them and these can be accessed at a later time via a separate drop-down menu. Users can save current ingredients in their household that include attributes such as name. In addition, the database would have to store the user's liked/saved recipes that they would like to return to the future. All of this information will be stored in a database running on our Firebase server.

d. Mathematical Model

Our mobile application does not use any mathematical model in order for it to operate.

7. Interaction Diagrams

UC1: Registration

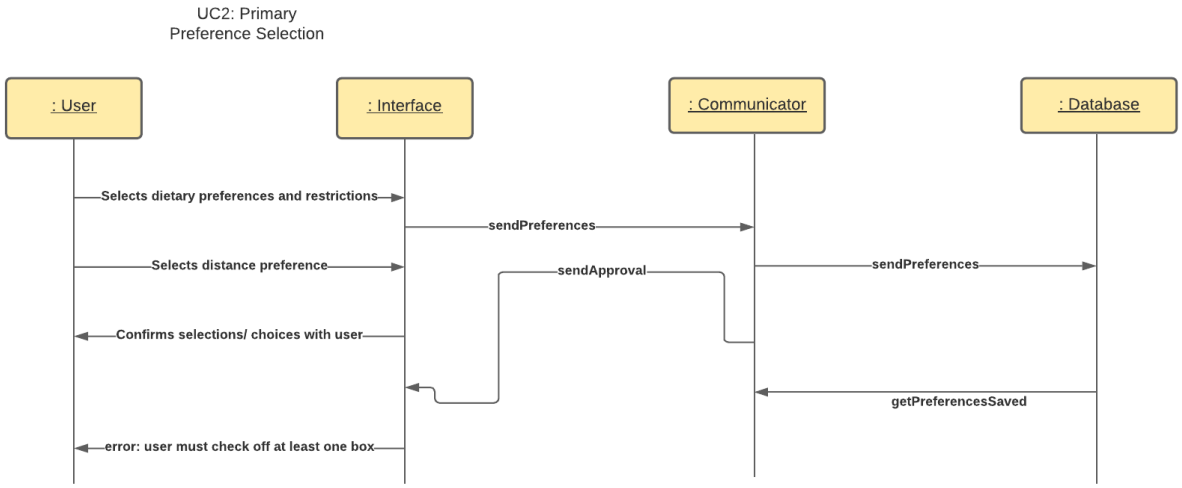


The above interaction diagram is for UC1: Registration. The user enters their desired username and password through the Interface. The Interface will then send the information to the Authenticator, which will check to see if the entered username is already existent. If it is, then it sends an error message back to the user, prompting them to enter in another username and password combination. If the Authenticator declares that the username and password combination are valid, then the home screen for the application is displayed and the user is able to use the other functionalities of the application.

Design Pattern Used: Publisher - Subscriber Pattern

The reason we chose to use the Published - Subscriber Pattern is because this use case is event-driven. This pattern separates the publisher and subscriber, where the Authenticator is the publisher. The authenticator notifies the subscriber (in this case the interface and the user) whether the registration has been completed or not. The Authenticator will simply send back the event, and the publisher will decide what to do depending on what the event shows. This way, the publisher and the subscriber are separated, which is known as loose coupling.

UC2: Primary Preference Selection

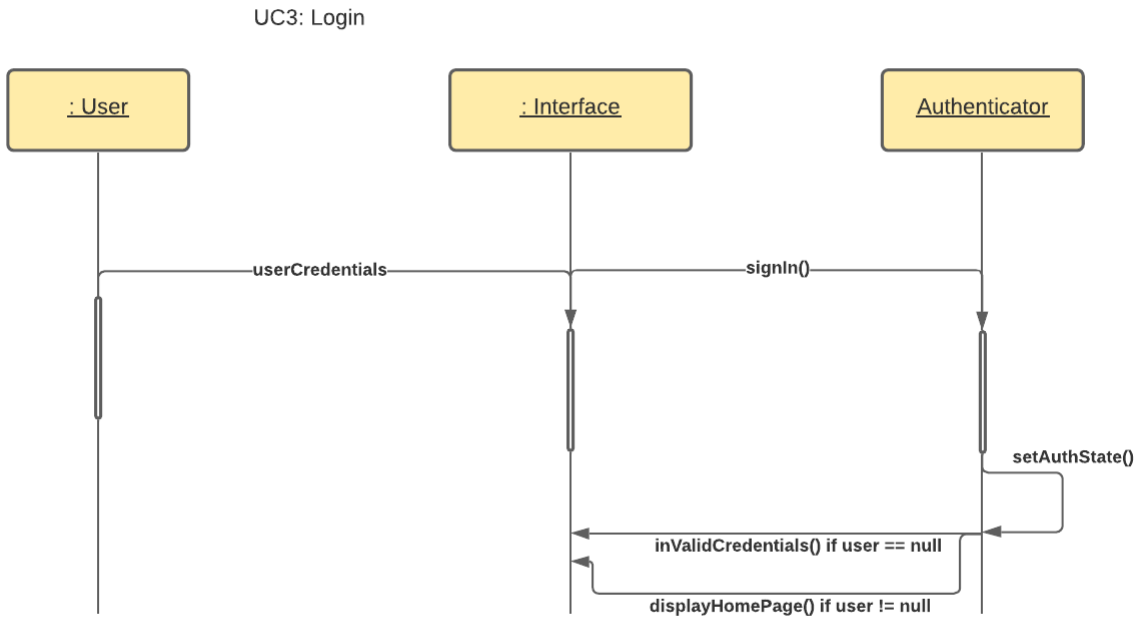


The above interaction diagram is for UC2: Primary Preference Selector. The user will select dietary preferences and restrictions, such as Vegetarian, Vegan, or additional allergies that they might have. The user will also select their distance preference for grocery stores. These will all be done via the Interface, which will then confirm the user's selections with them. The user's preferences/restrictions will be sent to the Communicator by the Interface, and will be stored there in the Database. The Communicator can also get saved preferences from the Database, and the Communicator can send the approval to the interface. If the user hasn't selected any boxes then an error message will be sent by the Interface to the User.

Design Pattern Used: Indirection Design Pattern

The Indirection Design Pattern is good for this use case because we have a Communicator. This pattern is best used when there is an intermediate object between other objects. This fits the Communicator role description, where the Communicator is between the Database and the Interface. This design also supports the low coupling design pattern, where each component doesn't completely affect another component, and can have more reuse between components.

UC3: Login

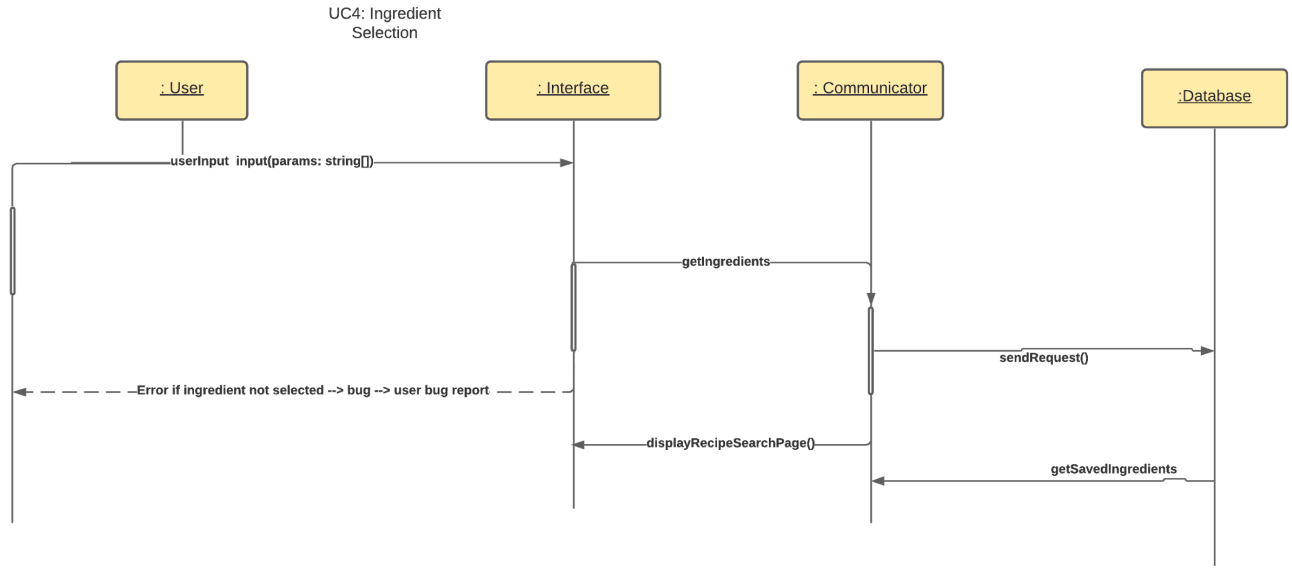


The above interaction diagram is for UC3: Log-in. The user will enter their username and password that was validated back when they first registered through UC1: Registration. The user will input in their credentials via the Interface. The Interface will send the credentials to the Authenticator when the User selects “Sign - In”, and will then set the authentication state depending on whether or not the credentials match those stored or not. If the login was successful then the home screen will display. If the login was not successful, then the user will be told to input their user information again.

Design Pattern Used: Publisher - Subscriber Pattern

The reason we chose this pattern is because it separates the publisher and subscriber. Much like UC1: Registration, the publisher (in this case the Authenticator), will set the state for the event that will be sent to the subscriber (Interface). This will separate the publisher and separator, where the publisher will send the events and the subscriber will determine what to do with that information.

UC4: Ingredient Selection

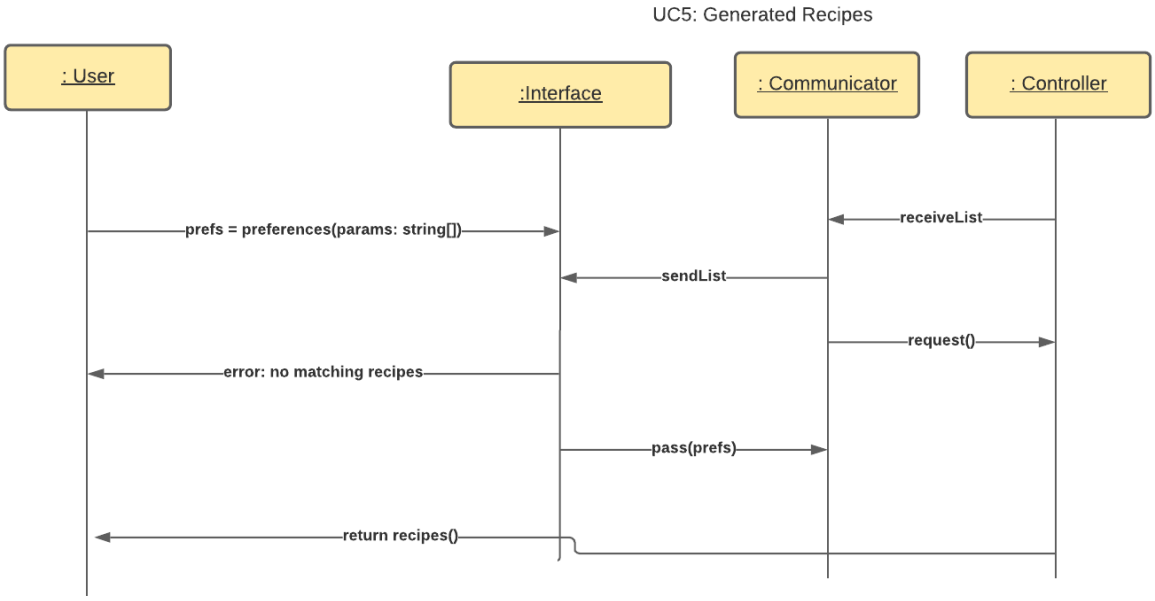


The above interaction diagram is for UC4: Ingredient Selection. The user will input their available ingredients which will output a list of recipes based on the imputed ingredients. The Communicator will send the request to the Database, who will send back the saved ingredients to the Communicator. The Communicator will display the recipe search page to the Interface. If no ingredients can be selected/are selected, then an error report is sent.

Design Pattern Used: Indirection Design Pattern

The Indirection Design Pattern is good for this use case because we have a Communicator. This pattern is best used when there is an intermediate object between other objects. This fits the Communicator role description, where the Communicator is between the Database and the Interface. This design also supports the low coupling design pattern, where each component doesn't completely affect another component, and can have more reuse between components.

UC5: Generate Recipes

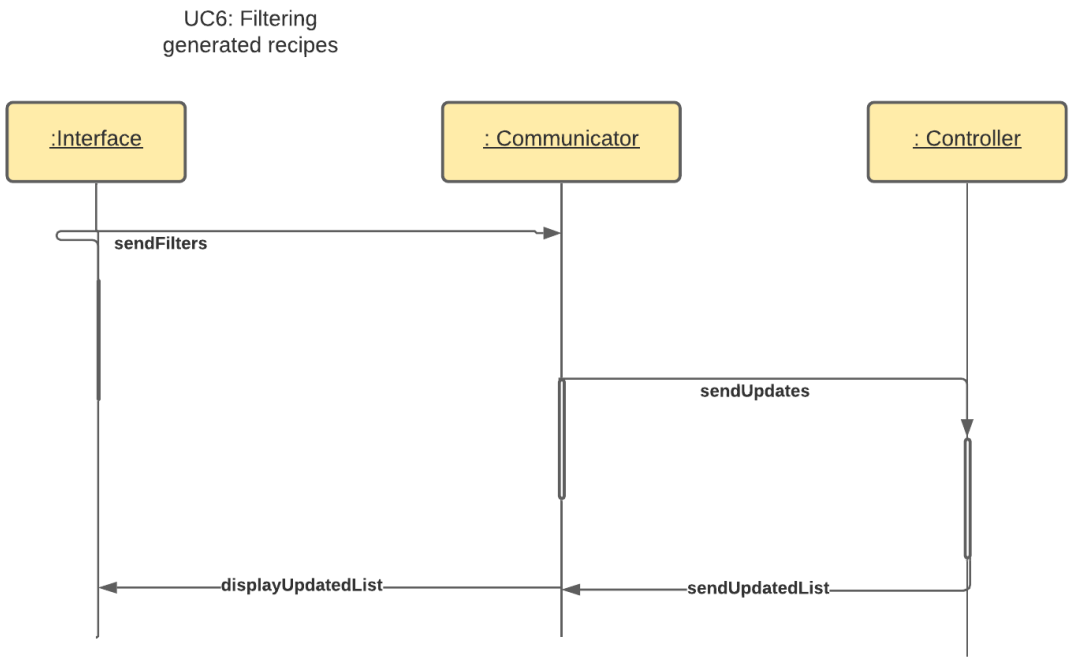


The above interaction diagram is for UC5: Generated Recipes. The user will be able to select and filter out recipes based on their dietary restrictions and current ingredients that they have already inputted into the application. The Communicator will send a request to the Controller, and the Controller will send back the recipes to the Communicator, which will send it to the Interface. If there are no matching recipes, an error message will be shown from the Interface to the User. The Controller will send the recipes to the Interface to be displayed.

