Software Engineering: 14:332:452:01
Group 1
Home Kitchen Automation

ChefCart

https://github.com/max-legrand/chefcart

Project Report 3
23 April 2021

Team Members:

- Indrasish Moitra
- Milos Seskar
- Shreyas Heragu
- Kevin Lin
- Maxwell Legrand
- Jonathan Wong
- Allen Chang
- Elysia Heah
- Mark Stanik
- Brandon Luong

Individual Contributions

All team members contributed equally

Table of Contents

Summary of Changes

- Edited Customer Statement of Requirements
- Edited System Requirements
- Edited Functional Requirements Specification
- Edited Use Case and Use Case Diagram
- Edited Traceability Matrix
- Edited Glossary of Terms
- Edited System Sequence Diagrams
- Edited Interface Requirements
- Edited Complexity Factor
- Edited Domain Analysis and Traceability Matrix
- Edited Domain Model Diagram
- Edited Table of Contents
- Edited Data Model
- Edited Database Schema
- Edited System Requirement
- Edited the Interface Diagram and content
- Edited Class Diagram
- Edited Data Types and Operation Signatures
- Included Design Patterns
- Included OCL
- Edited Data Structures
- Edited User Interface Design and Implementation
- Edited Design of Test Cases
    - Included Algorithm Nonfunctional requirements and UI requirement test cases

Customer Problem Statement

Problem Statement:

Cooking is a skill that many people struggle to learn. Cooking carries many benefits, as it allows the chef to both save money and have better control over their nutritional intake. In 2016, results from a survey demonstrated that 80% of millennials thought that cooking at home was a "good way to live". However, despite all the positive attributes associated with cooking, only 45% of those ages 18-24 and 64% of those 25 to 34 consider themselves to be "somewhat good or "good" at cooking. If cooking is portrayed to be such an important skill, then why do so few people know how to do it?

Learning how to cook is deceptively complex and time consuming for the chef. Cooking requires ingredients, which involves being able to keep track of what the chef already owns, while also having to periodically buy groceries from the supermarket. To top it off, chef's need to find recipes that both match what they want to eat and can be created from what they want in the fridge. Specific dietary needs have to be met, and finding particular recipes that match certain diets can be frustrating. Cooking takes a lot of effort, and those who don't see the effort worth the benefits in cost and nutrition will not bother trying to learn how to cook in the first place.

With the rise of the COVID-19 pandemic, a record high amount of people are staying home due to isolation practices. This has driven up the need of home-cooking, as many restaurants are closed or limited in capacity. Cooking from home provides a safe way for people to get a good meal, without having to put themselves at risk by going to a restaurant.

Software can alleviate these issues by automating the tedious parts of cooking. Problems such as memorizing ingredients at home and ingredients that need to be bought can be solved using a digital list. Many people use their web browsers at many times throughout the day and therefore can check their ingredients digitally whenever they want. The API spoonacular contains price and nutrition information about ingredients, while also containing different recipes. Spoonacular can search for specific recipes based on an ingredient list and dietary restrictions. Using this API can allow a user to easily generate recipes depending on what they have in their fridge. This would eliminate the time needed to search online for a recipe that hits all the factors of: matching ingredients, dietary restrictions, and desired dish of the user.

This product would be beneficial to not only beginners, but advanced home cooks as well. Taking a look at multiple people's perspectives will help solidify a need for our product.

Non-home Cooks/Beginners:

Cooking at home can be extremely difficult and time consuming. Not only do I need to keep track of key ingredients I have, I have trouble finding recipes for the ingredients I do have. I could always go grocery shopping for missing ingredients, but I have a 9-5 job and just want to relax after work. It can be really time consuming to pick out a good recipe beforehand and have time to go grocery shopping while also keeping in mind the ingredients I already have. Because of all this, it becomes much easier to just get restaurant takeout every night.

I would consider cooking at home more if I had something to let me know what ingredients I have, while also picking appropriate recipes for the ingredients I do have or giving me a missing ingredient list alongside a recipe. This would save me a lot of hassle and worry about missing ingredients and picking good recipes. I would have more time in my busy day and will be able to save more money and eat healthier by cooking at home more.

How will ChefCart help me? This application will be able to help me keep track of ingredients I already have via a digital pantry, so I can always see what I have and what I need right from my computer. A great part of the application being a web app is that in the grocery store, I can also log into my account on my phone and pick up where I left off. The application will also help me pick recipes that are simple enough to make for my level and will taste good. If desired, the ChefCart account can apply to my whole family and not just myself. Involving other people would make cooking more fun and enjoyable.

Picky Eaters:

I love to cook at home, but I can never find new recipes that I can actually eat. With being a vegan and having gluten sensitivity, my options are limited and I am often stuck making the same meals throughout the week. I would love to explore new cuisines and cultures, but it is difficult with all my food restrictions. I have to spend a lot of time researching different dishes and altering them myself to get a version that abides with all my restrictions. Furthermore, finding grocery stores that sell certain ingredient substitutions can be challenging. Going to a store only to find it does not have what I need is often demoralizing and makes me want to give up on finding new dishes.

I would definitely cook more diverse dishes if I could only find good recipes for all my food needs. It would be great if there were something to give me certain recipes given all my restrictions. If there were also some way to know what ingredients I need and where to find all the necessary ingredients, it would save me a lot of time and frustration and make cooking more appealing.

How will ChefCart help me? This application will generate a set of recipes for me to cook. I only need to input my restrictions and then I will be able to pick from a bunch of recipes that I know will be safe for me to cook and eat. Whenever I want to make a meal, I can select a recipe and it will tell me what ingredients I am missing and where to find them. The application will use Walmart and Wegmans apis as well as web scraping to find local stores that carry the proper items.

Advanced Home Cooks:

I often cook at home and have many ingredients already. Many times I would like to just cook a quick meal with the ingredients I already have; however, it can be a pain to scour recipe sites in search of recipes I can readily make. I would love to explore new dishes, but I often find I am missing a few key ingredients. If there were a way for me to find recipes without having to make a grocery run for only a few items, I would definitely be willing to cook more diverse dishes.

How will ChefCart help me? This application will keep track of the ingredients I already have. When I want to make a quick, unique meal, I can search for specific cuisines and ChefCart can generate recipes that only contain ingredients that I already have, saving me a trip to the supermarket. This will make exploring new dishes and recipes much easier. A great feature of ChefCart is how it will keep track of existing ingredients. By updating different weights or number of ingredients, the application will notify me if I am running low on certain ingredients and notify me if I am missing ingredients for a recipe.

Foodie:

I love food. I love exploring different cultures through myriad cuisines and recipes. Normally I like to eat out and discover cultures through local restaurants, but that limits my reach to what is nearby. Furthermore, it isn't economically feasible to eat out all the time. I want to cook at home more with the ingredients I already have; however, sometimes I have difficulty discovering new recipes that fit my diet and intolerances while also showing me a new cuisine type.

I would definitely cook at home more versus eating out if I had the ability to cook new cuisines I want to explore. It would be great if there were something to give me recipes from a certain cuisine. It would be even better if it could also filter cuisines to fit my required diets and intolerances. I really want to explore italian cuisine, but I am lactose intolerant and cannot eat cheese, a staple in italian food.

How will ChefCart help me? This application will generate a set of recipes for me to cook. These recipes will take into consideration my desired cuisine in addition to my current diet and intolerances. When I want to explore a certain cuisine, I can simply input that cuisine and ChefCart will take into account my set diet, intolerances, and what's in my pantry, and ChefCart will return a list of recipes that I may enjoy.

Overall, ChefCart aims to help cooks of all levels through software and automating the home cooking process. By keeping track of current ingredients; finding recipes with desired ingredients, cuisines, diets, and/or intolerances; keeping track of missing ingredients via a grocery list; and helping cooks find ingredients at their local grocery stores; ChefCart is able to eliminate many barriers for home cooks everywhere. ChefCart makes cooking easy, streamlined, and hassle free for all people.

Problem Decomposition

Problem 1: Keep track of pantry items

- Ingredient Data Structure - Ingredients will be treated as a dictionary data type. The keys of the dictionaries will be relevant information to be recorded for each ingredient. This includes name, quantity, weight, volume, expiry dates, threshold etc. For example, we can create a dict as follows:

  bananas = {name: “bananas”, quantity: “7”, weight: “”, volume: “”, expiry: “2/18/21”, threshold: “”}

  We will then design sections of the web app to appropriately show key value pairs for ingredient dictionaries so the user can see useful information about the ingredients.

- Adding/Changing ingredients - We will create inline buttons to add ingredients to the pantry. On clicking the button the user will be given text boxes for key value pairs. After filling the boxes the user can submit the new dictionary to the Digital Pantry database. We will also add inline buttons to discrete metrics like quantity or weight so that they can be changed without having to repopulate the other key value pairs.
  Finally, we will add a button for removing pantry ingredients. This feature will be paired with notifications that are tied to expiry dates and ingredient quantities.

Problem 2: Manage the grocery list

- Grocery List Data Structure - The grocery list will be generated on demand for the user each time they would like to purchase groceries. It will query the database for the Digital Pantry to add any missing ingredients to the grocery list based on the thresholds set by the user.

- Add and Remove Items from grocery list - Similar to the digital pantry, we will have inline buttons for adding, removing, or editing items to/on the grocery list. We will use the user input from these buttons to update the grocery list tables displayed to the user.

- Integrate with Digital Pantry - Use the ingredient dictionary structure to update ingredients that have been purchased using the grocery list. Save these changes locally in the user interface for one time use.

- Find a Store - Once the grocery list is generated the user must be able to locate these ingredients at local stores. This will be achieved by leveraging grocery store API like the Walmart API to search for ingredients at the local store. When the ingredients are found, the store information is displayed at the top and the ingredients that the store sells are listed below in a digestible list. Each item is shown to the user with a link to see it in the store website, the ingredient price, reviews, ratings, and stock.

Problem 3: Recipe Generator

- Integrate Spoonacular API - Create "go-between" functions that ask the user for necessary Spoonacular inputs, especially ingredients, and package them to deliver to the API. We must also explore different Spoonacular functions to add user parameters like dietary restrictions. Finally, we need to find API limitations and assess the user's need for those features, so that we can choose to develop them as needed.

- Identify missing ingredients - Query the Digital Pantry database with the ingredients for a selected recipe. Notify the user that there are missing ingredients in the recipe.

- Sort and filter recipes by important metrics - This will be achieved using SQL commands. Most commonly we can use "order by" on ingredient name, missing ingredients, and other fields that will be available for the generated recipes. We will also use "where" to choose binary parameters like whether or not a recipe is vegetarian. More particularly, we would want to filter recipes based on ingredients used, cuisine, and dietary preferences and intolerances.

Glossary of Terms

Home Cook **-** An individual that creates food in their own kitchen with their own ingredients rather than ordering from a restaurant

Pantry **-** A small room or closet where food and ingredients are kept.

Recipe **-** A set of instructions to create and prepare a certain dish. Usually will also list the ingredients that you need.