Design Pattern Used: Command-based Design Pattern

The Command-based Design Pattern is useful for this design because the Communicator is sending requests to the Controller to access the ingredients so recipes can be generated. The Command-Based design pattern is the pattern where the request is sent as an object, and in this case, it is a JSON object. As such, the information is able to be modified and sent to the interface as it needs to.

UC6: Filtering Generated Recipes



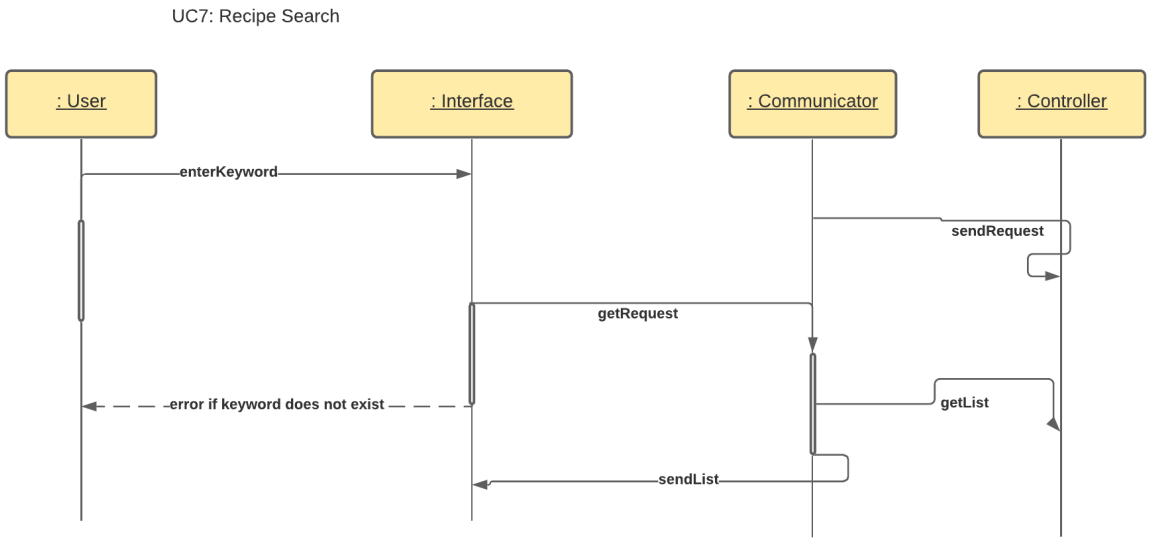
The above interaction diagram is for UC6: Filtering Generated Recipes. After a generated list is displayed via the Interface, the user can filter it out to make it narrower and more focused on what they want. The Communicator will send those updates to the Controller. The Controller will then update the list and send it back to the Communicator so it can send it to the Interface again and the new updated list can be displayed.

Design Pattern Used: Indirection and High Cohesion Design Pattern, Polymorphism

The reason we chose this design is because the Indirection Design pattern is meant to have an intermediate component which communicates with other components. For this use case, we have a communicator in between the interface and the Controller. The Controller is the middle component, which will handle all the requests and method passing from the Interface and the Controller.

The reason we chose polymorphism is because it is used to handle alternatives based on type. Polymorphism assigns responsibility for alternatives based on how each one varies by type (class). This will help us distinguish between filters for each recipe.

UC7: Recipe Search

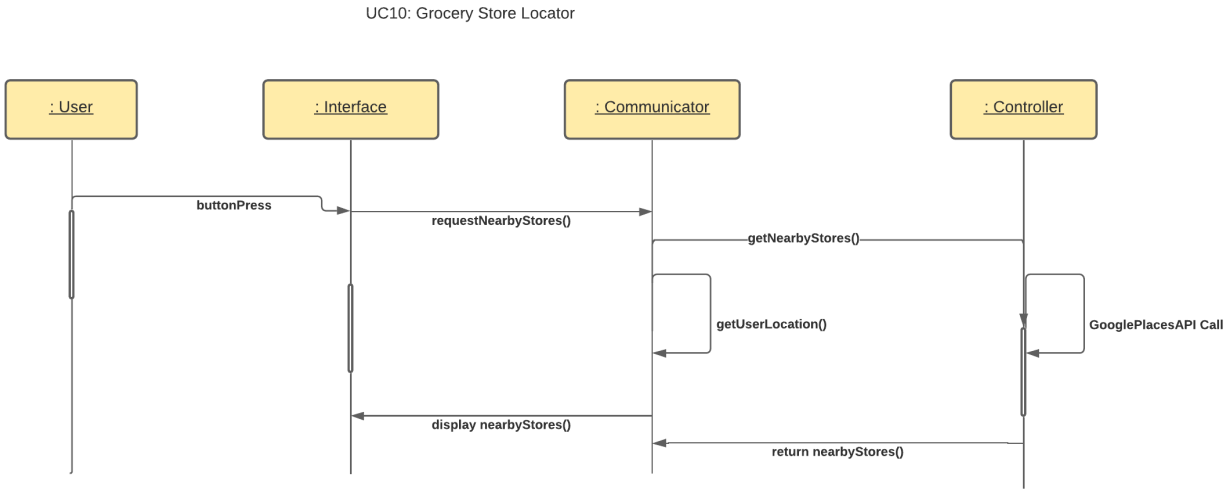


The above interaction diagram is for UC7: Recipe Search. The user will be able to input a keyword via the Interface to search for what specific recipe they want. The Communicator will receive those requests, and will then send back a list of recipes that match the keyword inputted. The Communicator will send the request to the Controller. The Interface will now display that list for the user to view and browse. If there are no recipes that match to the keyword entered, then an error message will be displayed.

Design Pattern Used: Command-based Design Pattern

The Command-based Design Pattern is useful for this design because the Communicator is sending requests to the Controller to access the ingredients so recipes can be generated. The Command-Based design pattern is the pattern where the request is sent as an object, and in this case, it is a JSON object. As such, the information is able to be modified and sent to the interface as it needs to.

UC10: Grocery Store Locator



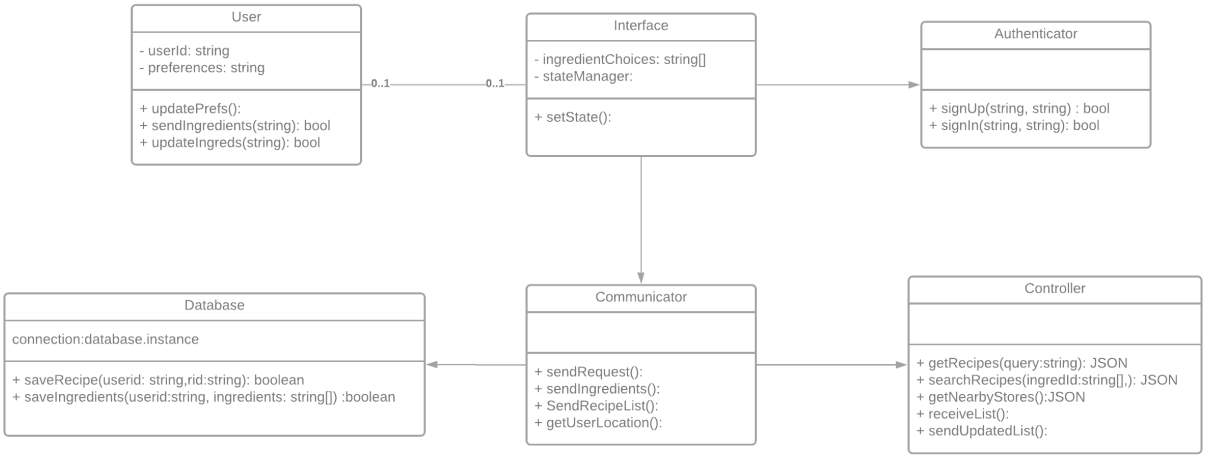
The above interaction is for UC10: Grocery Store Locator. Once the user is informed on what ingredients are missing from their desired recipe, they will interact with the Interface and seek a grocery store nearby with the missing ingredients by pressing a button. The Interface will then request nearby grocery stores to the Communicator, and the Communicator will then tell the Controller to get the nearby stores. The Communicator will get the user's current location through the location feature on their device, and the Controller will place an API call to the Google API. The Controller will return the nearby grocery stores to the Communicator, which will then send the information to the Interface to be displayed.

Design Pattern Used: High Cohesion Design Pattern

For this use case, we decided to use the High Cohesion Pattern. The High Cohesion design pattern is when an object doesn't take on too many responsibilities. We see that each object that we have created here is not handling all of the responsibility for the program. The Communicator is handling different responsibilities from the Controller, which is different from the Interface. In this way, the objects handle various different responsibilities so as not to make one thing do too much. This is the case for UC 10, and also for the other use cases, where the responsibility has been split up between the classes.

8. Class Diagram and Interface Specification

a. Class Diagram



b. Data Types and Operation Signatures

i. User:

The concept behind the User class is to allow an individual to enter and search for information regarding their account and recipes. The various attributes and functions that the User will perform are:

1. Attributes:
    a. **userId: string**
    This attribute will allow for the user to have an identity within the mobile app's system. The user will use this ID to sign in and operate the app. The userId will be the user's preferred email address.
    b. **preferences: string**
    This attribute will allow for the user's dietary preferences and restrictions to be selected, saved and passed to other parts of the system when searching for recipes.
2. Functions:
    a. **updatePrefs():**
    This function will allow for the user to change their dietary preferences and restrictions anytime they desire. If any preferences are changed, they will be saved and updated to the database.
    b. **sendIngredients(string): bool**
    This function will allow for current ingredients that are selected by the user to be saved into the database. This information will then be used when searching for recipes for the user, with consideration of the user's preferences.

c. **updateIngreds(string): bool**
This function will be used for when the user has to update their current ingredients within their possession. Allowing for the user's ingredients list to be updated will allow an easier convenience for them, as they don't have to keep entering in prior information with every use of the app.

ii. Interface:

The concept behind the Interface class is the software product that will allow the user to interact with the mobile application product. The Interface should be user-friendly and simple to operate in order to make the user experience enjoyable and useful. The operations that will be performed by the Interface are:

1. Attributes:
    a. **ingredientChoices: string[]**
    This attribute will allow for a list of ingredients that the user has selected to be generated and stored under the user's account. These ingredients will be used to help look for recipes for the user's meal.
    b. **stateManager:**
    Each part of the interface will have a state manager will be updated with the call of the setState function
2. Functions:
    a. **setState(): void**
    This method will be to update the state of all the elements of the app which are interactable.

iii. Controller:

The concept behind the Controller class is to manage the flow of data coming in from other classes within the system. The various functions that are performed by the Controller are:

1. Functions:
    a. **getRecipes(query:string): JSON**
    This function allows for a list of found recipes to be generated that are based around the user's information that was given when searching for a recipe to make.
    b. **searchRecipes(ingredid:string[]): JSON**
    This function allows for recipes to be searched for based on the user's current ingredients and preferences that are stored in the Database. The API then takes this information and looks for recipes with these specifics from the user.
    c. **getNearbyStores(): JSON**
    This function allows the mobile app to locate nearby grocery stores that have the user's missing ingredients available, as well as accounting for the distance the user is willing to travel.
    d. **receiveList():**

This function allows for a generated list of recipes to be sent back to the user. Based on the user's current ingredients and dietary preferences/ restrictions, an organized list of recipes with specific instructions will be presented to the user.

e. **sendUpdatedList():**
This function allows the the generated list of recipes that have been narrowed down even further to be sent back to the user. If the user decides to filter more specific requirements to their meal, those filters would then be applied and a newly generated recipe list will be presented to the user.

iv. Communicator:

The concept behind the Communicator class is that it allows the application to talk between different classes within the software. The Communicator will be able to receive and send data across the system, especially from the Interface and the Controller. The various functions that will be performed by the Communicator are:

1. Functions:
    a. **sendRequest():**
    This function allows for the system to communicate with other classes within the mobile app. Requests will usually be sent from the Communicator to either the Controller or Database in order to incite the system to perform a certain action.
    b. **SendRecipeList():**
    This function is used to allow the recipes that fit the user's criteria in a meal to be displayed on the Interface. It will allow the user to see all the options that the Database and API have found for them.
    c. **getUserLocation():**
    This function allows for the mobile app to get the current location of the user. The current location of the user will be requested by the system and stored on the device, as their location could be everchanging. The user's location will allow them to find stores that have their missing ingredients for their recipe chosen.

v. Authenticator:

The concept behind the Authenticator class is to ensure the safety and security of a user's account. The Authenticator will make sure that the account the user is logging in with is a valid account within the Database. The various functions that are performed by the Authenticator are:

1. Functions:
    a. **signUp(string, string): bool**
    This function is used to create a new user within the system. After signing up for the mobile app, the user's information is sent to the database to create a new user, as well as checks that the username isn't already taken.
    b. **signIn(string, string): bool**

This function is used to authenticate the user, which will allow the user to operate the application. The system checks whether the user is present within the database or not.

vi. Database:

The concept behind the Database class is to allow for the storage of all data that will be used to run and operate this mobile application. The Database allows for information to be kept in one secured location and be accessible anytime when utilizing the mobile application. The data that will be stored will be generated by the user and API that are used in our application. The operations that are performed by the Database are:

1. Attributes:
    a. **connection.database.instance**
    This attribute allows for the database to connect to the software behind the mobile application and lets for information to enter the database and be returned back to the interface.
2. Functions:
    a. **saveRecipe(userid; string,rid:string): boolean**
    This function will save any recipe that the user likes/favorites to their profile and allow the user to refer back to the recipe at a later time.
    b. **saveIngredients(userid:string, ingredients: string[]): boolean**
    This function will save any ingredients that the user has inputted into the system, allowing the user to add and remove their current ingredients with ease.

c. Traceability Matrix

The table below depicts how the classes discussed above have evolved from previous domain concepts and how they relate to the use cases.

	User	Interface	Communicator	Controller	Authenticator	Database
UC-1	Enters desired username and password through the Interface	Relays the user's information to the Authenticator			Verifies whether the entered username is already existent	

UC-2	Selects dietary preferences and restrictions and distance preferences for grocery stores through the Interface	Confirms user's selections and sends them to Communicator; receives approval from Communicator of saved preferences	Sends preferences to the Database; receives saved preferences from the Database and sends approval to the interface			Stores user's preferences
UC-3	Enters valid username and password through the Interface	Sends credentials to the Authenticator; shows state of login to user			Sets the authentication state; relays state back to Interface	
UC-4	Inputs available ingredients for recipe generation through the Interface	Sends user's inputs to the Communicator and displays recipes to user	Sends request to the Database; displays the recipe search page to the Interface			Stores ingredients and sends back saved ingredients to the Communicator
UC-5	Applies selection for recipes based dietary restrictions and current ingredients through the Interface	Sends filters to the Communicator and displays filtered recipes to user	Sends a request to the Controller; sends recipes back to the Interface	Receives request and returns list of recipes back to Communicator		
UC-6	Applies more specific filters to generated recipes through the Interface	Relays the user's filters to the Communicator and displays filtered list to user	Sends updated preferences to the Controller; receives and sends updated list back to the Interface	Updates the list of recipes and sends it back to the Communicator		
UC-7	Inputs a keyword via to search for a specific recipe through the Interface	Relays inputs to the Communicator and displays list of recipes to user	Receives requests from the Interface and sends them to Controller; receives a list of recipes that match the keyword inputted and sends it to the Interface	Handles request and sends back list to the Communicator		
UC-10	Presses a button to seek a grocery store through the Interface	Requests nearby grocery stores to the Communicator; displays nearby grocery stores to the user	Uses location of user's device and sends a request for a nearby grocery store to the Controller; receives request and sends it to the Interface	Uses an API to return location of nearby grocery stores to Communicator		

d. Design Patterns

The design patterns we used were Publisher-Subscriber, Indirection, Command-based, and Polymorphism. We feel as though these were sufficient. The Publisher - Subscriber Design pattern was used mainly for authentication use cases, because it effectively separated the Authenticator and the Interface, allowing each of them to work independently but still be able to communicate with each other in case of a log-in or registration request. Another design pattern that was commonly used was the Command-based design pattern. This essentially gave power to the Communicator to send data based on the input, or command. The communicator is essentially between the database and the Spoonacular API, where depending on the command, the action would change. The Communicator will always access something when it is needed, but this way it can identify whether it needs to go to the database or make an API request. Our application essentially functions completely off of the Communicator. This is why Indirection design pattern is perfect, because it identifies that there is a central, intermediate object between the other key objects in the application. Additionally, we chose Polymorphism because we were able to use the Communicator to differentiate between various filters for UC-6: Filtering Recipes. It was able to make API calls to the Spoonacular API, and change the data from there, showing that the Communicator can fit the Polymorphism pattern of having responsibility based on different alternatives (filters).

e. Object Constraint Language (OCL)

As the application was realized in Flutter, we do not have standalone classes, but rather stateless and state-ful widgets that implement classes. Below is the OCL description of each widget:

auth

class AuthenticationView
Invariant: _usernameField, _emailField, _passwordField, _passwordCheckField
Pre-conditional:
Post-conditional: AuthenticationViewState();

class FirebaseAuth
Invariant: username, email password
Pre-conditional:
Post-conditional: _firebaseAuth.authStateChanges();

grocery_search

class Geometry
Invariant: location
Pre-conditional: this.location == True;
Post-conditional: parsedJson[‘location’];

class Location
Invariant: lat, lng
Pre-conditional: N/A
Post-conditional: parsedJson[‘lat’], parsedJson[‘lng’]

class Place
Invariant: name, rating, userRatingCount, vicinity, geometry
Pre-conditional: N/A
Post-conditional: parsedJson[‘geometry’]

class search
Invariant: <Position>(context)
Pre-conditional: GeoLocatorService()
Post-conditional: places[gometry, lng, lon]

class geolocatorService
Invariant: Future<Position>
Pre-conditional: Golocator()
Post-conditional: startLat, startLong, endLat, endLong

class PlacesService
Invariant: N/A
Pre-conditional: Map<parameters>
Post-conditional: Uri.https(baseUrl, parameters)

ingredient_selection

class FirestoreService
Invariant: FirebaseFirestore.instance
Pre-conditional: Map<String, dynamic> userData
Post-conditional: recipeIds.add(int.parse(doc.id)));

class IngredientsView
Invariant: ChangeNotifierProvider<ExpandedDropDown>(IngredientDropDown)>

Pre-conditional: N/A
Post-conditional: notifyListeners();

Profile

class IntolerancesWidget
Invariant: Provider.of<UserData>(context)
Pre-conditional: inteoleranceSelection
Post-conditional: N/A

class ProfileView
Invariant: Provider.of<UserData>(context)
Pre-conditional: <AuthenticationService> == True
Post-conditional: ProfileUpdater(userdata)

class ProfileUpdater
Invariant: userdata.userIntolerances =! null
Pre-conditional: GestureDetector(diets)
Post-conditional: N/A

class UserData
Invariant: Map <> intolerances
Pre-conditional: uid, username, diet, intolerances
Post-conditional: update toMap()

Recipe_search

Class RecipeView
Invariant: _recipe
Pre-conditional: recipes != null
Post-conditional: StepsDisplay(_recipe.steps)

Class RecipeListWidget
Invariant: recipes
Pre-conditional: recipes != null
Post-conditional: RecipePreview(recipe: recipes[index])

Class Recipe
Invariant: id, readyInMinutes, title, summary, imgLink, cuisines, dishTypes, diets, steps, ingredients
Pre-conditional: ingredients != null, steps != null
Post-conditional: N/A

Class recipe_preview
Invariant: recipe
Pre-conditional: recipe != null
Post-conditional: child: RecipeView(recipe)

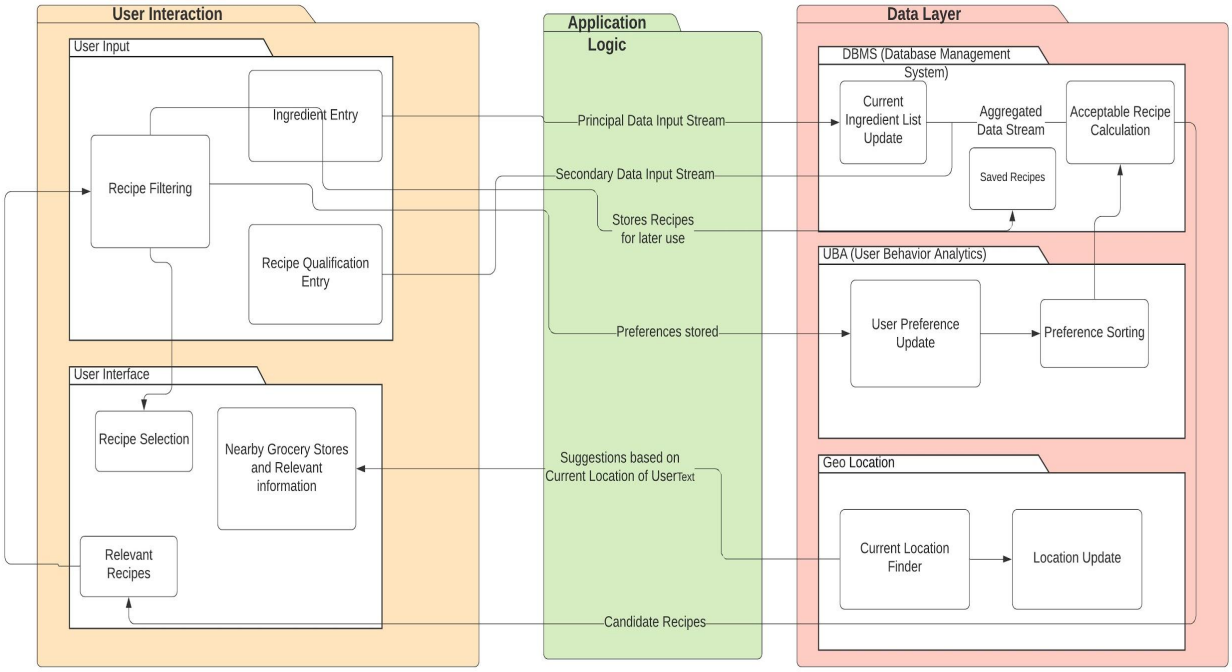
Class SearchView
Invariant: none
Pre-conditional: N/A
Post-conditional: N/A

saved_recipes

Class SavedRecipesView
Invariant: {Key key}
Pre-conditional: N/A
Post-conditional: RecipeListWidget(recipes: savedRecipes)

9. System Architecture and System Design

a. <u>Identifying Subsystems</u>



Pictured above is the UML diagram to aid in subsystem identification

The UML diagram provided in this section demonstrates the procedural behavior of users and associated backend procedure systems. The diagram is divided into two principal modules: the User Interaction and the Data Layer modules. The purpose of this division is to separate all user-generated activity from the procedurally generated machine activity of the application. Each module is further subdivided into various subsystems. Below is a breakdown of the principal objectives of each module and subsystem:

- User Interaction Module - Everything remotely related to user actions. Subdivided into the following categories:
    - User Input - The necessary and sufficient information that the user needs to input for the application to render a successful decision.

    - Ingredient Entry - Fulfilling UC-4, the Ingredient Entry module allows the user to enter the ingredients that are available to them. This information is then passed as the Principal Data Input Stream to the subsequent decision making processes.
    - Recipe Qualification Entry - Partially fulfilling UC-2, the Recipe Qualification module allows the user to select all attributes that "qualify" a recipe to be suitable for him or her. This includes dietary restrictions, cuisine preferences, and macronutrient/caloric requirements. This information is passed as the Secondary Data Input Stream to the decision making protocols. It is secondary since the user can choose to omit entering any information in this section, thus it is not critical.
    - Recipe Filtering - In the penultimate step of the user experience, the user is given the option to choose from the recipes provided by the application. The process of selecting said recipes by the application is not conducted in this module.
- User Interface - This module hosts everything that the user sees on his or her display but does not necessarily interact algorithmically.
    - Recipe Selection - A display of the selected recipes that the user wished to cook. This is the final step of the user experience.
    - Relevant Recipes - A collection of all the recipes that have been provided by the application for the user to select from. This module is chronologically between the Recipe Selection and Recipe Filtering.
    - Nearby grocery stores and Relevant Information - A list of grocery stores in the area provided to the user by the application. This is where the capabilities of the application come to an end, and the user may choose to go to the suggested grocery store and purchase the necessary items.

- Data Layer Module - This module hosts all of the back-end that the user does not directly interact with. It hosts tools for managing the various databases it interacts with, as well as the engine that is responsible for suggesting recipes to the user based on past history.

- DBMS (Database Management System) - This serves as the engine for handling all API calls, as well as accessing the local database of recipes that the Selection Algorithm has chosen for the user.
    - Current Ingredient List Update - Maintains a running list of ingredients available at the users disposal. The running list may account for non-perishables that the user has previously input, or may be updated altogether.
    - Acceptable Recipe Calculation - After taking into account the ingredients that the user has on hand, as well as past preferences from UBA, the console cross-references the database to determine which recipes the user is able to cook.
    - Saved Recipes - A running log of the recipes that the user has favorited in the past. The Saved Recipes log is saved locally on the user's machine, and is factored into the Acceptable Recipe Calculation.

- UBA (User Behavior Analytics) - This module contains all the decision-making process behind the suggestion generation for preferred recipes.
    - User Preference Update - Using preferences that users put towards the process of filtering recipes (whether that be cuisine, type of meal, saved recipes etc.), these particular preferences will be stored to the user's profile for future use.
    - Preference Sorting - Using information stored from User Preference Update, those user specific preferences will be used towards helping filter out relevant recipes that they may search for in the future.
- Geolocation - This module is responsible for finding the current location of the user while using ChefPal.
    - Current Location Finder - Based on where the user is currently located while using the application, nearby grocery stores will be recommended to the user if they need to get additional ingredients
    - Location Update - Last used location is stored in case the next time the user needs nearby groceries, they are accessing from the same place (e.g. their home)

b. Architecture Styles

Our application will be a combination of the Layered Architectural style, and the Client-Server architectural style. In the Layered Architecture we will have 3 layers: user interface, application logic and database. In the user interface layer, the user will directly communicate with the application, from a front-end perspective. The application logic layer is incredibly important, because this is the layer that directly interacts with the database layer and is in charge of what the application will do next depending on the specific user input. The application logic handles the API requests that will connect between the user and the data stored in the AWS server. The database layer is where all the user information is stored, as well as the ingredients, recipes, and all the information about those recipes is stored as well. This style is ideal for our application because it clearly separates the different responsibilities of each layer, so that layer only has to focus on one part. This does not mean, however, that each part is independent of the other layers. It simply means that each layer has its own priorities to handle, and working together with the other layers can develop a fully functional application.

Our application also uses a Client-Server style architecture. In this case, the part of the architecture that is always waiting for a command is our database. The recipes and information are always ready to be obtained, and are simply waiting to be accessed. The client, or the user, will be the one performing the actions in order to tell the server that it is looking for some particular information. In our application, the API is the connector from the user to the server. This style works really well with our application because all of the information is stored in a database, while multiple clients all obtain information from that one server.

c. Mapping Subsystems to Hardware

In this application, the user will be able to access and interact with the software using a cell phone or tablet. This device will handle all the user inputs that will require information to be retrieved from the data layer. Any input that will require information to be retrieved, or even navigation of the application will be done through the touch screen on the mobile device or tablet.

There will also be a backend server, which will contain the databases for ingredients, recipes, as well as saved recipes. This will also hold any additional backend code that we need for the app to operate. The server will communicate with the client through different API requests, depending on how the user interacts with the application. Multiple users will be able access the database on various different devices, although it will only be one user per device.

There will also be a connector, which will send the API requests to get the information from the data layer to the user. The connector facilitates API requests, so that the user never needs to directly interact with the database, and allows for processing of contextual information.

d. Connectors and Network Protocols

Our application will run on a user's mobile devices and will communicate with our own database which will store the user's preferences, saved recipes, and other information. This database will be stored on Firebase. This data will need to be accessed using an API provided by Firebase. In order to perform recipe searches and get information about nearby grocery stores we would be using existing REST APIs to get the necessary information from the existing data pools. To get the recipe data we will use the Spoonacular REST API and for the grocery stores we will use the Google Places API. These APIs would follow the same control flow as the API we would be using for our own database, except we do not have to implement the connection from the API to the database. The user would perform an action on the app such as a recipe search. The app would then make HTTP requests to the API. The API would then fetch the data from the database and pass it back to the API. Then the API will finally pass the given information back to the user.

e. Global Control Flow

Our system is event-driven. In this sense, there is no “linear” set of instructions that the user has to go through each time they use the application. The user may be able to input ingredients, or immediately search the general database for recipes. The system will wait for an instruction to come from the user, and depending on the user’s input, the system will respond accordingly. In our case, our participating actors, or the Spoonacular and Google Places APIs, both use HTTP requests with the response data as either JSON or XML. This will allow the system to wait until any instructions are received from the user, and then send out the relevant information based on which instruction to perform. Our system is an event-response system because the app responds to events initiated by the user. For example, the user will initiate an event and the application will process that event request and serve the user the data or service they require. This will be used to serve the user the data or service they require.

Since our application is event driven there is no concern for realtime.The application can be used at any time of day where there is a stable internet connection to connect with the server. There is no real-time constraint, and the system can be executed at any time.

f. Hardware Requirements

Our application should be compatible with any iOS device running iOS 11 or above and with any android device running android 5.0 or above. Due to these OS requirements, the device should have a color display with size restriction of a 4 inch display (iphone 5s) or above for smartphones; for tablets, there should be a 7.9 inch display or above. Our application also restricts the user to operate only in portrait mode. The size of our application is still under review so as default we would require the user to have a minimum of 500 mb of free storage space on their device. We would also require a device with GPS service for our grocery store locator function. We will update the hardware requirements as the design of the software progresses.