Spoonacular **-** A website that allows you to plan your meals in advance and save different recipes from around the web so you can have it all in one place

Grocery List - A list of ingredients that the user wishes to buy.

Filter - To remove recipes that have certain ingredients that are unwanted by the user.

System Requirements

Business Goals

Business Goal Diagram

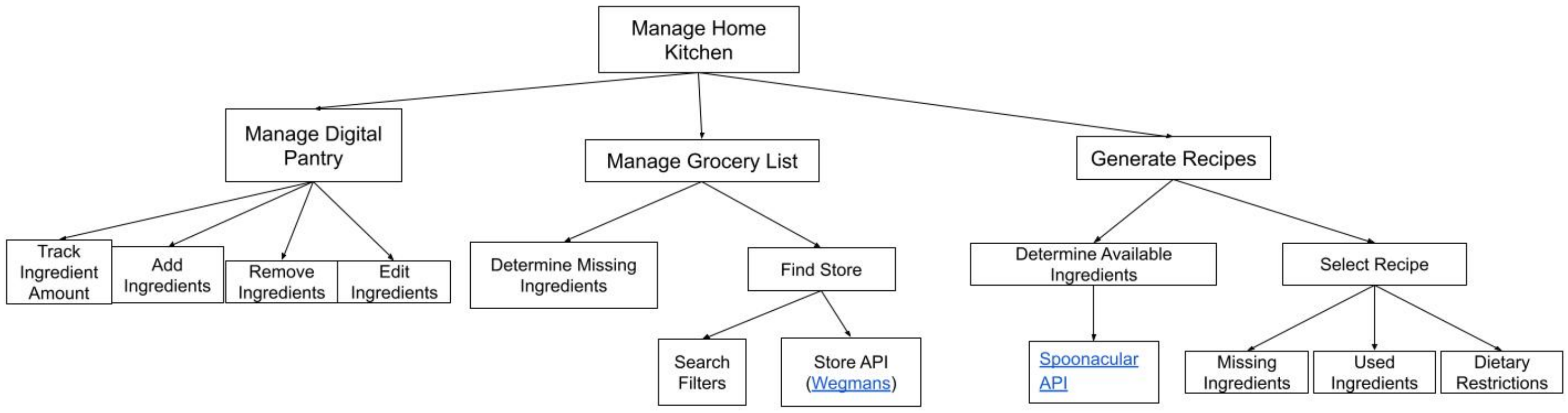


Figure 1: Business Goals Decomposed

The Business goals are divided into three main parts. The Digital Pantry, Grocery List, and Find Recipes are the main parts. The digital pantry needs to be able to track and update ingredients. The grocery list needs to be able to determine missing ingredients and be able to find a store that sells the ingredient. To generate recipes, we need to be able to determine available ingredients and be able to select a recipe based on the used ingredients and missing ingredients and dietary preferences. Below are the requirements for the end user to help satisfy these business goals.

Enumerated Functional Requirements

*Higher numbers are used to represent higher priority.
*User Authentication and Account Editing are assumed.

Enumerated Functional Requirements - Web App General Usage

Label	Priority Weight	Description
REQ-1	3	ChefCart will allow users to switch between the Digital Pantry, Recipe Generator, Home Page, and Grocery List features on a Web App.
REQ-2	2	ChefCart will allow users to set their account dietary restrictions in their Edit Account Page.

Table 1: Functional Requirements for Web App General Usage

Table 1 lists the functional requirements for general usage for the web app with a priority weighting and a description.

Enumerated Functional Requirements - Digital Pantry

Label	Priority Weight	Description
REQ-3	5	ChefCart will allow users to add and remove ingredients to a digital pantry as well as edit their amounts.
REQ-4	5	ChefCart will allow users to save their digital pantry for later usage.
REQ-5	4	ChefCart will allow users to set their required threshold for each ingredient in their digital pantry and show a warning if below threshold.
REQ-6	4	ChefCart will allow users to view the ingredients name, quantity, weight, volume, and expiration date in their digital pantry.

Table 2: Functional Requirements for Digital Pantry

Table 2 lists the functional requirements for the digital pantry with a priority weighting and a description.

Enumerated Functional Requirements - Grocery List

Label	Priority Weight	Description
REQ-7	3	ChefCart will allow users to determine missing ingredients in their digital pantry and create a grocery list from it for viewing.
REQ-8	4	ChefCart will allow users to find the closest store to purchase their grocery list items with a sort for item name, availability, price, rating, and number of reviews.
REQ-9	3	ChefCart will allow users to add and delete additional ingredients to their grocery list if desired.

Table 3: Functional Requirements for Grocery List

Table 3 lists the functional requirements for the grocery list feature with a priority weighting and a description.

Enumerated Functional Requirements - Find Recipes

Label	Priority Weight	Description
REQ-10	3	ChefCart will allow users to input ingredients, instead of using the digital panty.
REQ-11	5	ChefCart will allow users to generate recipes that they can cook with as much of their available ingredients and view them via Web App.
REQ-12	4	ChefCart will allow users to filter the generated recipes by important metrics such as dietary preferences, intolerances and cuisines.
REQ-13	4	ChefCart will allow users to select a generated recipe to view required ingredients, instructions, price, and missing ingredients.

Table 4: Functional Requirements for Finding Recipes

Table 4 lists the functional requirements for the finding of recipes with a priority weighting and a description.

Enumerated Functional Requirements - Additional Functional Requirements

Label	Priority Weight	Description
REQ-14	4	ChefCart will verify the ingredients that the user wants to add.
REQ-15	3	ChefCart will allow users to add missing recipe ingredients to their grocery list.

Table 5: Uncategorized Additional Functional Requirements

Table 5 lists the additional requirements that do not belong to any particular feature. REQ-15 is not being implemented.

Enumerated Nonfunctional Requirements

Enumerated Nonfunctional Requirements

Label	Priority Weight	Description
REQ-16	4	ChefCart should securely maintain user data. Data will not corrupt, will not be accessible by anyone other than the account user and will save automatically with 99% accuracy.
REQ-17	4	ChefCart should be able to run on the iOS 10 and above, Android 8.0 Oreo and above, Windows 7 and above, and MacOS Catalina (10.15) and above as well as the major Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari browsers.
REQ-18	2	ChefCart should be able to startup and shut down quickly within 3 seconds.
REQ-19	3	ChefCart should have smooth transitions from page to page that last less than 1 second.
REQ-20	3	ChefCart should present relevant data within 3 seconds and would allow a person with average eyesight to read it from 1 meter.
REQ-21	3	ChefCart should display properly when resizing the application window.

Table 6: Table of Nonfunctional Requirements

Table 6 lists the nonfunctional requirements for the end user. These requirements apply to the entire application itself. There is a priority weighting and description for each requirement.

User Interface Requirements

*User authentication User Interface Requirements are assumed.

UI Requirements - Web App First Time Usage

Label	Priority Weight	Description
UIREQ-1	3	A user that launches ChefCart will be greeted by the main page where they can choose to login or sign up.
UIREQ-2	3	Upon clicking the login button in the login page with the correct credentials, the user will be taken to the Home Page..
UIREQ-3	5	All pages beyond the Login Page will have buttons for the user to

		navigate between the Home Page and then to Edit Account, Digital Pantry, Grocery List, and Find Recipes pages.
UIREQ-4	5	Clicking on any of the Edit Account, Digital Pantry, Grocery List, and Find Recipes buttons on the Home Page will take the user to the corresponding page.
UIREQ-5	4	On pages beyond the Main Page, there will be a back button option to navigate back to the previous page.
UIREQ-6	3	In the Edit Account page, the user can edit their account details and save them.
UIREQ-7	4	Clicking the Logout button lets the user logout of ChefCart and returns them to the Main Page

Table 7: Table of Web App First Time Usage User Interface Requirements

Table 7 lists the User Interface requirements for using the web app for the first time. Many sign up and registration requirements as well as some general usage is listed here. There are priority weightings and descriptions for each requirement.

UI Requirements - Digital Pantry

Label	**Priority Weight**	**Description**
UIREQ-8	5	In the Digital Pantry, clicking the edit button, it lets the user edit the ingredient amount.
UIREQ-9	5	In the Digital Pantry, clicking Add will let the user add a new ingredient.
UIREQ-10	5	In the Digital Pantry, clicking the delete button will let the user delete the corresponding ingredient
UIREQ-11	5	In the Digital Pantry, allow users to view the current ingredients in the pantry in a tabular fashion.

Table 8: Table of Digital Pantry User Interface Requirements

Table 8 lists the User Interface requirements for using the digital pantry. There are priority weightings and descriptions for each requirement.

UI Requirements - Grocery List

Label	Priority Weight	Description
UIREQ-12	4	In the Grocery List Page, missing ingredients will already display and the user can click Add to add more ingredients to their grocery list and delete them as in the digital pantry.
UIREQ-13	5	In the Grocery List Page, clicking on the Search button will display the closest store to users to purchase the ingredients.
UIREQ-14	3	In the Search Page, there will be an option to sort ingredients by price, availability, rating, and review.
UIREQ-15	4	In the Search Page, the user can click on the ingredient and find information relating to it on the grocery store site.

Table 9: Table of Grocery List User Interface Requirements

Table 9 lists the User Interface requirements for using the grocery list. There are priority weightings and descriptions for each requirement.

UI Requirements - Find Recipe

Label	Priority Weight	Description
UIREQ-16	5	In the Find Recipe Page, the user can add ingredients that they want or click ingredient in the Digit Pantry and click the Find Recipes button to generate recipes. The user will then be sent to the Recipes page.
UIREQ-17	3	In the Recipes page, the list of recipes displayed can be sorted by recipe name, used ingredients, and missing ingredients.
UIREQ-18	4	In the Recipes page, a user can click on one of the recipes to display its instructions, ingredients, missing ingredients and estimated price range in the Meal Page.

Table 10: Table of Find Recipe User Interface Requirements

Table 10 lists the User Interface requirements for discovering recipes with ingredients. There are priority weightings and descriptions for each requirement.

UI Requirements - Additional Requirements

Label	Priority Weight	Description
UIREQ-19	4	In each page that manipulates ingredients, allow for the user to only input valid ingredients that they can add.
UIREQ-20	4	In the Digital Pantry, allow for users to see a warning when their ingredients are below a certain threshold.
UIREQ-21	3	In the Meal Page, they can click “Shop” to add missing recipe ingredients to their grocery list, sending them to the Grocery List page.

Table 11: Table of Additional User Interface Requirements

Table 10 lists the User Interface requirements that have been added along the way. There are priority weightings and descriptions for each requirement. UIREQ-21 is in red as is not being implemented.

Acceptance Test Cases

Listed below are the acceptance test cases for each requirement.