10. Algorithms and Data Structures

a. Algorithms

Our mobile application does not use any algorithms in order for it to operate.

However, ChefPal does offer the opportunity to implement algorithms in the future. The idea for ChefPal is to make searching for, and finding recipes as easy as possible. Taking this vision into account, we want ChefPal to eventually be able to recommend certain recipes for you, based upon the types of recipes you have viewed/generated in the past. It is currently not implemented because there is not much user data available, so there is no way to actually recommend recipes, if there are no users to recommend for. This is definitely something that was put into consideration when making the application, and we believe that it has tremendous potential in future versions. The algorithm, which would implement machine learning, would track which types of recipes the user generates/selects the most based on frequency, type of food, type of cuisine, etc. From this information, ChefPal will recommend recipes for the user to try.

b. Data Structures

Many of the data structures involved in this application are predefined classes within flutter. They are described below:

1) Trees

Flutter operates off of widgets, and these widgets are organized in a tree data structure. Each widget has depths, where interacting with a certain widget can cause you to go deeper within the tree. The tree is meant to organize the widgets in a way that the user can interact with them in an organized manner.

2) Lists for navigation pages

The navigation pages manage the user experience and their redirection to a number of other pages. As such, they contain an array of references to other page links. Navigation pages offer an easy to implement solution for redirecting users between screens with minimal computation overhead. Navigation pages are a predefined class.

3) Lists, dictionaries for JSON

The JSON information that we are retrieving through API requests comes in the form of lists and dictionaries. We have to parse through the JSON data in order to get the information we need. Lists and dictionaries make the information easy to navigate, and is a crucial part to our application.

c. Concurrency

Our application is being programmed with Dart which is only single threaded. Dart, which is the language used in Flutter, operates using a widget tree, where the code is processed in order from top to bottom. No threads are created when using Flutter, therefore our program is single threaded.

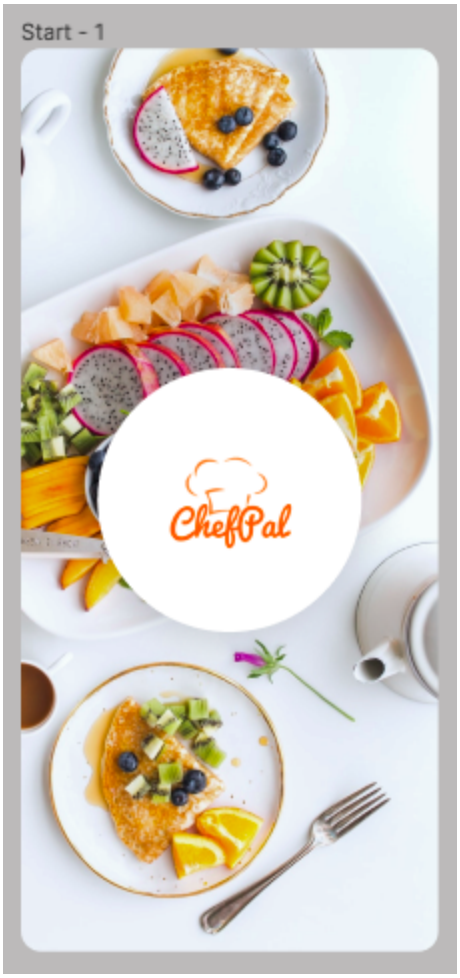
11. User Interface Design and Implementation

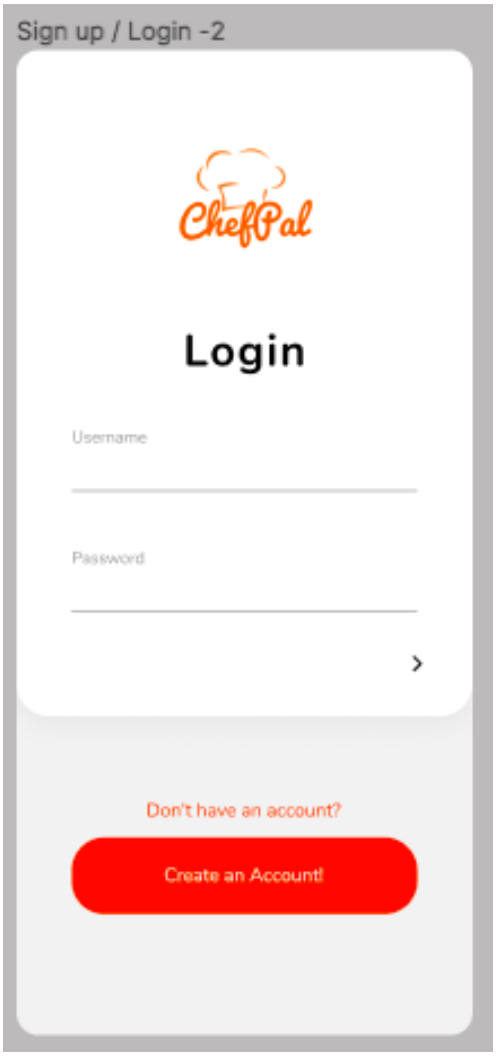
For the UI design, we used Figma to help illustrate the application and the steps that a typical user would go through to access and find recipes. This allowed us to get a general idea of how we wanted our mobile app to look and feel. In addition, it allowed for every member to see and give feedback on their likes and dislikes of the app design. With this user interface design, we were able to offer a preview of what our users should be expecting to see and be using from this app. The way that the user interface was initially designed, it was meant to be user friendly and allow the user to use minimum effort. The preliminary design of ChefPal can be seen in Full Report #1.

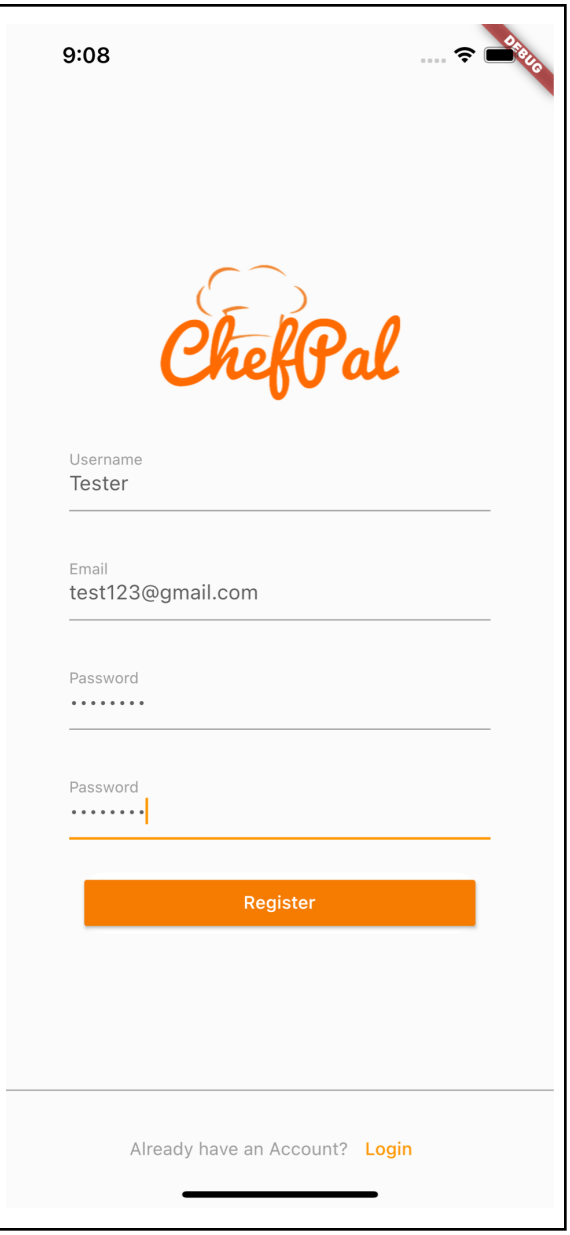
After having the UI design done and a general guidelines of what we expect the mobile application to look like, implementation is the next step. Implementing the initial mock-up drawing for the User Interface will be done with the use of Flutter for frontend development. Within Flutter, we will be using Dart, which is the coding language of Flutter, to implement our design. The use of Flutter was decided upon as it is easy to use, compatible with iOS and Android, which will allow us to make the User Interface user friendly. With such simple mock-up drawings and concept design that is easy to follow, having to implement the drawings into an interface will not be as complex as we had originally expected. Using inspiration from the preliminary UI design, the final UI design for ChefPal can be seen below.

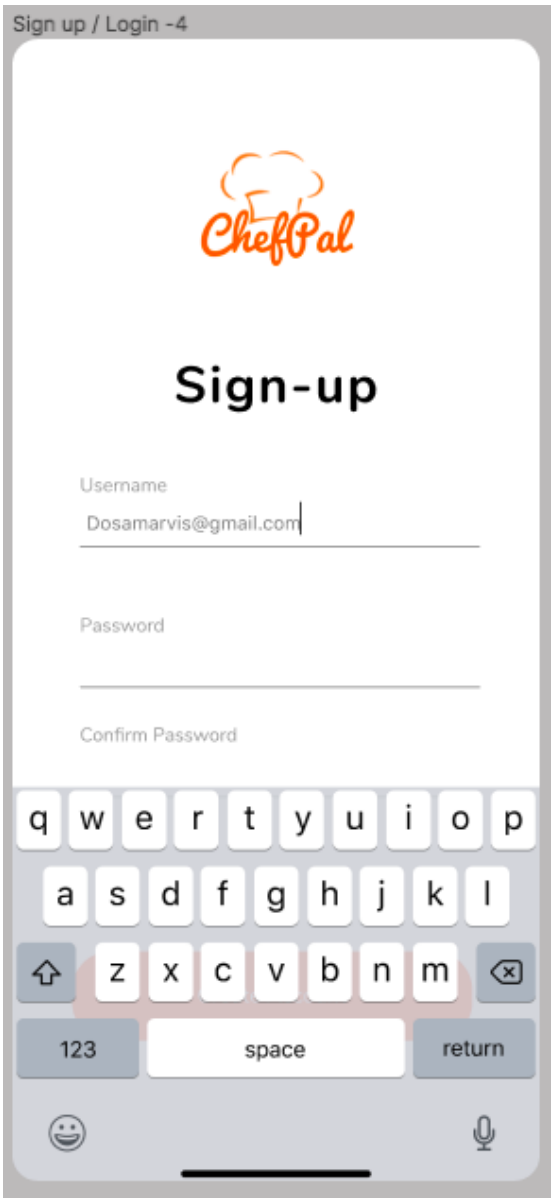
a. Final User Interface Design

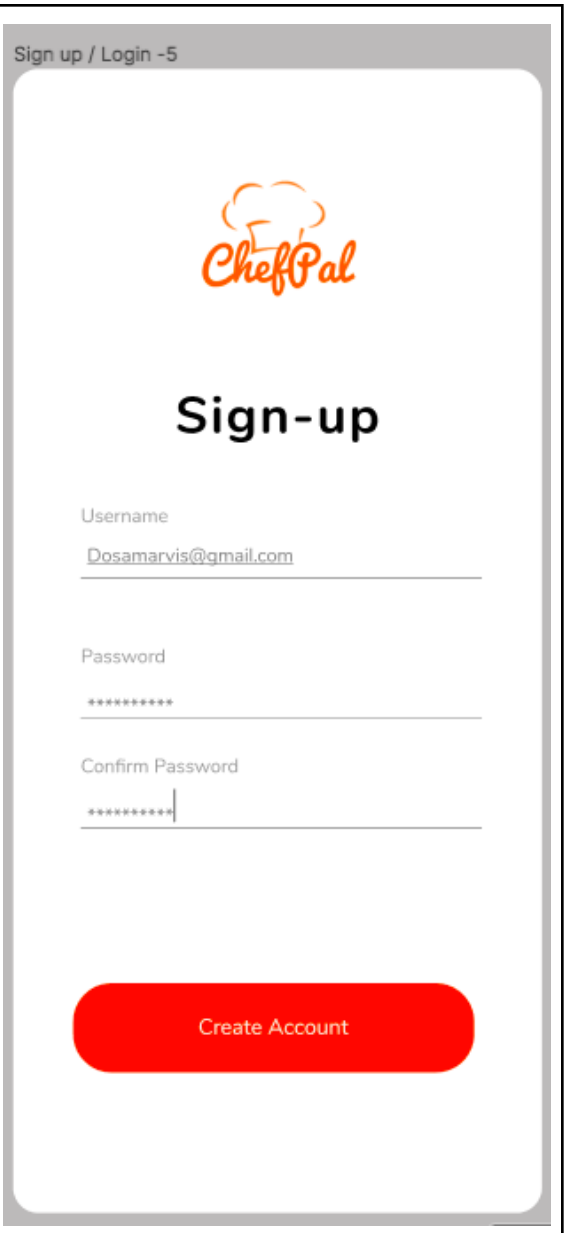
UC-1: Registration

1) User opens the application and sees the welcome screen while the application loads.	

2) User clicks “Create an Account!” and is redirected to the sign-up screen.	

3) User clicks on each textbox to input data.	

4) User inputs their email and password associated with the account.	

5) User clicks “Create Account” once all required fields are entered.	

UC-2: Primary Preference Selection

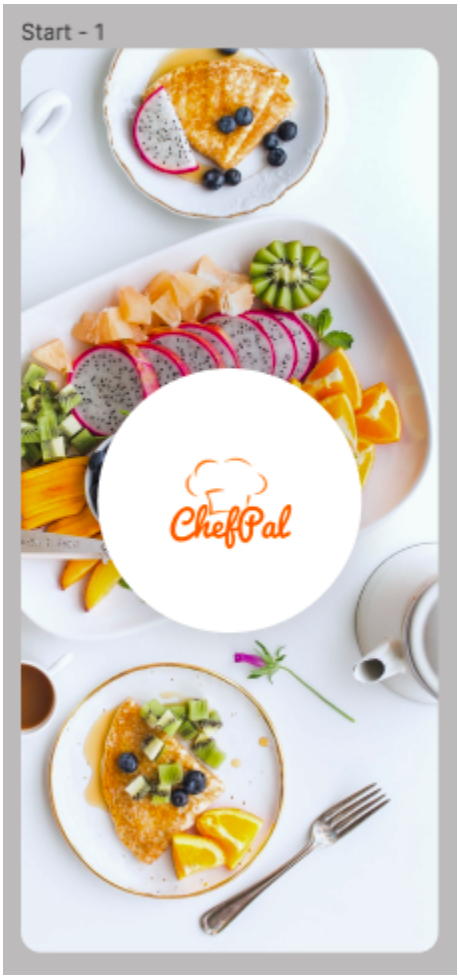
1) After creating an account, a checklist of types of dietary restrictions is displayed. The user can check off as many or as little as he/she desires.	