1. REQ-1
   a. ATC1.01 Ensure a user can switch between the 4 features: Digital Pantry, Recipe Generator, Home Page, and Grocery List.
2. REQ-2
   a. ATC2.01 A user can change their account dietary restrictions, appropriately filtering out recipes shown.
3. REQ-3
   a. ATC3.01 Add and remove ingredients by quantity, weight, volume as well as edit expiry date and verify by visual update.
   b. ATC3.02 Disable removal if there are no ingredients to remove.
   c. ATC3.03 Fail if inputted quantity, weight or volume is not a positive Number or is too large.
4. REQ-4
   a. ATC4.01 Ensure a working network connection to edit ingredients, save them, navigate out of the Digital Pantry and back to verify the persistence of data.
5. REQ-5
   a. ATC5.01 Set an existing ingredient threshold above their current amount and verify that a warning is shown.
6. REQ-6
   a. ATC6.01 Ensure a user can view the ingredient amount by name, quantity, weight, volume, and expiry date
   b. ATC6.02 Notifies the user if there are no ingredients to view.

7. REQ-7
    a. ATC7.01 Manually determine missing ingredients in the Digital Pantry and verify that the generated Grocery List restocks those ingredients above the set threshold when generated.
8. REQ-8
    a. ATC8.01 A user can generate a store to purchase their ingredients.
    b. ATC8.02 Ensure the user can sort and filter the ingredients.
    c. ATC8.03 Notify the user if no stores are generated.
    d. ATC8.04 Fail if no ingredients are in the grocery list.
9. REQ-9
    a. ATC9.01 Add and delete ingredients in the grocery list and verify visual update in the grocery list table of ingredients.
    b. ATC9.02 Fail if the ingredient is not a valid food item.
10. REQ-10
    a. ATC10.01 Input manual ingredients that are not part of the Digital Pantry and generate recipes to ensure those ingredients were used.
11. REQ-11
    a. ATC11.01 A user can generate recipes using their digital pantry.
    b. ATC11.02 Ensure the user can view the recipes
12. REQ-12
    a. ATC12.01 Sort and filter recipes by name, used ingredients and missing ingredients and verify order.
13. REQ-13
    a. ATC13.01 Select a recipe and verify the display of ingredients, instructions, price, and missing ingredients.
14. REQ-14
    a. ATC14.01 Ensure that ingredients added are rejected if invalid.
15. REQ-15
    a. ATC15.01 Add missing recipe ingredients to their grocery list

Functional Requirements Specification

Stakeholders:

The stakeholders listed below are going to be interested in ChefCart:

1. Aspiring chefs such as home cooks, parents, and college students will have direct usage of ChefCart and have interest in using it to manage their pantry, grocery shopping, and meal planning.
2. Grocery Stores have an interest because ChefCart's grocery shopping functions will list them as potential options for home chefs to purchase from, bringing more business to them.
3. Healthcare and insurance companies will have an indirect interest in ChefCart due to the project's positive effect on user health. Users will be able to decrease consumption of fast food and unhealthy takeout by encouraging home cooked meals
4. Environmentalists have a stake in ChefCart because the project will reduce food waste and excess consumption

Actors and Goals:

Initiating Actors

Actor	**Role**	**Goal**
Home Chef (User)	The user of the application. They can add or remove ingredients from their digital pantry, search for new recipes based off of the ingredients in their pantry, and create grocery lists for ingredients they need and/or are missing	The goal for the home chef is to have a stress-free cooking experience when managing the home kitchen.

Table 11: Table of Initiating Actors

Table 11 lists the initiating actors for ChefCart. In our case, there is only one user for ChefCart at a time that will initiate actions. This user is the Home Chef. The table shows the roles and goals each actor has.

Participating Actors

Actor	Role
Database/System	The database system records the user's digital pantry ingredients, dietary restrictions, and account information. The system queries the grocery store APIs to provide the client side with information on the inventory and pricing of ingredients to help the user shop and view different stores. The system also queries the Spoonacular API to provide information and data about ingredients and recipes.

Table 12: Table of Participating Actors

Table 12 lists the participating actors for ChefCart. In our case, there is only one such actor for ChefCart and that would be the database/system. Each participating actor is described with the role it plays. In this regard, the user and system interact with each other and no other actor is involved. It should be noted that external APIs such as Spoonacular and grocery store APIs are consolidated into the one participating actor that is the system as well.

Use Cases

Casual Description

General Usage

UC-1: Signup - Allows individuals to sign up for an account through the webapp
Derivations: UIREQ-1

UC-3: Login - Allows users to login in with their account credentials
Derivations: UIREQ-2, UIREQ-3

UC-10: **Account Editing** - Allows users to view account settings and edit them. The settings include, dietary restrictions, password, and location.
Derivations: REQ-2, UIREQ-6

UC-11: Logout - Allows users to logout of any page and be sent back to the Main screen
Derivations: UIREQ-7

UC-18: Verify Ingredients - Allows the user to verify an ingredient when trying to add it to either the grocery list or digital pantry.
Derivations: REQ-14, UIREQ-19

Digital Pantry

UC-2: Track Pantry - Allows users to view and manipulate ingredients from their digital pantry.
Derivations: REQ-1, REQ-3, REQ-4, REQ-5, REQ-6, REQ-14, UIREQ-2, UIREQ-3, UIREQ-4, UIREQ-8, UIREQ-9, UIREQ-10, UIREQ-11

UC-12: View Pantry Ingredients - Allows the user to view ingredients currently present in the digital pantry.
Derivations: REQ-4, REQ-6

UC-13: Add Pantry Ingredients - Allows the user to add ingredients or add existing ingredient amounts to the digital pantry.
Derivations: REQ-3, REQ-4, REQ-14, UIREQ-8, UIREQ-9, UIREQ-19

UC-14: Remove Pantry Ingredients - Allows the user to remove ingredients or remove existing amounts from the digital pantry.
Derivations: REQ-3, REQ-4, UIREQ-8, UIREQ-10

UC-19: Ingredient Threshold Warning - Allow the user to be warned when the ingredient amount is below a threshold
Derivations: REQ-5, UIREQ-20

Grocery List

UC-4: Create Grocery List - Allows users to generate a grocery list based on missing ingredients in the digital pantry and ingredients they would like to purchase.
Derivations: REQ-1, REQ-7, REQ-9, REQ-14 UIREQ-4, UIREQ-12, UIREQ-3

UC-5: View Grocery Store - Allows users to view the grocery store that has the ingredients for them to purchase and complete their grocery shopping.
Derivations: REQ-8, UIREQ-5, UIREQ-13, UIREQ-15

UC-6: Sort Grocery Store Items- Allows users to sort the list of items from the generated store based off of any set of user-specified preferences such as price, name, ratings, reviews and stock.
Derivations: REQ-8, UIREQ-5, UIREQ-14

UC-15: View List Ingredients - Allows the user to view ingredients currently present in the grocery list.
Derivations: UIREQ-12

UC-16: Add List Ingredients - Allows the user to add ingredients or add existing ingredient amounts to the grocery list.
Derivations: REQ-9, REQ-14, UIREQ-12, UIREQ-19

UC-17: Remove List Ingredients - Allows the user to remove ingredients from the grocery list or remove existing ingredient amounts.
Derivations: REQ-9, UIREQ-12

Recipe Generator

UC-7: Discover Recipes - Allows users to view a list of generated recipes based off of ingredients available in their pantry as well as any manually-inputted items
Derivations: REQ-1, REQ-10, REQ-11, REQ-12, REQ-14, UIREQ-4, UIREQ-16, UIREQ-19, UIREQ-3

UC-8: Sort/Filter Recipes - Allows users to sort and filter the list of generated recipes based off of any set of user-specified dietary preferences, name, used ingredients, and missing ingredients.
Derivations: REQ-12, UIREQ-5, UIREQ-17

UC-9: Viewing Recipe - Allows users to view the recipe instructions and other relevant information such as required ingredients, price, prep time, etc.
Derivations: REQ-13, UIREQ-5, UIREQ-18

UC-20: Shop Missing Recipe Ingredients - Allow the user to add missing recipe ingredients to their grocery list after viewing a recipe.
Derivations: REQ-15, UIREQ-21

UC-21: Show Missing Recipe Ingredients - Allow the user to be shown the missing ingredients that they do not have for a recipe
Derivations: UIREQ-18

Use Case Diagram

Use Case Diagram

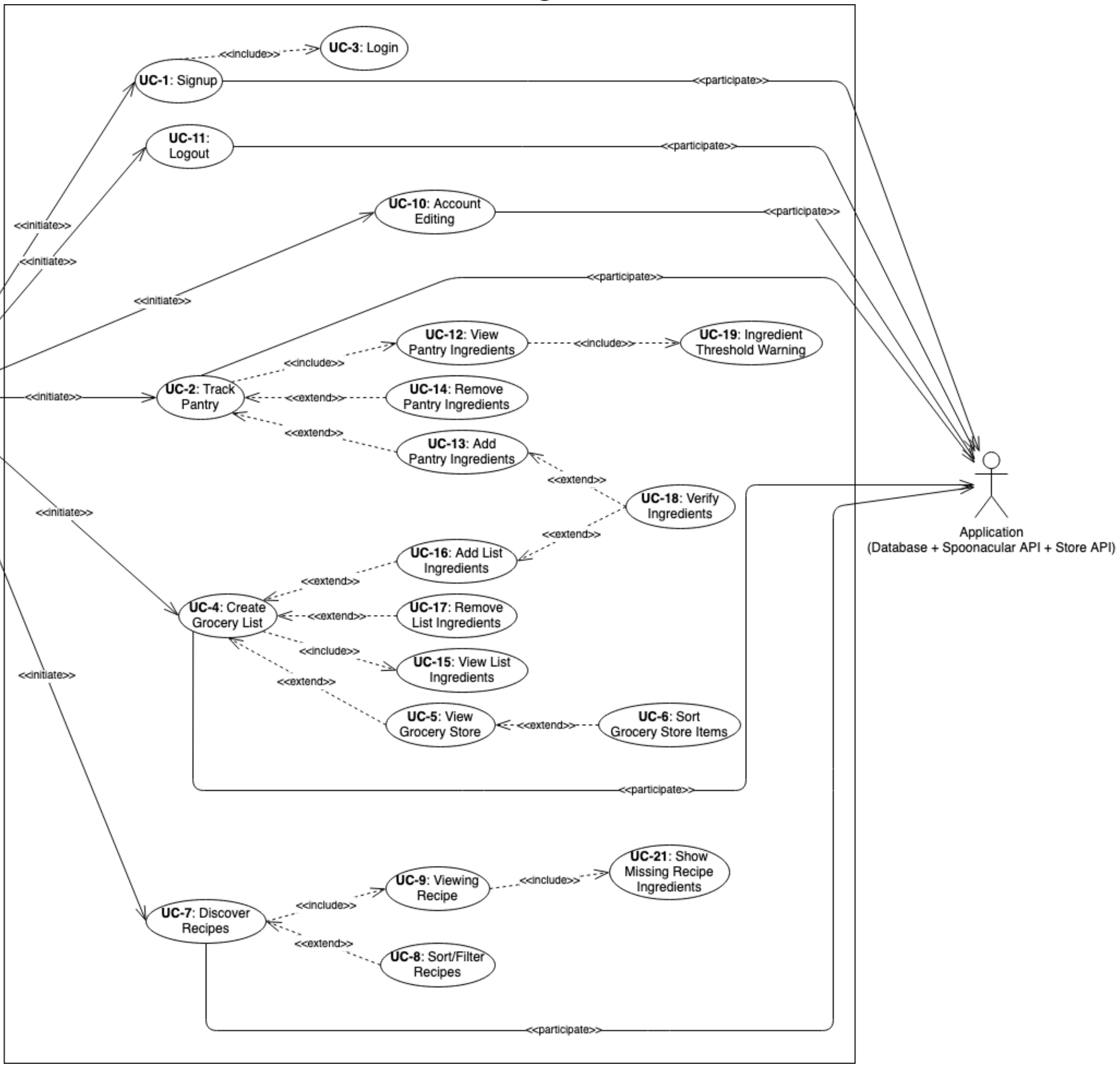

UC-1: Signup
UC-3: Login
<<include>>
UC-11: Logout
UC-10: Account Editing
UC-2: Track Pantry
UC-12: View Pantry Ingredients
UC-19: Ingredient Threshold Warning
UC-14: Remove Pantry Ingredients
UC-13: Add Pantry Ingredients
UC-18: Verify Ingredients
UC-16: Add List Ingredients
UC-4: Create Grocery List
UC-17: Remove List Ingredients
UC-15: View List Ingredients
UC-5: View Grocery Store
UC-6: Sort Grocery Store Items
UC-7: Discover Recipes
UC-9: Viewing Recipe
UC-21: Show Missing Recipe Ingredients
UC-8: Sort/Filter Recipes
<<initiate>>
<<participate>>
<<extend>>
User
Application (Database + Spoonacular API + Store API)