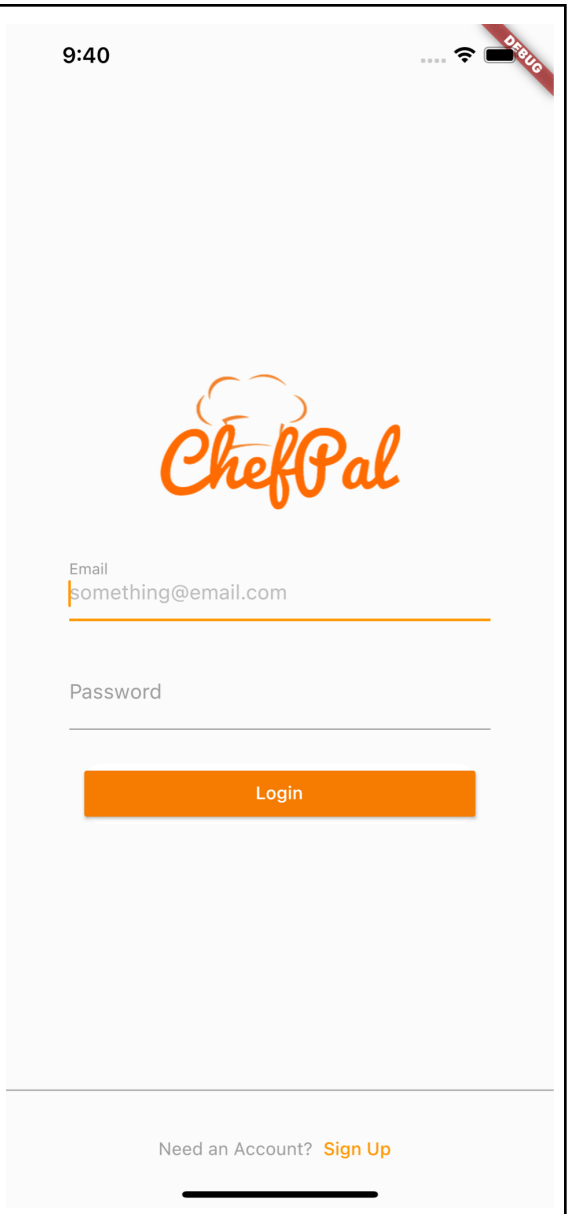
2) User clicks “Submit”.	

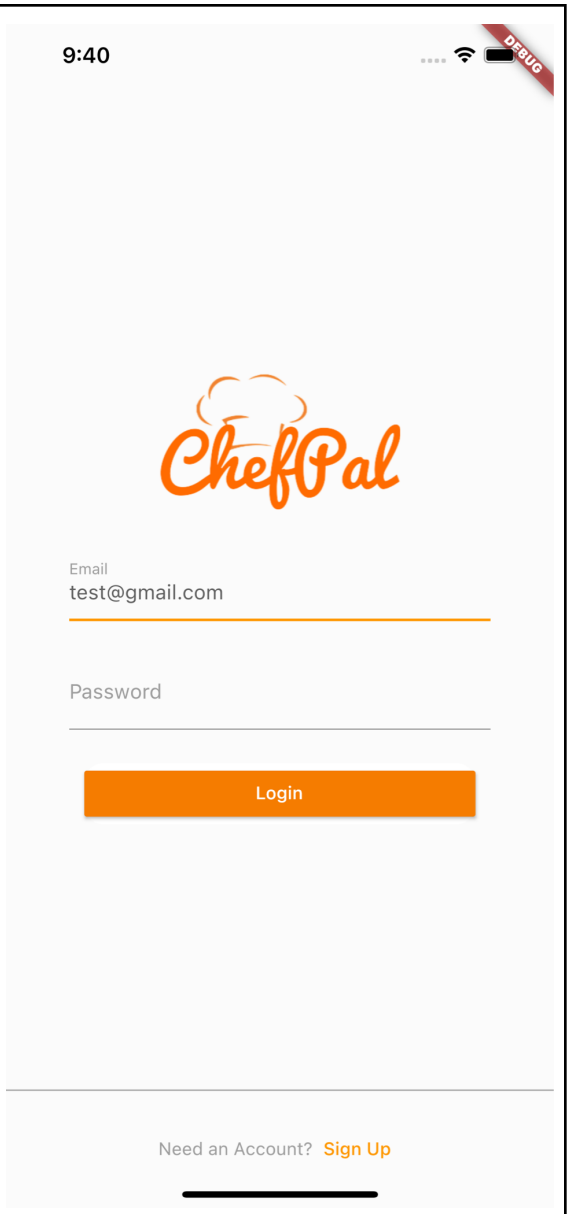
3) User is redirected to a distance radi to choose from. This will be the preferred range of distance between the user and the presented grocery stores for missing ingredients they may have.	

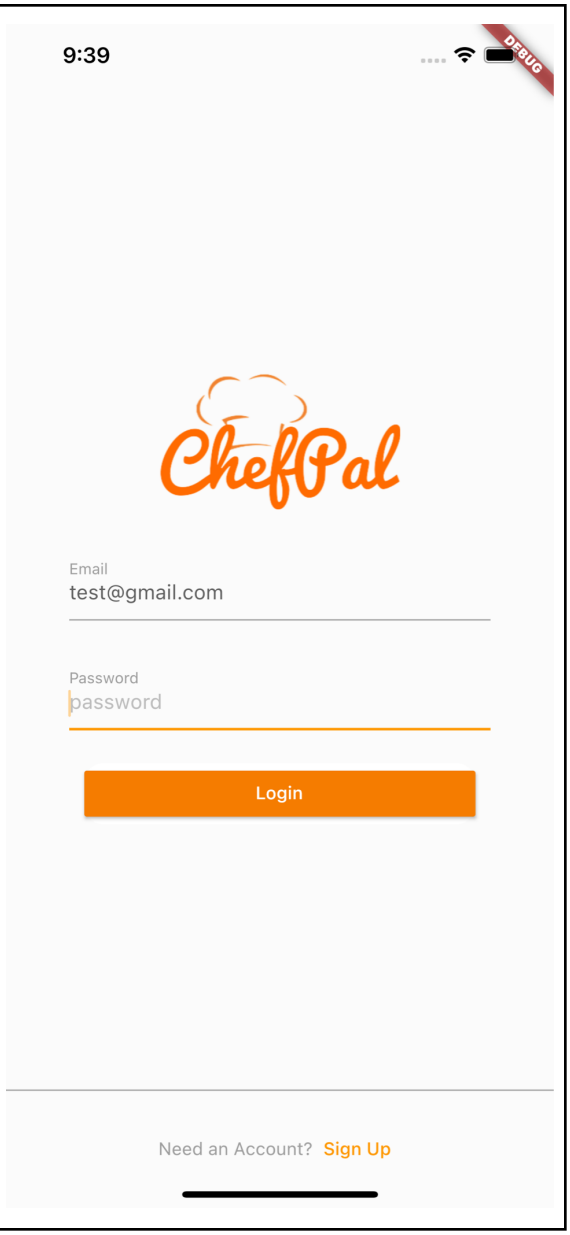
4) User clicks “Submit” on their preferred radi.	

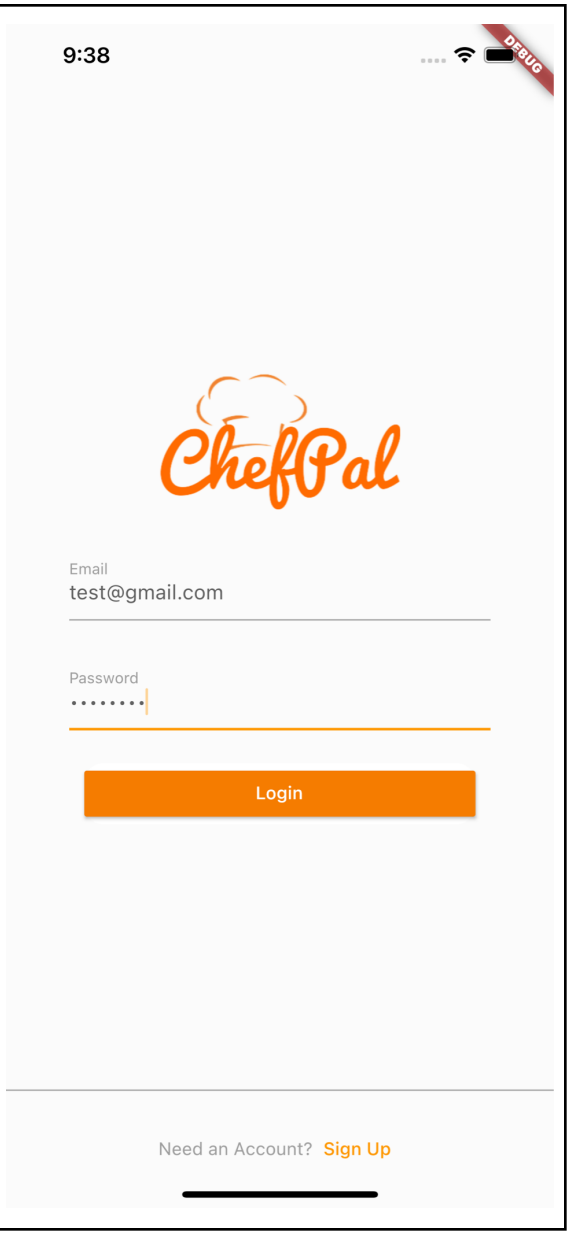
UC-3: Login

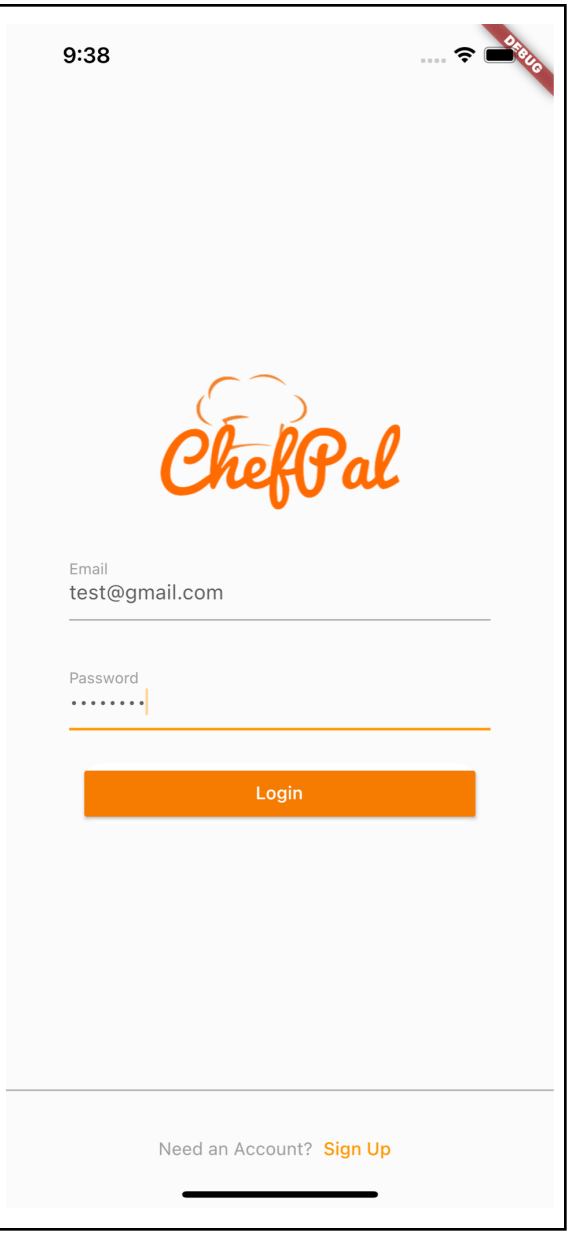
1) User clicks to open the application.	

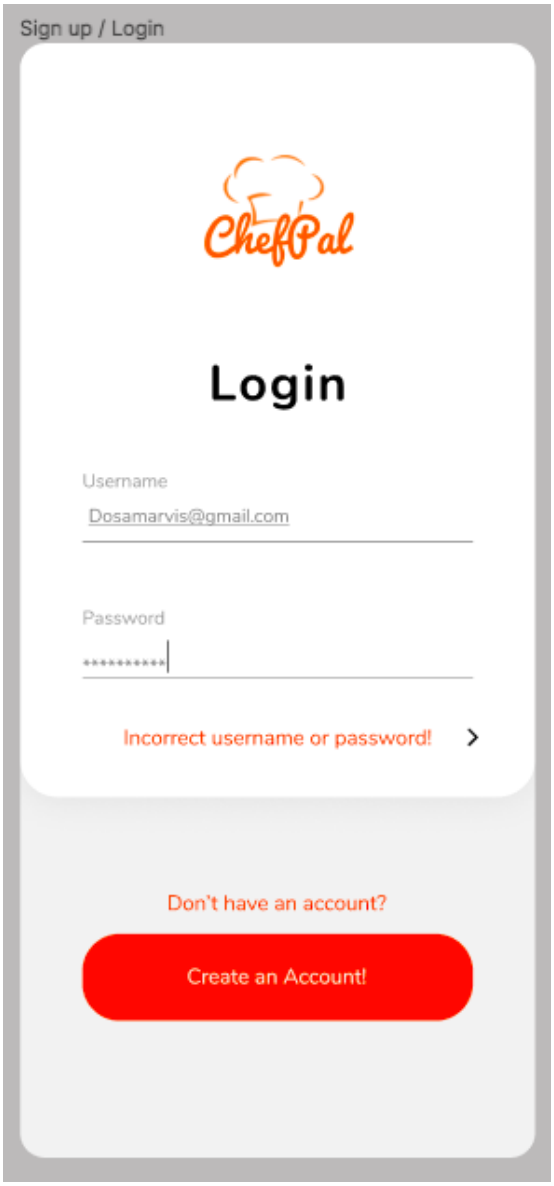
2) User clicks username textbox.	

3) User enters username.	

4) User clicks password textbox.	

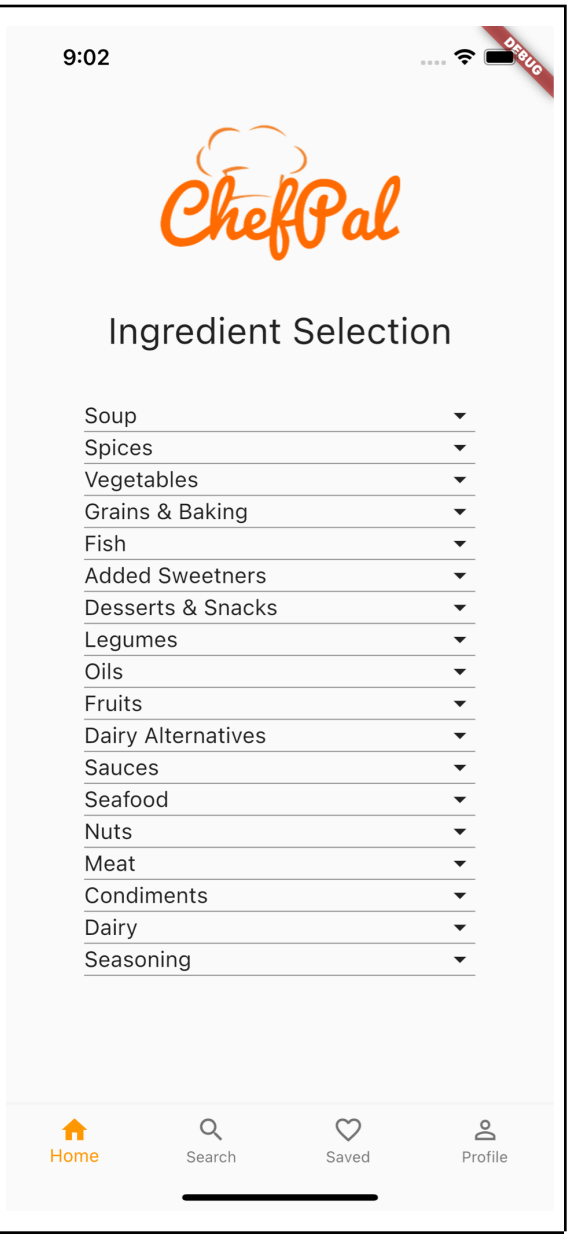
5) User enters password.	

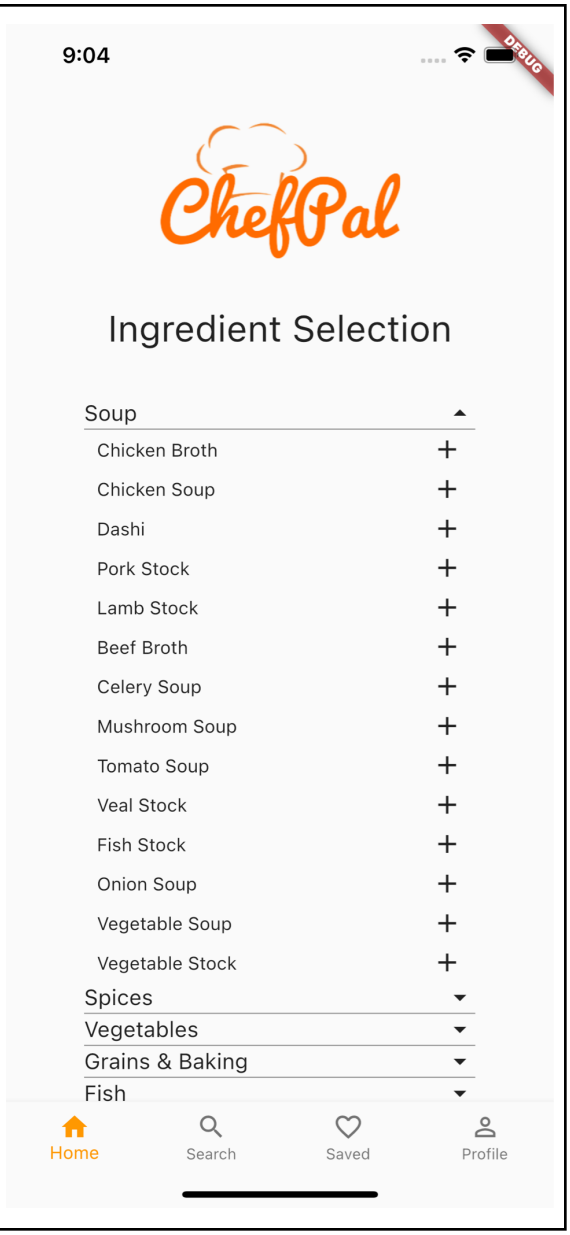
6) User clicks “Log In”.	

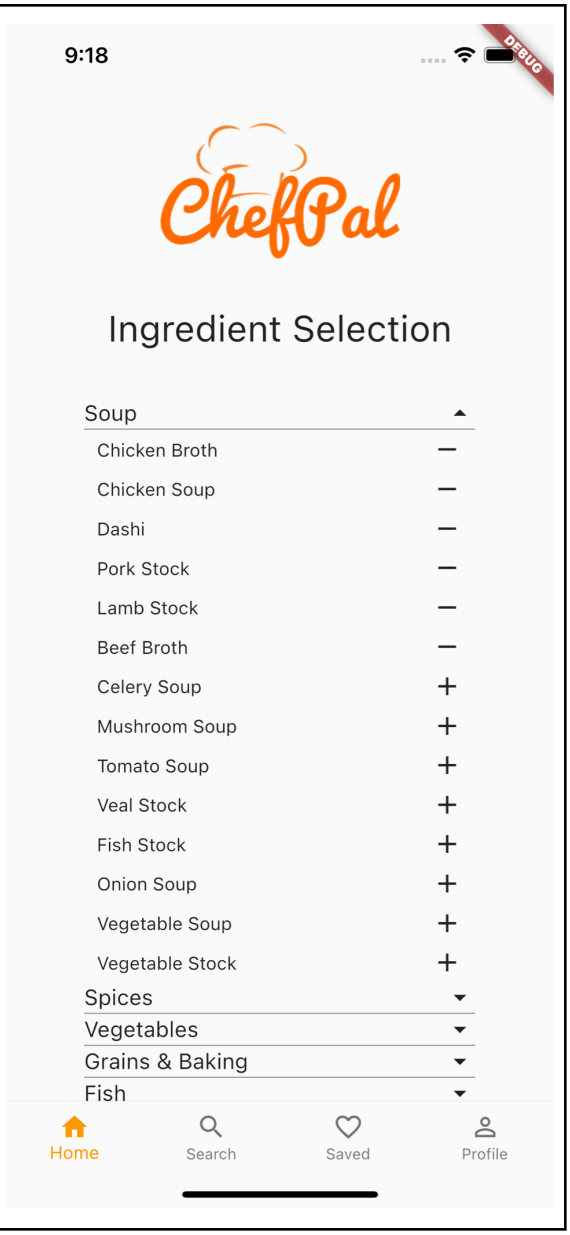
7) In the case the username or password is incorrect, the user gets the following screen.	

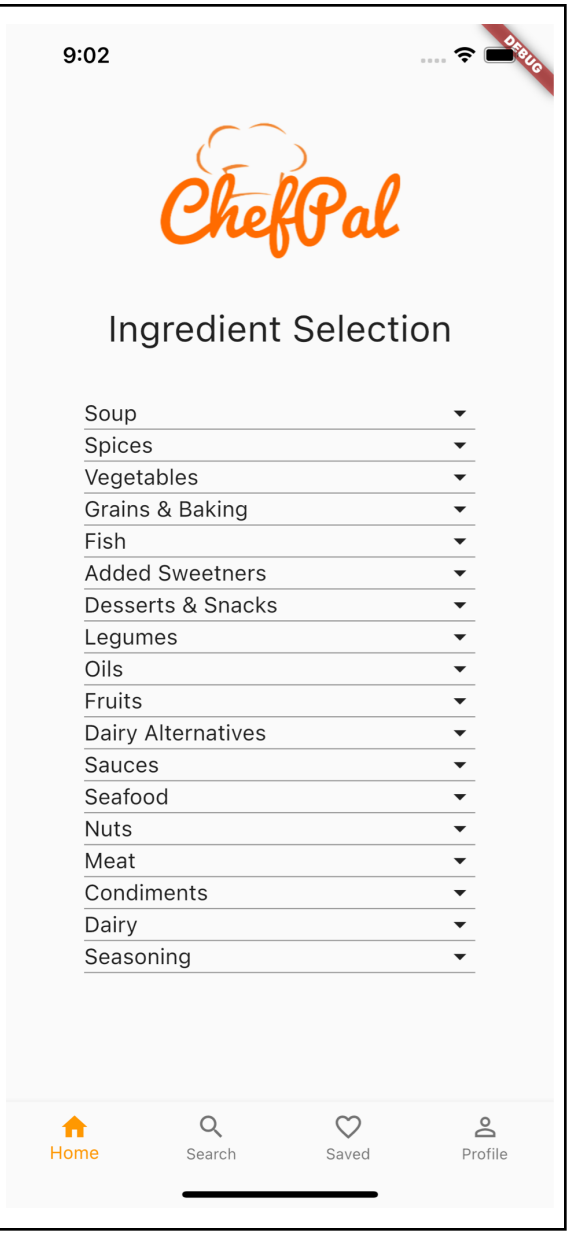
UC-4: Ingredient Selection:

Step-by-Step of how user enters information:

1) The user clicks on the shopping basket icon on the bottom tab.	

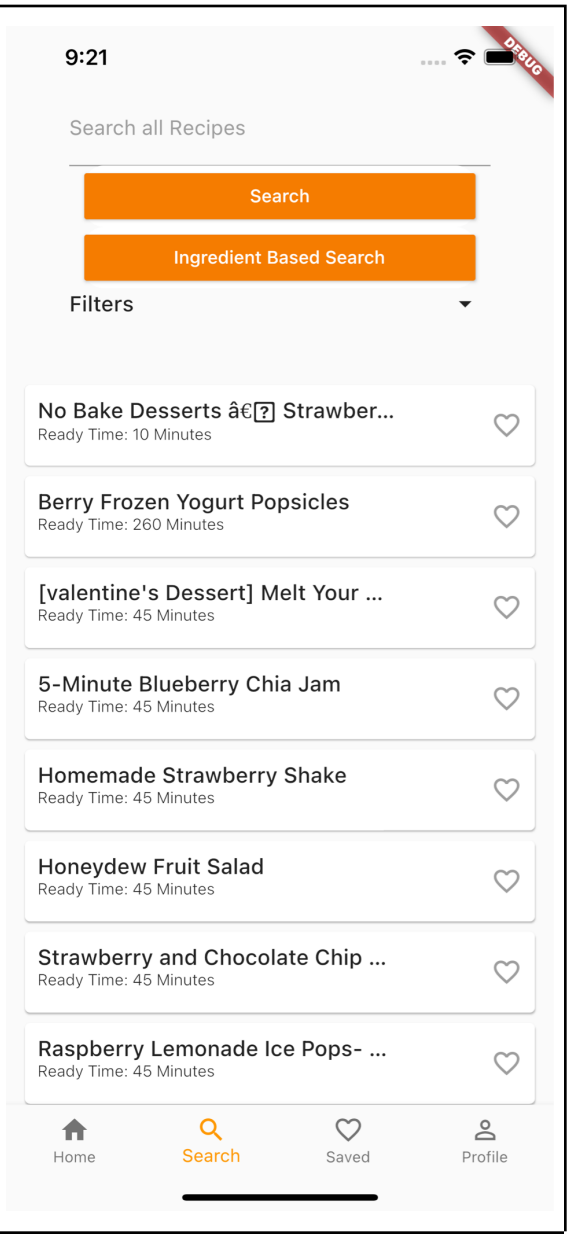
2) The user is presented with the titles of food categories that, when clicked, shows a drop down check list of ingredients.	

3) The user can click the buttons next to the ingredient to check it off.	

4) The user can click on the title of the drop down check list to minimize it again back to the original display.	

UC-5: Generated Recipes

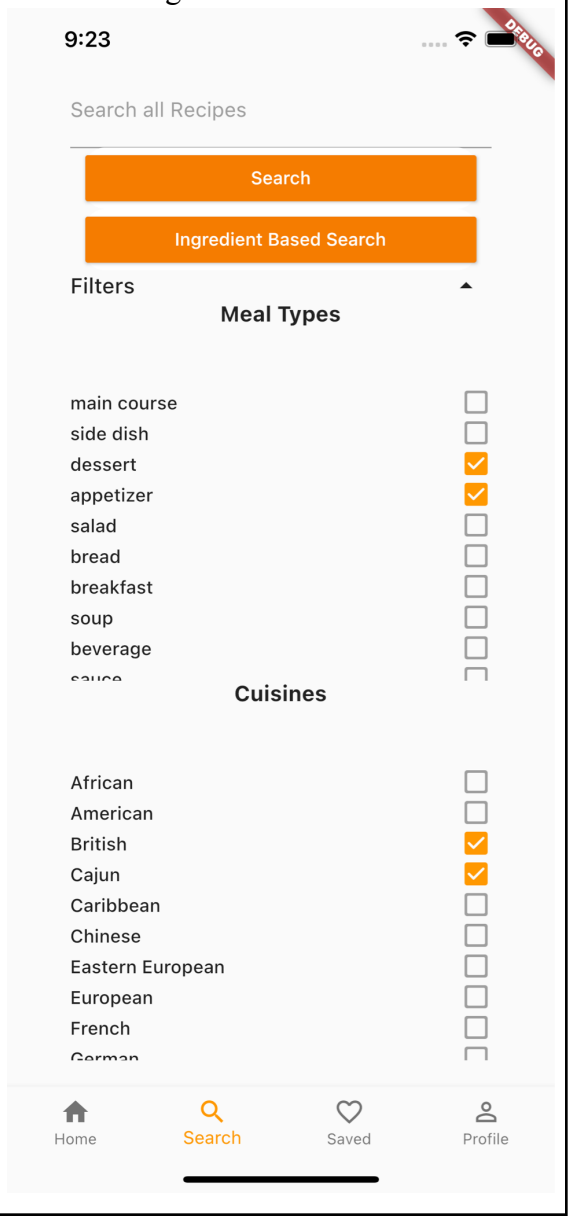
1) User clicks “Search” tab at the bottom and is presented with the generated list of recipes based on their previously entered dietary preferences and available ingredients.



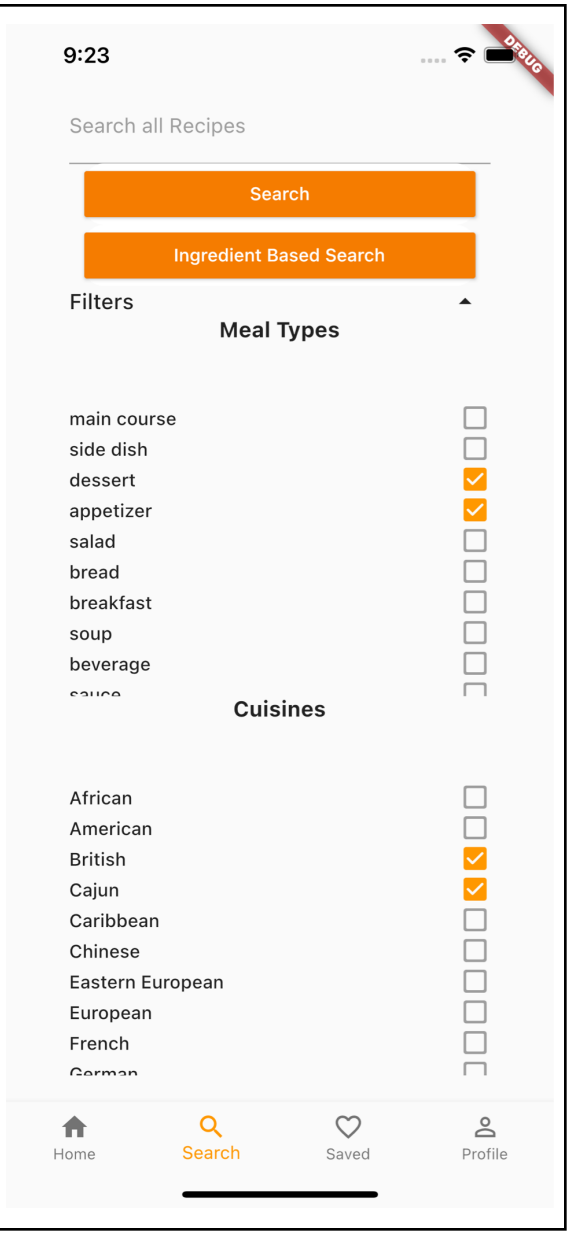
2) User can view a recipe by clicking on a title from the list. Users can click the back arrow to go back to the generated list.	