Figure 2: Image for Use Case Diagram

The use case diagram has an initiating actor (**user**) and a participating actor (**application**). The base cases are shown to be directly initiated by the **user**. Subroutines are related to their parent routines through an “**extends”** relationship if the subroutine MAY occur during a parent routine, and through an “**includes”** relationship if the subroutine ALWAYS occurs during a parent routine.

Note: We didn’t connect **Login** to the **user** because we were advised not to connect any subroutine to an initiating actor (**Login** is an **included** subroutine of **Signup**).

Traceability Matrix

Traceability Matrix

Req't	PW	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
REQ-1	3		X		X			X														
REQ-2	2			X							X											
REQ-3	5		X	X										X	X							
REQ-4	5		X										X	X	X							
REQ-5	4		X																	X		
REQ-6	4		X										X									
REQ-7	3				X																	
REQ-8	4					X	X															
REQ-9	3				X												X	X				
REQ-10	3							X														
REQ-11	5							X														
REQ-12	4							X	X													
REQ-13	4									X												
REQ-14	4		X		X			X						X			X		X			
REQ-15	4																				X	
UIREQ-1	3	X																				
UIREQ-2	3		X																			
UIREQ-3	5		X		X			X														
UIREQ-4	5		X		X			X														
UIREQ-5	4					X	X		X	X												
UIREQ-6	3										X											
UIREQ-7	4											X										
UIREQ-8	5		X											X	X							
UIREQ-9	5		X											X								
UIREQ-10	5		X												X							

UIREQ-11	5		X																			
UIREQ-12	4				X											X	X	X				
UIREQ-13	5					X																
UIREQ-14	3						X															
UIREQ-15	4					X																
UIREQ-16	5							X														
UIREQ-17	3								X													
UIREQ-18	4									X												X
UIREQ-19	4							X						X			X		X			
UIREQ-20	4																			X		
UIREQ-21	3																				X	
MAX PW		3	5	5	5	5	4	5	4	4	3	4	5	5	5	4	4	4	4	4	4	4
TOTAL PW		3	53	7	27	17	11	38	11	12	5	4	9	28	24	4	15	7	8	8	8	4

Table 13: Table for traceability matrix of Requirements and Use Cases

In Table 13 requirements are mapped to each use case (Use cases on the columns) in this traceability matrix. From the table above, we can see that each use case is mapped to at least one requirement, so every case is accounted for. We also weighed the priority of each use case and total them in the last row.

Fully Dressed Description

We created fully dressed use case descriptions to flesh out their functions and performance. We considered the most important use cases to be UC-2, UC-4, UC-5, and UC-7 for tracking the pantry, making a grocery list, viewing store items and finding relevant recipes.

→ This symbol will be used to denote when the user performs an action.
← This symbol will be used to denote when the system performs an action.

UC-2: Track Pantry
Related Requirements: ● REQ-1 ● REQ-3 ● REQ-4 ● REQ-5 ● REQ-6 ● REQ-14 ● UIREQ-2 ● UIREQ-3 ● UIREQ-4 ● UIREQ-8 ● UIREQ-9 ● UIREQ-10 ● UIREQ-11
Initiating Actor: ● Home Chef
Actor's Goal: ● To track ingredients in the user's physical pantry.
Participating Actors: ● Database / System
Preconditions: ● The user has logged in and has navigated to the Digital Pantry tab.
Minimal Guarantees: ● The application will provide the user the ability to manually update their food stock in the digital pantry. Each field (ingredient: string, quantity: int, weight: float, volume: float, expiry: date, threshold to be quantity, weight or volume) will be implemented such that there is no possibility for input error (i.e. for number fields, you use a number picker, for dates you use a date picker, and for ingredient names, you can search/select from a list).

Success Guarantees:

- The application will update the corresponding ingredients with their quantity, volume or weight, and expiration date.

Plan of Action for Main Success Scenario:
Adding a new ingredient

1. → The user enters a new ingredient or updates an existing ingredient by inputting the fields for ingredient quantity, volume, weight, threshold and expiry date in a popup.
2. → If the user cancels the addition of the ingredient, nothing will be changed and the user will be returned to the digital pantry precondition state, ending the use case.
3. → The user confirms the addition of this ingredient.
4. ← The system verifies this ingredient with the Spoonacular API.
5. ← If the ingredient is invalid, the use case ends and the ingredient is not added.
6. ← The ingredient is verified and valid, so the system updates the database with the new ingredient attributes accordingly.
7. → The new ingredient fields are displayed back to the user in the Digital Pantry.

Plan of Action for Alternate Scenarios or Contingencies:
Editing/Updating ingredients

1. → The user selects an ingredient field for editing.
2. → If the user unselects the field, editing will cancel and the unedited Digital Pantry information will be redisplayed, returning them to the precondition state and the use case ends.
3. → The user inputs their new ingredient value for either of quantity, volume, weight, threshold and expiration date.
4. → The user confirms the edit.
5. ← The system updates the ingredient in the database and redisplays the updated table.

Deleting Ingredients

1. → The user clicks on the trash icon to delete the corresponding ingredient.
2. ← The system asks the user for confirmation of the delete with the popup.
3. → If the user cancels the delete, the user is returned to the precondition state and the use case ends.
4. → The user confirms the delete.
5. ← The system will delete the corresponding ingredient in the database and redisplays the updated table to the user.

UC-4: Create Grocery List

Related Requirements:

- REQ-1
- REQ-7
- REQ-9
- REQ-14

- UIREQ-4
- UIREQ-12
- UIREQ-3

Initiating Actor:
- Home Chef

Actor’s Goal:
- Generate a grocery list based on missing ingredients in the digital pantry and ingredients they would like to purchase.

Participating Actors:
- Database / System

Preconditions:
- The user has launched the ChefCart app.
- The user has successfully logged into ChefCart.
- After logging in, the user has navigated to the Grocery List Tab.

Minimal Guarantees:
- Same as success guarantees: no failure involved.

Success Guarantees:
- The grocery list will reflect the user’s desired changes.

Plan of Action for Main Success Scenario:
1. ← The system generates a grocery list based on the ingredients under a user set amount threshold in the digital pantry.
2. ← The grocery list is displayed to the user, allowing for further addition, deletion and editing of ingredients.

Plan of Action for Alternate Scenarios or Contingencies:

a. Deleting Ingredients
   1. → The user presses the trash can icon to delete the relevant ingredient.
   2. ← The system asks for confirmation of the delete action with a popup.
   3. → If the user cancels the delete, the ingredient will not be deleted and the Grocery List page will be redisplayed and the use case ends.
   4. → The user confirms the delete.
   5. ← The grocery list refreshes itself with updated changes and is redisplayed.

b. Editing Ingredients
   1. → The user clicks on the field of an existing ingredient.
   2. ← The system provides the user with a picker or text field input for the user to input the new value.
   3. → If the user cancels the edit by unselecting the field input, the grocery list is redisplayed and the use case ends.
   4. → The user inputs their new value.
   5. ← The system updates the value and redisplays the updated information to the

user. c. Adding Ingredients 1. → The user presses the *Add Ingredient* button to open up an input window 2. → If the user cancels the add, the grocery list is redisplayed and the use case ends. 3. ← The application opens up an ingredient searcher through the Spoonacular API. 4. → The user searches for the desired ingredient. 5. → The user finds the desired ingredient. 6. → The user adds the relevant quantity of the ingredient to the grocery list. 7. ← The ingredient searcher closes, displaying an updated grocery list.

UC-5: View Grocery Store
Related Requirements: ● REQ-8 ● UIREQ-5 ● UIREQ-13 ● UIREQ-15
Initiating Actor: ● Home Chef
Actor's Goal: ● To find a suitable store to purchase ingredients from the grocery list.
Participating Actors: ● Database / System
Preconditions: ● The user has launched and logged into ChefCart ● The user was on the grocery list page, generated a grocery list and clicked on the "Choose Store" button.
Minimal Guarantees: ● There is no failure involved, it will be the same as success guarantees.
Success Guarantees: ● The displayed stores, if any, will allow users to click on them to view their information in the Store Page.
Plan of Action for Main Success Scenario: 1. ← The system searches for stores using the stores APIs available. 2. ← The system displays the closest grocery store and the relevant items.

3. → The user chooses to sort items by name, price, ratings, reviews or stock.
4. → The user clicks on an item.
5. ← The user is taken to the grocery store link to view the item and potentially purchase it online.

Plan of Action for Alternate Scenarios or Contingencies:

1. N/A

UC-7: Discover Recipes

Related Requirements:

- REQ-1
- REQ-10
- REQ-11
- REQ-12
- REQ-14
- UIREQ-4
- UIREQ-16
- UIREQ-19
- UIREQ-3

Initiating Actor:

- Home Chef

Actor's Goal:

- To find suitable recipes to cook at home.

Participating Actors:

- Database / System

Preconditions:

- The user has launched and logged into ChefCart
- The user has navigated to the Recipe Generator page.

Minimal Guarantees:

- No failure involved, this will be the same as success guarantees.
- The system will only allow correct user input when inputting ingredients. An ingredient searcher will be used to properly help the user select an ingredient.

Success Guarantees:

- The system will display any found recipes to the user.