UC-6: Filtering System for Generated Recipes

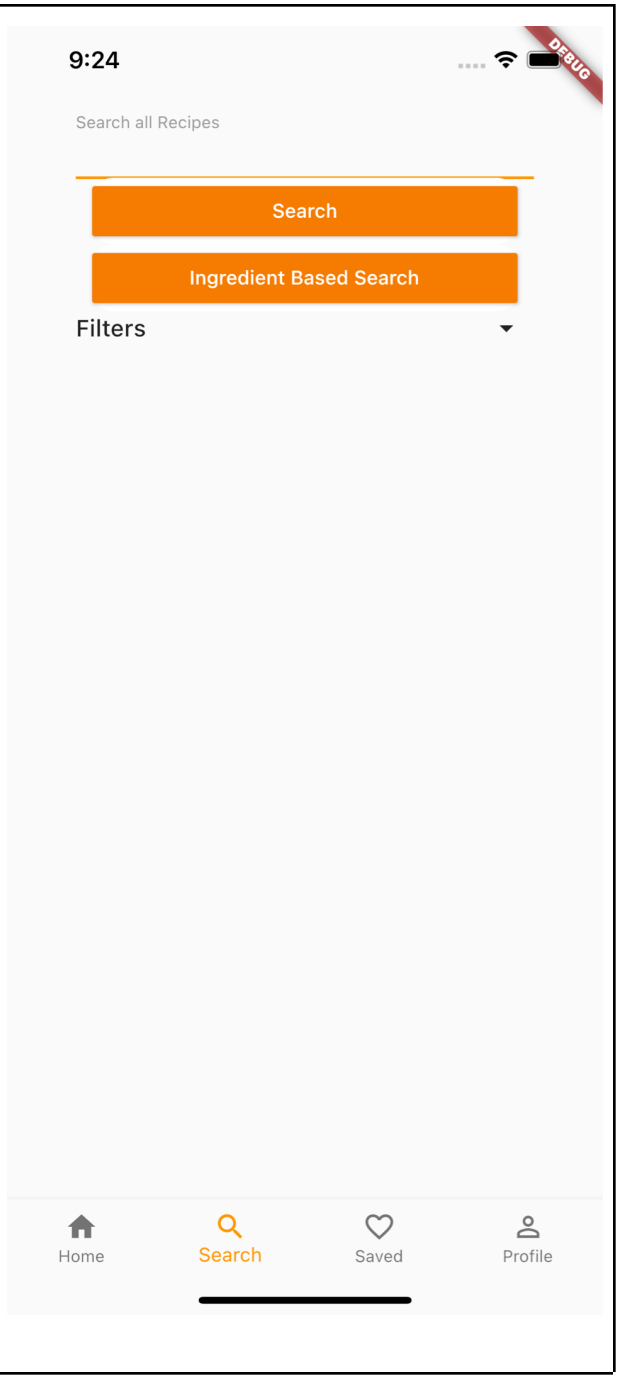
1) The user has the option to click on one or more of the filters at the top to further narrow down this list accordingly. Upon clicking a filter, the user will see the updated list.	Before clicking filter: 

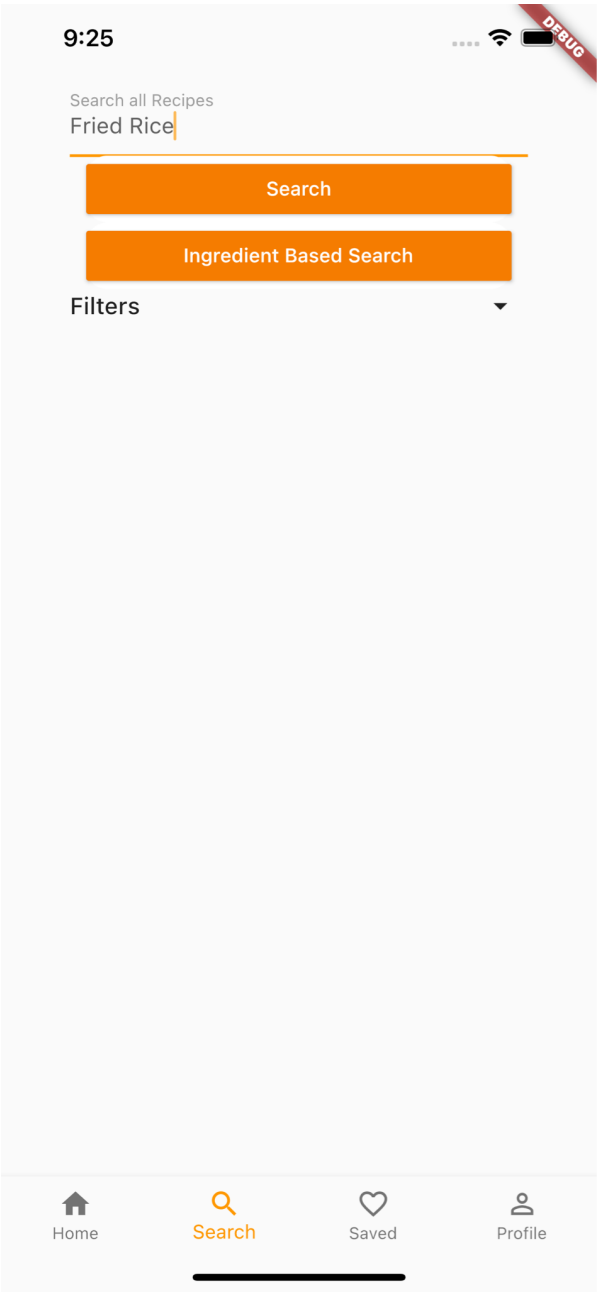
	After clicking filter: 

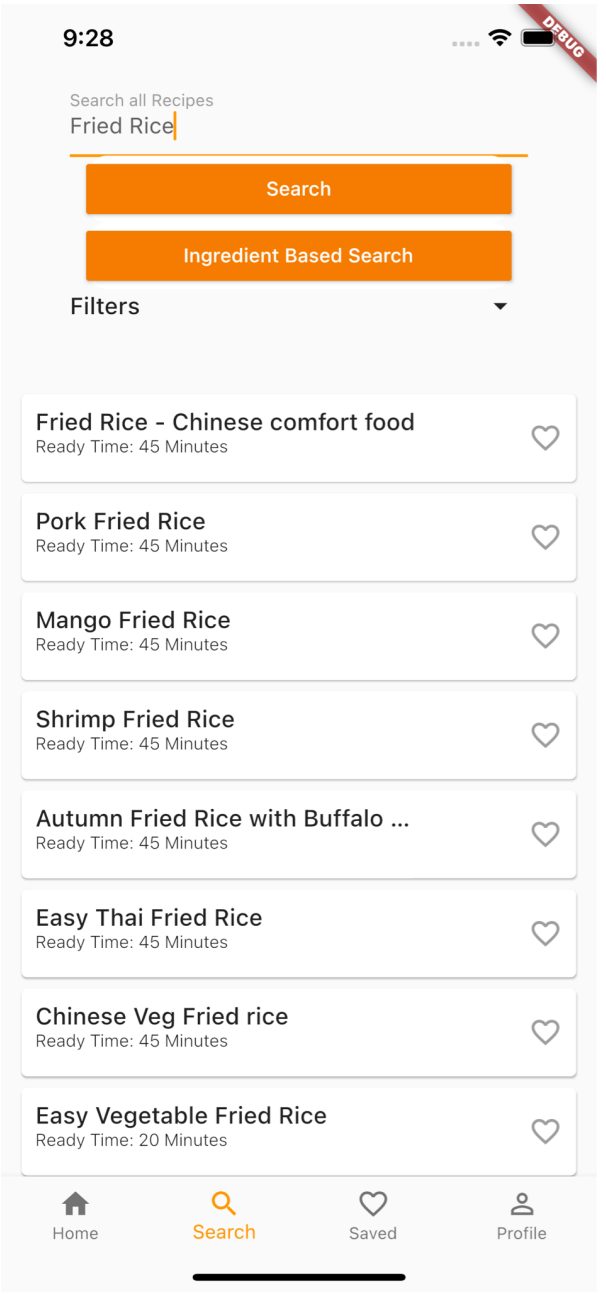
2) Users can see a full list of filters by clicking “Filter by” and can choose the filters there. They can then apply these by clicking “apply filters”.



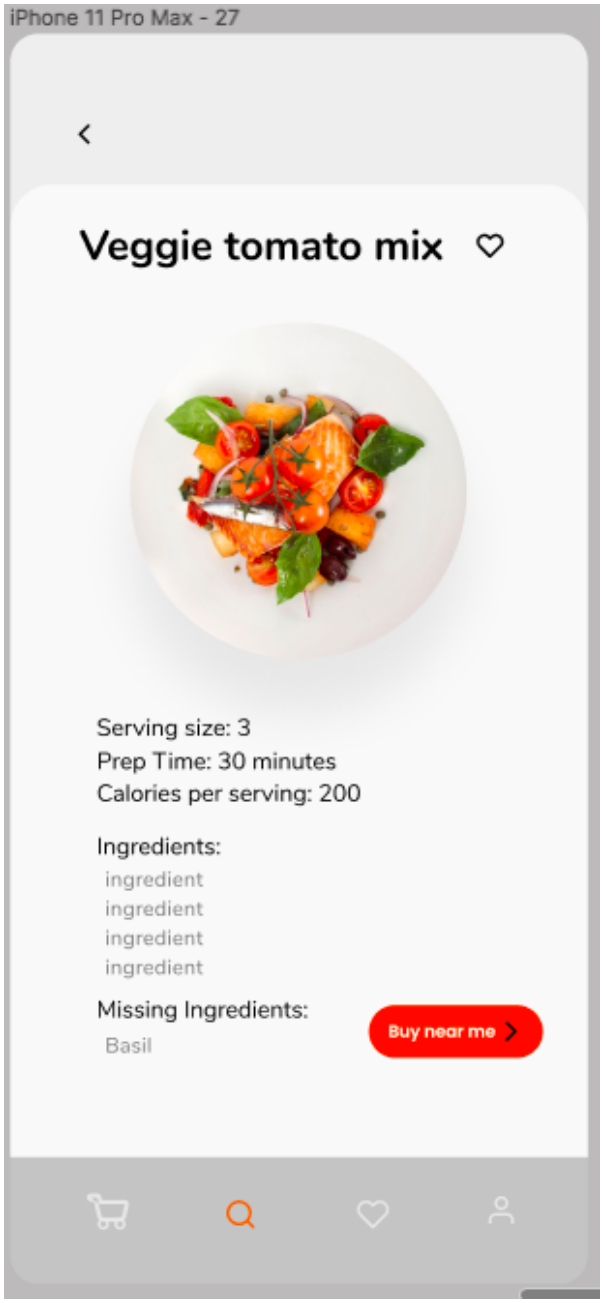
UC-7: Keyword Recipe Search

1) User clicks on the search bar.	

2) User inputs keywords for recipe search and clicks the search button.	

3) User gets a generated list of recipes related to the keyword, regardless if they have the ingredients or not. They can filter these recipes in the same way as the previous use case. They can also view these recipes in further detail as seen in UC-5.	

UC-10: Grocery Store Locator

1) User should have a recipe open to view that requires an ingredient that the user does not have. User clicks on “Buy Near Me”.	

2) User is redirected to a list of stores that sell the missing ingredient.	

12. Design of Tests

To ensure that our application works with no errors on the user interface and backend, we will be using Flutter to test the functionalities of the application. Below are the areas where the test cases will be used to ensure thorough functionality.

1. User Registration
    a. Goal: The user will be able to create a new account with a unique username and password.
    b. Test case: This test will be used to make sure that users can register themselves for a ChefPal account without any problems, given that the username is unique and that the password is sufficient.
        i. Success: The user creates a new account.
        ii. Failure: The user is unable to create a new account.
2. User Login
    a. Goal: The user will be able to login to his or her ChefPal account.
    b. Test case: This test will be to make sure that users can login into their ChefPal accounts without any problems.
        i. Success: The user logs into the ChefPal account with the correct username and password.
        ii. Failure: The user cannot login to the account, even with the correct username and password.
3. Ingredient Selection
    a. Goal: The user will be able to enter the ingredients to be used for recipe generation.
    b. Test case: This test will be to make sure that users can select ingredients into the application.
        i. Success: The user selects ingredients and will be able to view available recipes based off of the selected ingredients.
        ii. Failure: The user is unable to select ingredients, and the application will not register the ingredients as selected.
4. Recipe Search with Ingredients
    a. Goal: The user will be able to look up recipes based on his or her selection of available ingredients the user has that they selected from the drop down menu.
    b. Test case: This test will be to make sure that when users search for recipes using ingredients, the search results are valid.
        i. Success: The user views the available recipes based on his or her available ingredient list.
        ii. Failure: The user cannot view any recipes that include the selected ingredients available, or the application does not output any search matches, specifically when matches exist.
5. Recipe Search with Keyword
    a. Goal: For users to search for specific recipes using a keyword that they can type in.
    b. Test case: This test will assure that relevant recipes related to the keyword user types will be shown.
        i. Success: Recipes match the keyword user provided and outputs relevant results.