Plan of Action for Main Success Scenario:

1. → The user chooses to use Digital Pantry items by checking each item off they want to

use. 2. → The user clicks on the “Find Recipes” button. 3. ← The application will call the Spoonacular API to generate recipes based on the ingredients input and the user’s account dietary restrictions, displaying them. 4. ← The system sends the user to the Recipes Page and displays the generated recipes to the user. 5. → The user opts to sort each recipe by name, used ingredients, and missing ingredients.
Plan of Action for Alternate Scenarios or Contingencies: Adding manual ingredients 1. N/A

System Sequence Diagrams

System Sequence Diagram - Signup

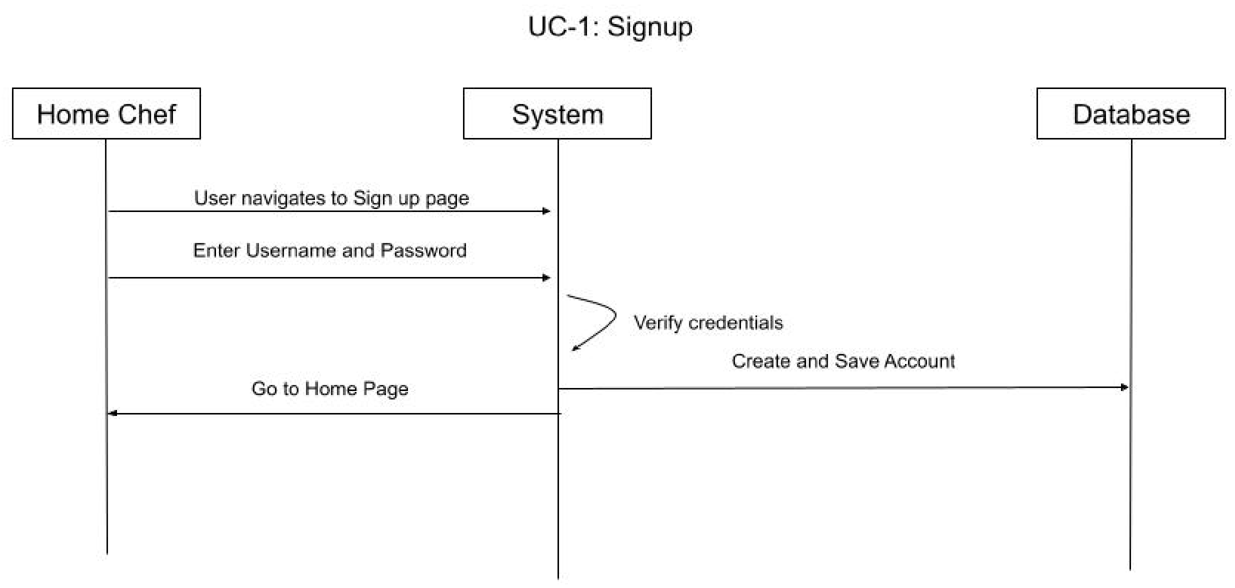


Figure 20: Signup System Sequence Diagram

Figure 20 is the system sequence diagram for signing up a new account. The user needs to enter a new username and password. The system will verify their validity. If valid, the database will create the account and save it. Then, the user will be navigated back to the Home Page.

System Sequence Diagram - Track Pantry

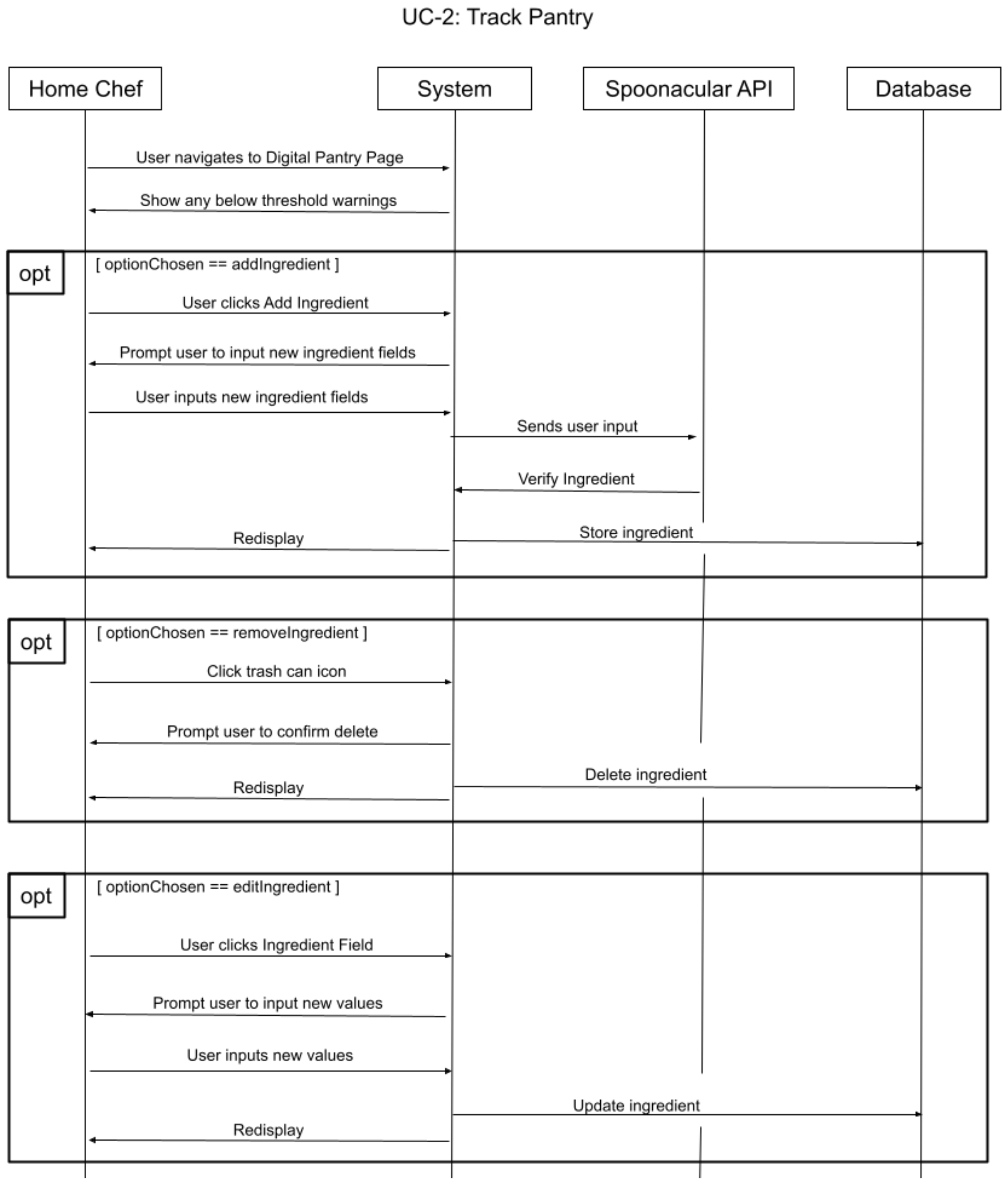

UC-2: Track Pantry
Home Chef
System
Spoonacular API
Database
User navigates to Digital Pantry Page
Show any below threshold warnings
opt
[optionChosen == addIngredient]
User clicks Add Ingredient
Prompt user to input new ingredient fields
User inputs new ingredient fields
Sends user input
Verify Ingredient
Redisplay
Store ingredient
opt
[optionChosen == removeIngredient]
Click trash can icon
Prompt user to confirm delete
Delete ingredient
Redisplay
opt
[optionChosen == editIngredient]
User clicks Ingredient Field
Prompt user to input new values
User inputs new values
Update ingredient
Redisplay

Figure 3: Track Pantry System Sequence Diagram

Figure 3 is the system sequence diagram for tracking a user pantry. Note that the user may cancel the addition, editing, and deleting of ingredients. To track the pantry, the user first navigates to the Digital Pantry Page which will immediately warn the user if there are ingredients that are low. Then, the user can choose from 3 options to add, remove or edit ingredients. By inputting the desired updated ingredient information, the database saves it and redisplays the corresponding updates to the user.

System Sequence Diagram - Login

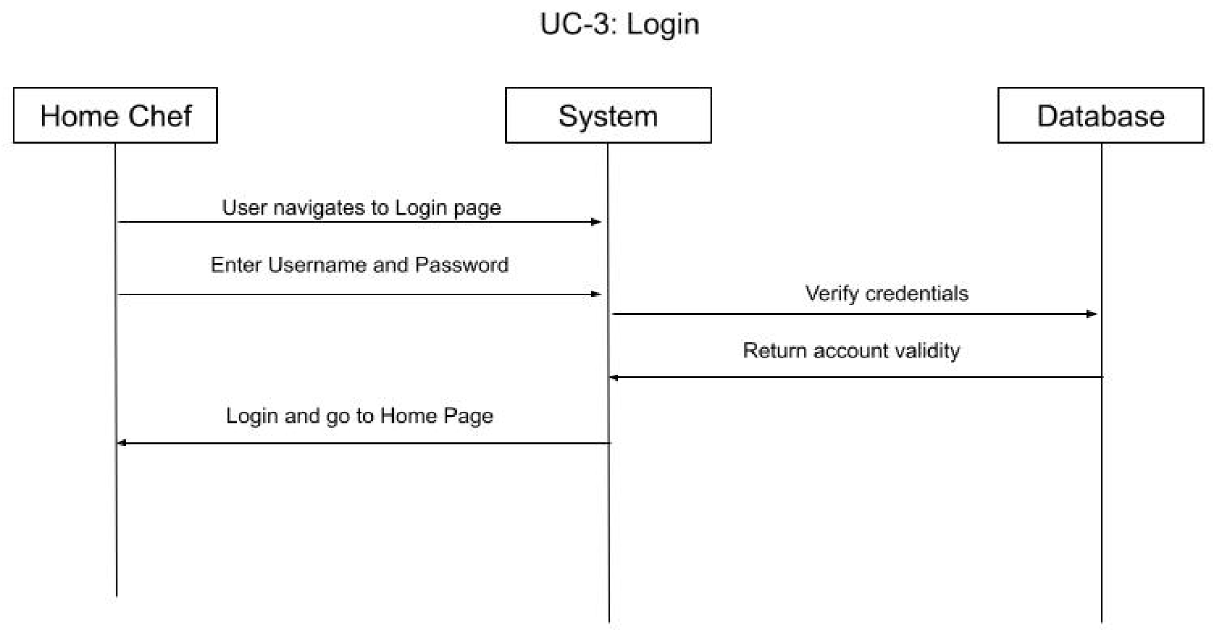

UC-3: Login
Home Chef
System
Database
User navigates to Login page
Enter Username and Password
Verify credentials
Return account validity
Login and go to Home Page

Figure 21: Login System Sequence Diagram

Figure 21 is the system sequence diagram for logging in. In the login page, the user enters their username and password. The system will ask the database to verify that the account credentials are valid. If valid, the user will be able to login and will go to the Home Page.