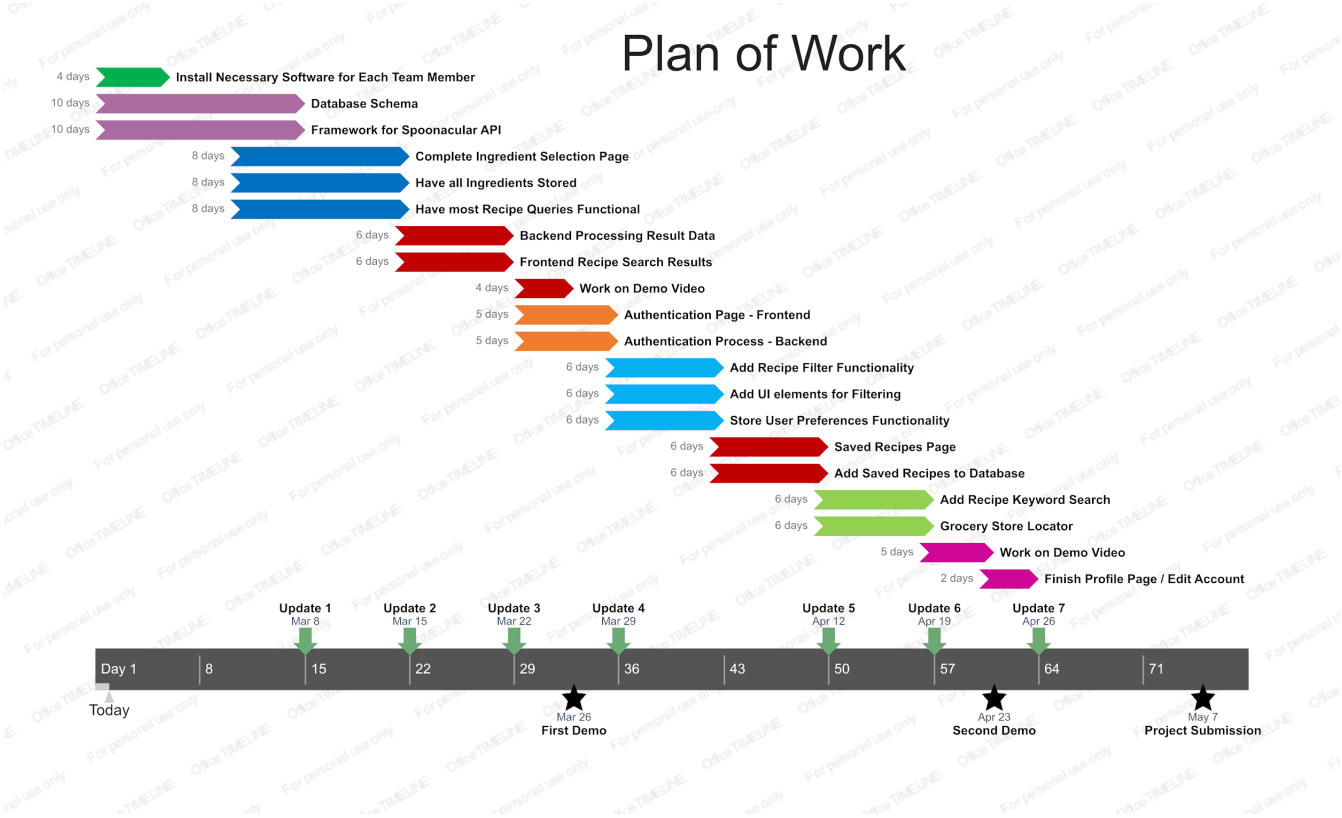
        ii. Failure: Recipes do not match to the keyword user provided and or yields no results to the keyword given.
6. Grocery Store Locator
    a. Goal: For users to locate a grocery store that has missing ingredients for their desired recipe
    b. Test case: This test will assure that the location of grocery stores will display if they possess what the user needs for their recipe
        i. Success: Locations of various grocery stores that have the missing ingredients in stock will show up for the user
        ii. Failure: No locations will display for users who are looking for their missing ingredients
7. Dietary Restrictions and Preferences
    a. Goal: User will be able to check off boxes that apply to their dietary restrictions and preferences
    b. Test case: This test will assure that the user can apply their dietary restrictions and preferences and generated recipes will reflect this
        i. Success: User will receive generated recipes that include their dietary restrictions and preferences, as well as include the ingredients they have
        ii. Failure: User will not receive any generated recipes that apply to their dietary restrictions and preferences
8. Recipe Filtering Functionality
    a. Goal: User can further narrow down their recipe search with more filtering options
    b. Test case: This test will assure that user can check off more filtered options and receive recipe results that fit their selections
        i. Success: Generated recipes will include the filtered options that the user selected, as well as following their ingredients, dietary restrictions, and preferences
        ii. Failure: Generated recipes will not include the filtered options the user selected
9. Saved Recipes Page
    a. Goal: For users to view the recipes that they have favorited in the past and be able to refer back to them
    b. Test case: This test will assure that the user can favorite a recipe and see the same recipe on their saved recipe page
        i. Success: User can refer back to their favorite recipe with ease in the saved recipes page
        ii. Failure: User can’t favorite a recipe or user can’t see the favorited recipe in their saved recipe page
10. User Interface
    a. Goal: To allow users to view and interact with the mobile application. Allowing users to input information and receiving results based on their information.
    b. Test case: This test will assure that the users can use the application by navigating through the different features of the application
        i. Success: User is able to interact and have full functionality of all the features in mobile application
        ii. Failure: User is unable to interact with mobile application

11. User Info Stored on Database
    a. Goal: To ensure that user information, such as username, password, ingredients, etc., are saved under the user's profile and is kept safe and secured
    b. Test case: This test will assure that a user's data will be saved and located correctly within the database
        i. Success: User's information is saved in database under their profile
        ii. Failure: User's information cannot be found or is located in the wrong place in the database
12. Authentication for Frontend and Backend
    a. Goal: To ensure that only registered users can access and use the mobile application
    b. Test case: This test will assure that only users who have made an account with the mobile application can access the application
        i. Success: A user who has previously made an account can sign in and use the application. A user who has not created an account cannot access the application
        ii. Failure: User cannot access account even with correct sign-in information. User who has not created an account can operate the mobile application

13. History of Work, Current Status, and Future Work

a. History of Work

In order to keep track of our group's progress, we initially created a Plan of Work in order to track the milestones and deadlines that we expected for ChefPal. These deadlines and milestones were meant to keep the group on pace in preparation for the completion of ChefPal. Our initial Plan of Work can be seen below.



At first, we decided to split the creation of ChefPal into four major parts in order to make the development of the mobile app easier on everyone. These major parts were essential ideas in order to give the user the full experience that we wanted to give them. Unfortunately, due to everyone's hectic schedules, sometimes other groups had to collaborate with others in order to speed up the process and keep us on track. Since some major parts relied on others to operate, it was easy to collaborate with one another to reach our milestones and deadlines.

As we advanced on the reports and the coding portion of the project, we were able to complete most of our milestones in order and at the appropriate deadlines that we assigned for them. By preplanning ahead of what we expected as a group, it made the development process that much easier for us. Although we were well organized with our milestones and deadlines, there were still times where we did not follow our plans through. For instance, we had added the keyword function into our program when filtering for recipes down even further. This was not originally in our milestones, but found that it could be useful to the user and decided to add it. In addition, there are instances where we pushed back deadlines and moved other deadlines forward. An example of this was when we had to push back the deadline for the ingredient selection page. This was due to having a lot of required information to be inputted and also required a database to store the user's choices. Furthermore, we had moved the recipe filtering functionality milestone up, as that was easy and quick to complete.

As we approach the completion of ChefPal, staying on top of deadlines and managing what we expect from each other has allowed us to easily develop a mobile application for a unique user experience. Having the milestones and deadlines evolving from a basic idea to reality has been really inspiring to keep pushing ourselves to make ChefPal even better. In comparison to the first and second reports, the milestones have mostly stayed the same throughout the entirety of ChefPal; for, we established these milestones from the outset of the developmental process.

b. Current Status

ChefPal is now a final product and can complete the following key accomplishments:

- Login page
- Sign-up page
- Recipe search
- Drop down menus for selecting ingredients
- Recipe filtering through dietary restrictions and preferences
- Recipe keyword search
- Saved recipe page
- User interface design from scratch
- Storing user information onto our database
- Functioning grocery store locator

Our current product is a mobile application with a focus in generating recipes. On this mobile application, there are different tabs that allow for the user to interact with ChefPal in different ways. The major functions that the user will experience are to select their ingredients and dietary restrictions, search for recipes based on their ingredients and preference, view liked recipes, and to manage the user's profile. In addition, there are more features offered for the user experience, such as more filtering options for recipes, detailed information on all recipes, and a grocery store

locator for missing ingredients. While still being simple and easy to use, ChefPal is fully functional and offers the user a unique experience.

c. Future Work

Although we have accomplished implementing our key functionalities for ChefPal, there is still plenty of room for improvement. During the initial stages of the project, we wanted to implement more detailed features that would benefit the user. For instance, we wanted to list the price of the missing ingredient as well as the location of the grocery store. We decided against including this feature because stores do not update their prices and because potential promotions would disrupt the prices the user would see. Additionally, some chain grocery stores might have different prices at different locations, depending on the location.

Another feature that we wanted to implement into ChefPal was adding a quantity, or amount of ingredients. This would help generate recipes more specific to the user and more accurately to the exact amount of ingredients that the user actually has. If a recipe calls for 10 apples, but the user only has 2 apples, then the recipe would not exactly match the user's needs at that moment. Adding this feature would tremendously increase the usefulness of ChefPal.

Suggesting recipes based on frequent recipes generated by the user is also something that would be beneficial. Implementing machine learning for recipe suggestions based on all of the recipes the user has searched for in the past will provide the user with a more comprehensive experience for their cooking. This has not been implemented because there is no user base currently, which means there is no data to base these suggestions off of. A pre-trained system with embedded on-device learning, such as an autoencoder, could be used for such an application.

Another interesting feature that should likely be added in future iterations is the ability to filter using metrics not offered by Google Maps. ChefPal would communicate with store databases, which are not accessible to the general public, to identify if they have ingredients required in stock.

In the end, we were able to create a mobile application that provides a unique one-stop-shop grocery shopping experience to the user. Creating a customized search engine that will give suggestions on how to begin the meal-prepping journey would be the next logical step in the life cycle of ChefPal. Due to time restraints and other class work needed to be completed, there are features that we wish ChefPal would offer the user. Our main goal was to make ChefPal operational for users, but there will always be a way to make ChefPal even better for the user in the future.

14. References

Amazon Web Services, Inc. 2021. *Build a Flutter Application on AWS*. [online] Available at: <https://aws.amazon.com/getting-started/hands-on/build-flutter-app-amplify/> [Accessed 24 February 2021].

Amplify Framework Documentation. 2021. *Amplify Framework Documentation*. [online] Available at: <https://docs.amplify.aws/lib/datastore/sync/q/platform/flutter> [Accessed 24 February 2021].

Docs.aws.amazon.com. 2021. *Getting started in React Native - AWS SDK for JavaScript*. [online] Available at: <https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-started-react-native.html> [Accessed 6 March 2021].

Eceweb1.rutgers.edu. 2021. [online] Available at: <http://eceweb1.rutgers.edu/~marsic/books/SE/projects/HealthMonitor/2019-g2-report3.pdf> [Accessed 3 March 2021].

Eceweb1.rutgers.edu. 2021. [online] Available at: <http://eceweb1.rutgers.edu/~marsic/books/SE/projects/Restaurant/2019-g13-report3.pdf> [Accessed 16 February 2021].

Ers.usda.gov. 2021. *USDA ERS - Food Prices and Spending*. [online] Available at: <https://www.ers.usda.gov/data-products/ag-and-food-statistics-charting-the-essentials/food-prices-and-spending/#:~:text=In%202019%2C%20households%20in%20the,representing%208.0%20percent%20of%20income> [Accessed 16 February 2021].

Figma. 2021. *Figma: the collaborative interface design tool.*. [online] Available at: <https://www.figma.com> [Accessed 14 February 2021].

Firebase. 2021. *Firebase Realtime Database*. [online] Available at: <https://firebase.google.com/docs/database> [Accessed 22 March 2021].

Firebase.flutter.dev. 2021. *FlutterFire Overview | FlutterFire*. [online] Available at: <https://firebase.flutter.dev/docs/overview/> [Accessed 6 March 2021].

Flutter.dev. 2021. *Firebase*. [online] Available at: <https://flutter.dev/docs/development/data-and-backend/firebase> [Accessed 24 February 2021].

Flutter.dev. 2021. *Write your first Flutter app, part 1*. [online] Available at: <https://flutter.dev/docs/get-started/codelab> [Accessed 24 February 2021].

Google Developers. 2021. *Overview | Places API | Google Developers*. [online] Available at: <https://developers.google.com/places/web-service/overview> [Accessed 16 February 2021].

Lucidchart. 2021. *Online Diagram Software & Visual Solution | Lucidchart*. [online] Available at: <https://www.lucidchart.com/pages/> [Accessed 12 March 2021].

Marsic, I., 2021. *Software Engineering - Final Project Report*. [online] Ece.rutgers.edu. Available at: <https://www.ece.rutgers.edu/~marsic/Teaching/SE/report3.html> [Accessed 1 April 2021].

Medium. 2021. *Size matters: Reducing Flutter App size best practices*. [online] Available at: <https://suryadevsingh24032000.medium.com/size-matters-reducing-flutter-app-size-best-practices-ca992207782> [Accessed 24 February 2021].

Medium. 2021. *Must try: Use Firebase to host your Flutter app on the web*. [online] Available at: <https://medium.com/flutter/must-try-use-firebase-to-host-your-flutter-app-on-the-web-852ee533a469> [Accessed 6 March 2021].

Rnfirebase.io. 2021. *React Native Firebase | React Native Firebase*. [online] Available at: <https://rnfirebase.io> [Accessed 6 March 2021].

Spoonacular.com. 2021. *spoonacular recipe and food API*. [online] Available at: <https://spoonacular.com/food-api> [Accessed 16 February 2021].

15. Project Management

In order to organize our group's thoughts and ideas, our group is using an instant messaging platform called Discord. With Discord, we are able to communicate with one another through messaging and voice chat. In addition, our groups had established specific times where we all would get together and discuss the next stage in our project and make sure we are all on the same page with everything going smoothly. Meetings were usually every Monday and Thursday at around 12 PM, which lasted around an hour. We have one group member working asynchronously from a different time zone, and one with scheduling conflicts. Also, we would work together on the weekend to finalize our report by editing and organizing any last minute changes. We used breakout rooms in Discord to work on the implementation of our code based on our sub-groups. In order for everyone to contribute towards writing and formatting the report, we used Google Docs in order to share and collaborate on the completion of the report as a group. Github was used to share code with one another, which allows us to collaborate on certain coding aspects of the project.

For the coding portion of the project, we will be using JavaScript - backend - and Dart - for frontend - in addition to whichever modules work best with mobile app integration. Furthermore, we decided to use Flutter for app development, which uses Dart. In addition to its syntax simplicity, Flutter also offers native support for many API's that we use for both recipe and grocery store search, such as Spoonacular and Google Maps API.

For the UI design, we had used Figma during the preliminary design to illustrate the application and the steps a typical user would go through to access and find recipes. Figma allows us to specify what we need and how we want our mobile application to look like in its final stage. With this user interface design, we are able to offer a preview of what our users should expect when using this application. We then implemented this UI mock-up design with the use of Flutter for frontend development. In Flutter, we used Dart to implement the UI design in order to develop the final UI design for our software development project.

For the database and storage of our code, we will be using Firebase. Firebase will be used for user registration and authentication in order to allow us to store the user's data and other data required for this project. With the use of Firebase, we will most likely be using SQL web services.