System Sequence Diagram - Create Grocery List

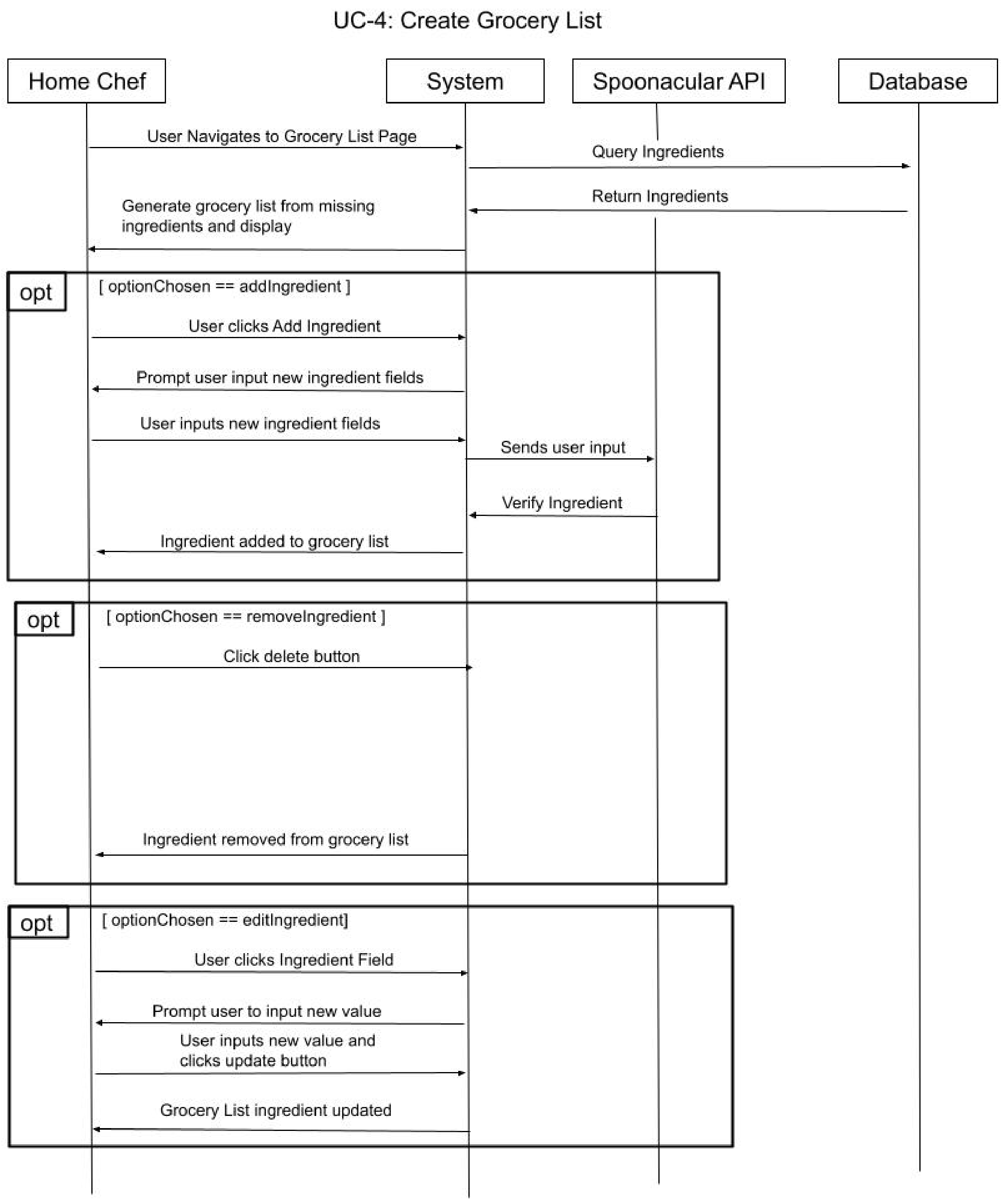

UC-4: Create Grocery List
Home Chef
System
Spoonacular API
Database
User Navigates to Grocery List Page
Query Ingredients
Return Ingredients
Generate grocery list from missing ingredients and display
opt
[optionChosen == addIngredient]
User clicks Add Ingredient
Prompt user input new ingredient fields
User inputs new ingredient fields
Sends user input
Verify Ingredient
Ingredient added to grocery list
opt
[optionChosen == removeIngredient]
Click delete button
Ingredient removed from grocery list
opt
[optionChosen == editIngredient]
User clicks Ingredient Field
Prompt user to input new value
User inputs new value and clicks update button
Grocery List ingredient updated

Figure 4: Create Grocery List System Sequence Diagram

Figure 4 is the system sequence diagram for creating a new grocery list. Note that the user may cancel the addition, editing, and deleting of ingredients. To create a grocery list, the user first navigates to the Grocery List Page. The system will then query the database for the missing ingredients that the user normally has. As in the digital pantry, the user may add, remove or edit ingredients in the generated grocery list.

System Sequence Diagram - View Grocery Store

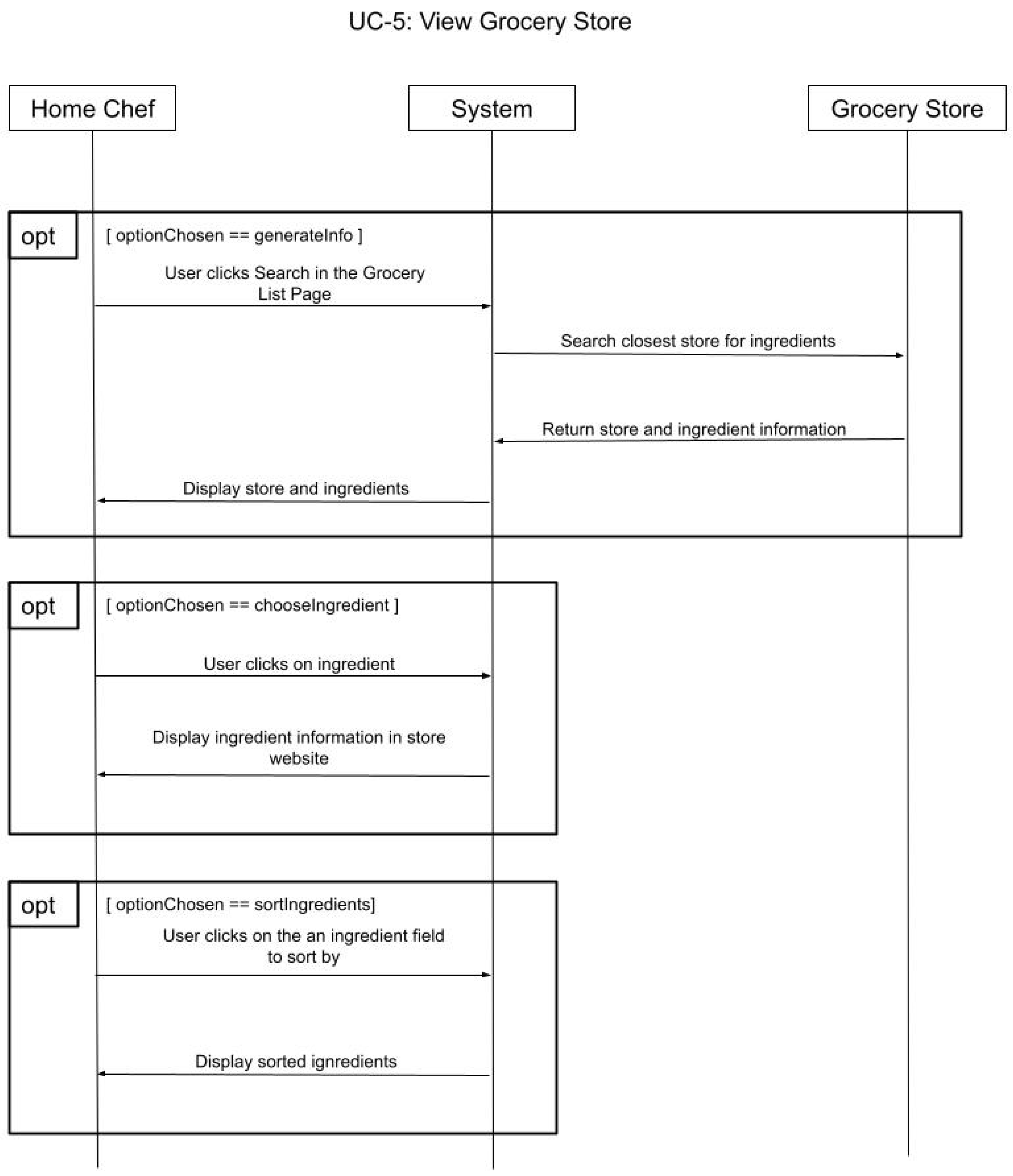

UC-5: View Grocery Store
Home Chef
System
Grocery Store
opt
[optionChosen == generateInfo]
User clicks Search in the Grocery List Page
Search closest store for ingredients
Return store and ingredient information
Display store and ingredients
opt
[optionChosen == chooseIngredient]
User clicks on ingredient
Display ingredient information in store website
opt
[optionChosen == sortIngredients]
User clicks on the an ingredient field to sort by
Display sorted ignredients

Figure 5: View Grocery Store System Sequence Diagram

Figure 5 is the system sequence diagram to view a grocery store and its ingredients to purchase. To find a store, the user must have generated a grocery list and clicked on the Search button. We comb grocery store APIs to search for the closest store with the right ingredients and return them to display to the user. Then, the user will have the option to sort the ingredients listed and click on one to see its information such as total price and store location.

System Sequence Diagram - Sort Grocery Store Items

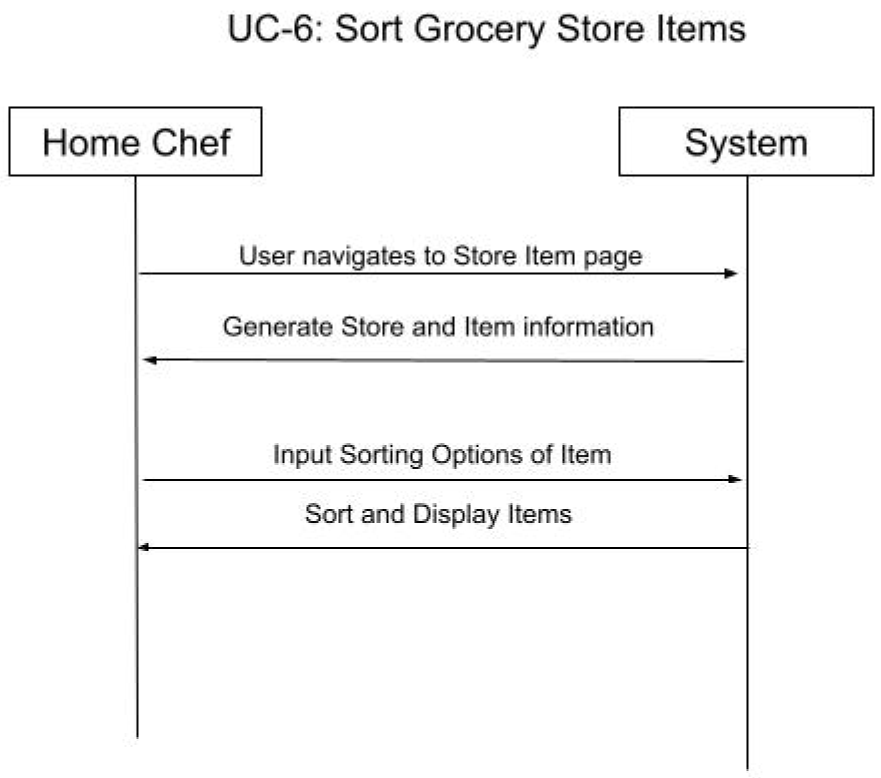

UC-6: Sort Grocery Store Items
Home Chef
System
User navigates to Store Item page
Generate Store and Item information
Input Sorting Options of Item
Sort and Display Items

Figure 22: Sort Grocery Store Items System Sequence Diagram

Figure 22 is the system sequence diagram for sorting grocery store items. After generating the grocery list, the user navigates to find a store. The closest store is returned to them and its information is displayed to the user. Items that the store sells that are relevant to the user are also displayed. The user can then sort the items by name, price, ratings, reviews and availability. After sorting, the system will redisplay the sorted items.

System Sequence Diagram - Discover Recipes

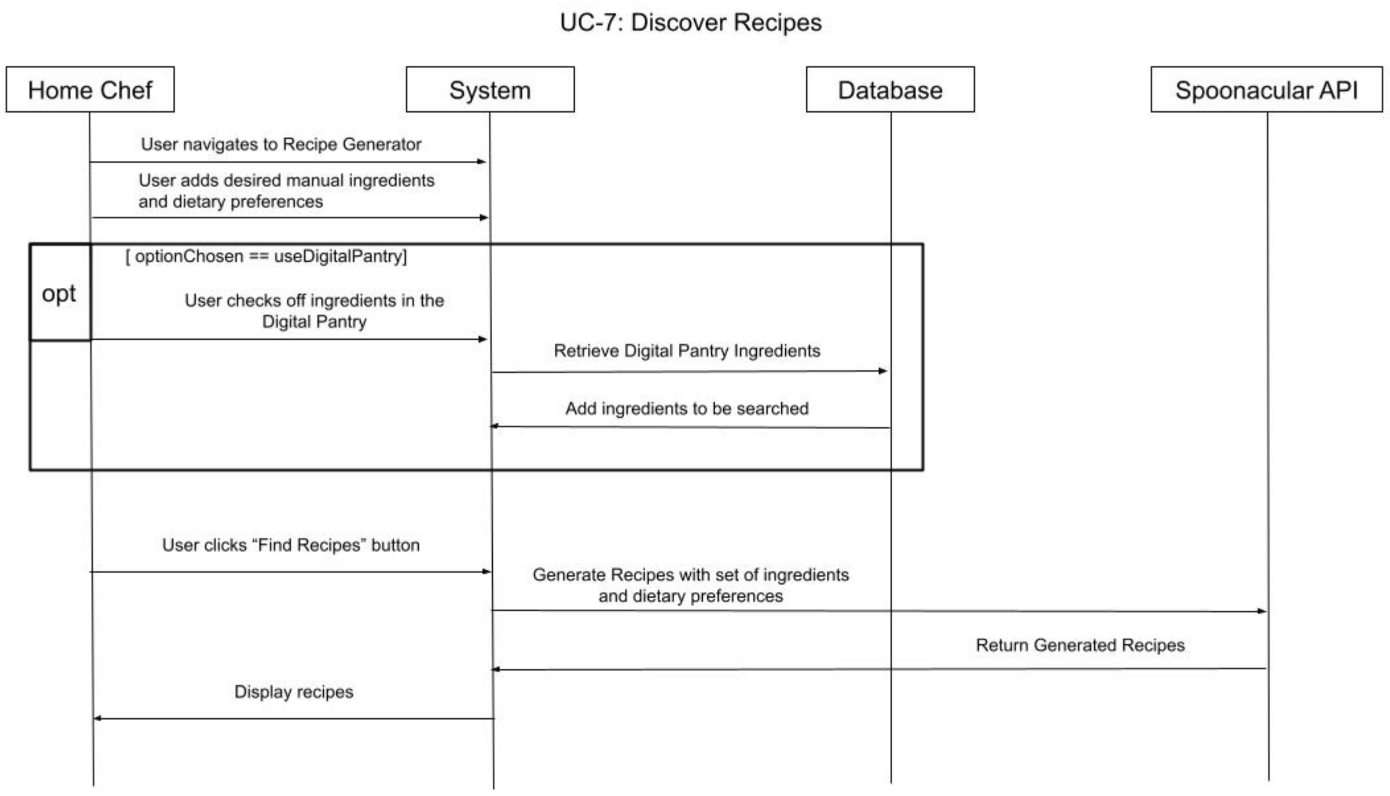

UC-7: Discover Recipes
Home Chef
System
Database
Spoonacular API
User navigates to Recipe Generator
User adds desired manual ingredients and dietary preferences
opt
[optionChosen == useDigitalPantry]
User checks off ingredients in the Digital Pantry
Retrieve Digital Pantry Ingredients
Add ingredients to be searched
User clicks "Find Recipes" button
Generate Recipes with set of ingredients and dietary preferences
Return Generated Recipes
Display recipes

Figure 6: Discover Recipes System Sequence Diagram

Figure 6 is the system sequence diagram for finding recipes. To discover recipes, the user first must navigate to the Recipe Generator page. The user may choose to add manual ingredients or opt to use ingredients in the digital pantry in their search for a recipe. Once they click the submit button, the Spoonacular will help generate the recipes with the set of ingredients and display them to the user

System Sequence Diagram - Sort/Filter Recipes

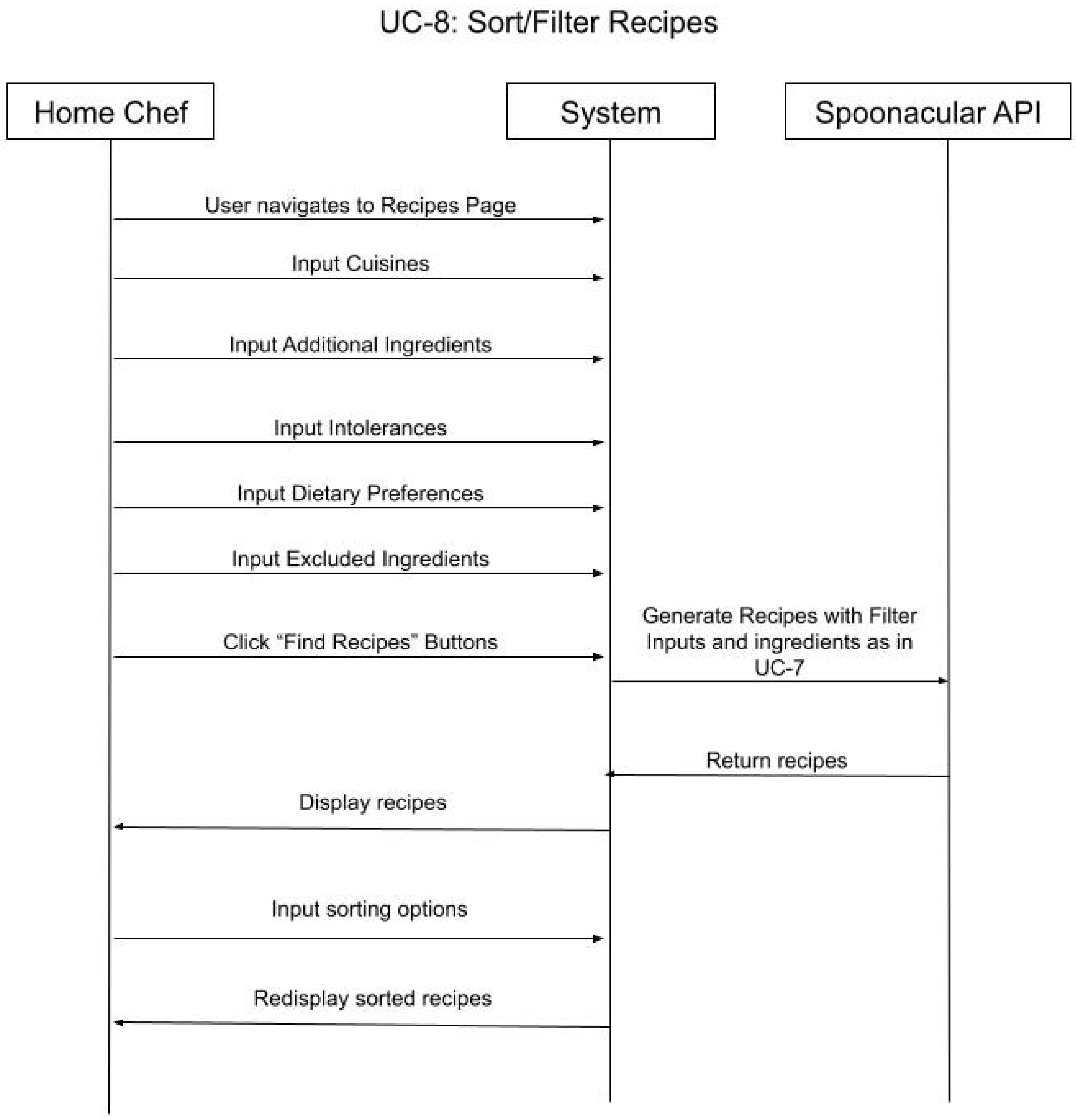

UC-8: Sort/Filter Recipes
Home Chef
System
Spoonacular API
User navigates to Recipes Page
Input Cuisines
Input Additional Ingredients
Input Intolerances
Input Dietary Preferences
Input Excluded Ingredients
Click "Find Recipes" Buttons
Generate Recipes with Filter Inputs and ingredients as in UC-7
Return recipes
Display recipes
Input sorting options
Redisplay sorted recipes

Figure 23: Sort/Filter Recipes System Sequence Diagram

Figure 23 is the system sequence diagram for sorting and filtering recipes. This is a two step process. In the recipes page, the user first enters the filters. They can filter what ingredients they want to include, exclude, their dietary preferences, intolerances and even cuisines. These filters are passed into the Spoonacular API to return the recipes. These recipes are displayed to the user and the user may choose to sort them. The user can sort the recipes by name, included ingredients and missing ingredients. The system will perform the sorting and redisplay to the user.

System Sequence Diagram - Viewing Recipe

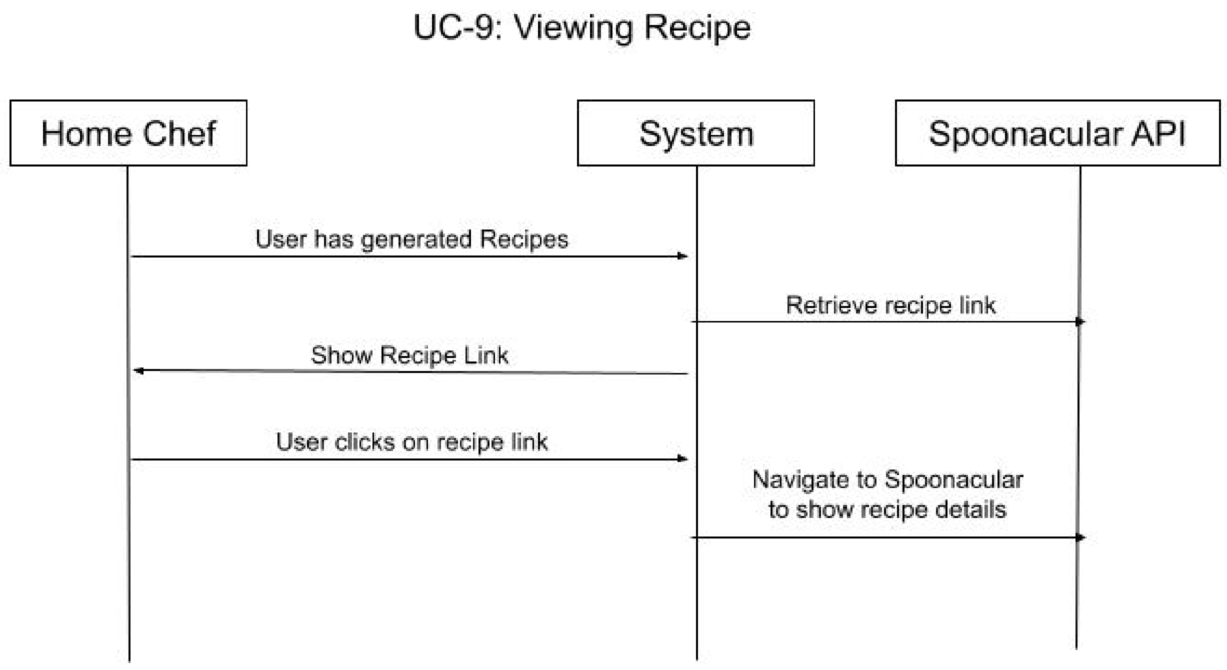

UC-9: Viewing Recipe
Home Chef
System
Spoonacular API
User has generated Recipes
Retrieve recipe link
Show Recipe Link
User clicks on recipe link
Navigate to Spoonacular to show recipe details

Figure 24: Viewing Recipe System Sequence Diagram

Figure 24 is the system sequence diagram for viewing a recipe. The user first must have generated recipes. The system gets the recipe link from Spoonacular and shows it to the user. When the user clicks on the link, they are sent to the Spoonacular website to view the recipe details.

System Sequence Diagram - Account Editing

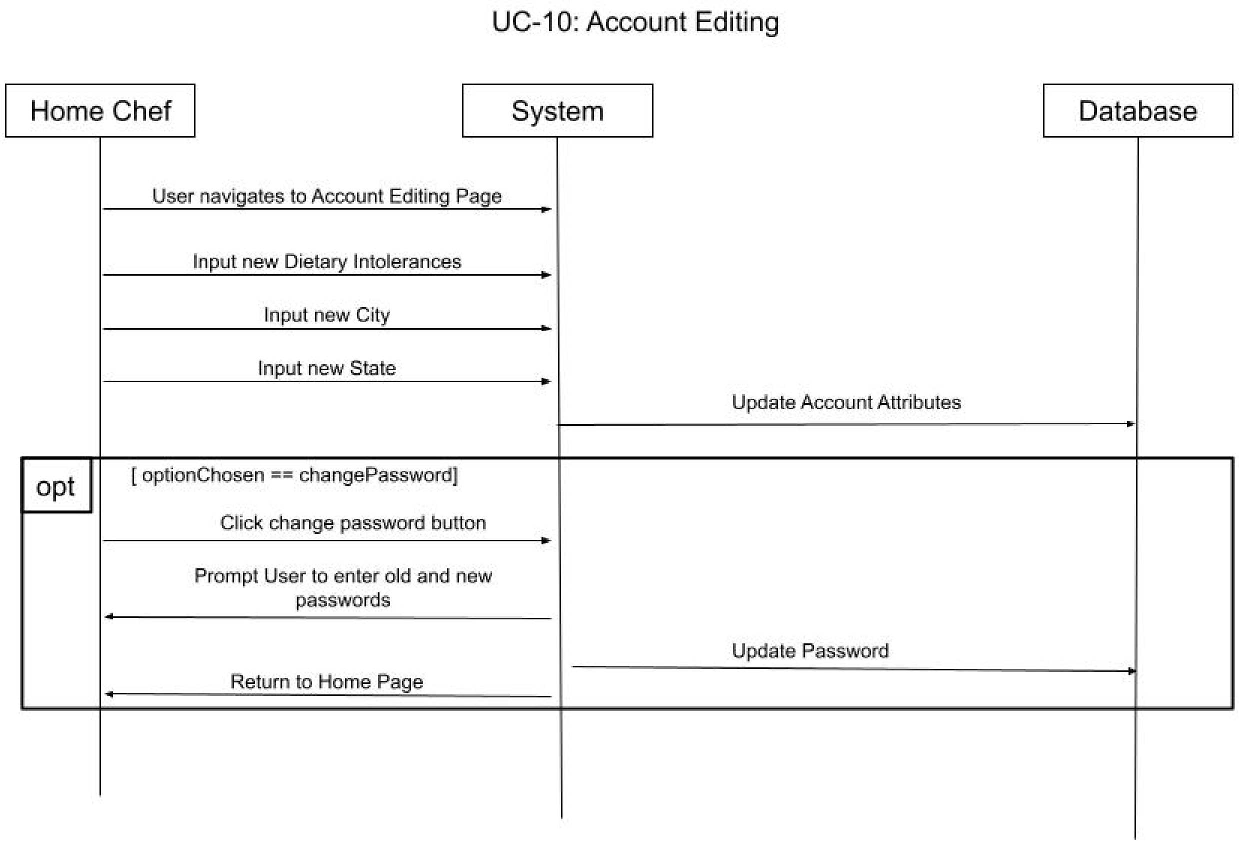

UC-10: Account Editing
Home Chef
System
Database
User navigates to Account Editing Page
Input new Dietary Intolerances
Input new City
Input new State
Update Account Attributes
opt
[optionChosen == changePassword]
Click change password button
Prompt User to enter old and new passwords
Update Password
Return to Home Page

Figure 25: Account Editing System Sequence Diagram

Figure 25 is the system sequence diagram for editing accounts. The user goes to the Account Editing page and can enter their new dietary intolerances, new city and state as needed. Upon confirmation, the database will update these attributes for them and save them. The user also has the option to change their password. They will be prompted to enter their old and new passwords to be updated in the database. Once changes are complete, the user will be returned to the home page.

System Sequence Diagram - Logout

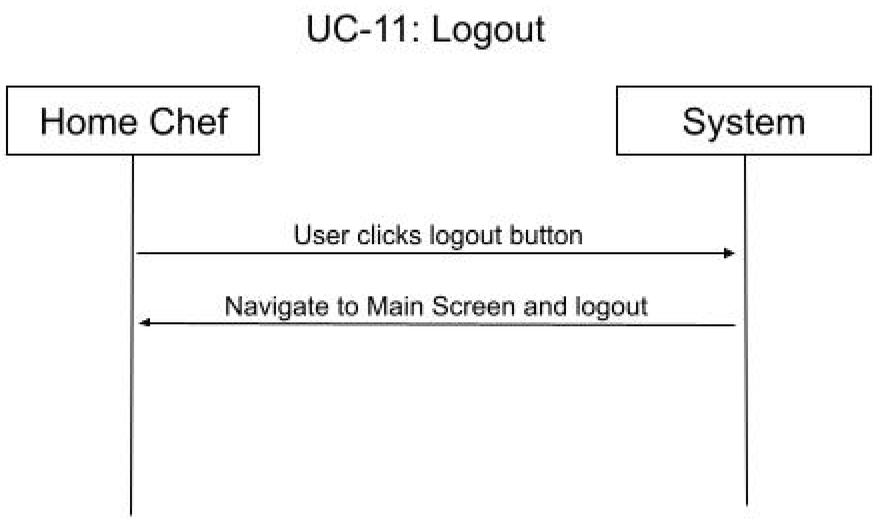


Figure 26: Logout System Sequence Diagram

Figure 26 is the system sequence diagram for logging out. The user simply clicks the logout button and they are logged out and returned to the main screen.

System Sequence Diagram - View Pantry Ingredients

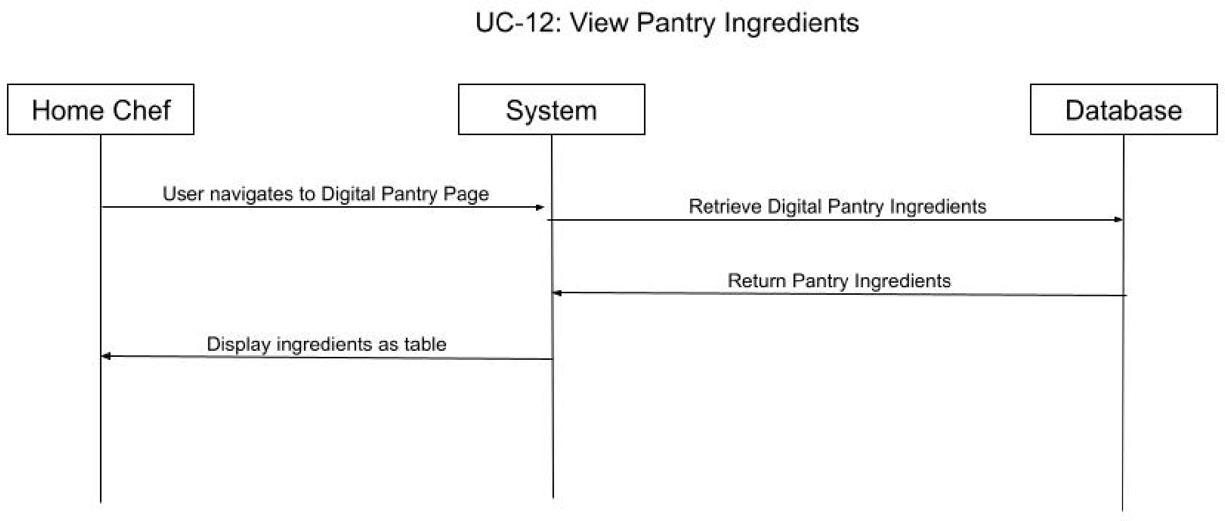

UC-12: View Pantry Ingredients
Home Chef
System
Database
User navigates to Digital Pantry Page
Retrieve Digital Pantry Ingredients
Return Pantry Ingredients
Display ingredients as table

Figure 27: View Pantry Ingredients System Sequence Diagram

Figure 27 is the system sequence diagram for viewing pantry ingredients. The user goes to the Digital Pantry Page and the system retrieves the ingredients from the database. The ingredients are listed in a tabular fashion with the ingredient name, quantity, weight, volume and expiration date.

System Sequence Diagram - Add Pantry Ingredients

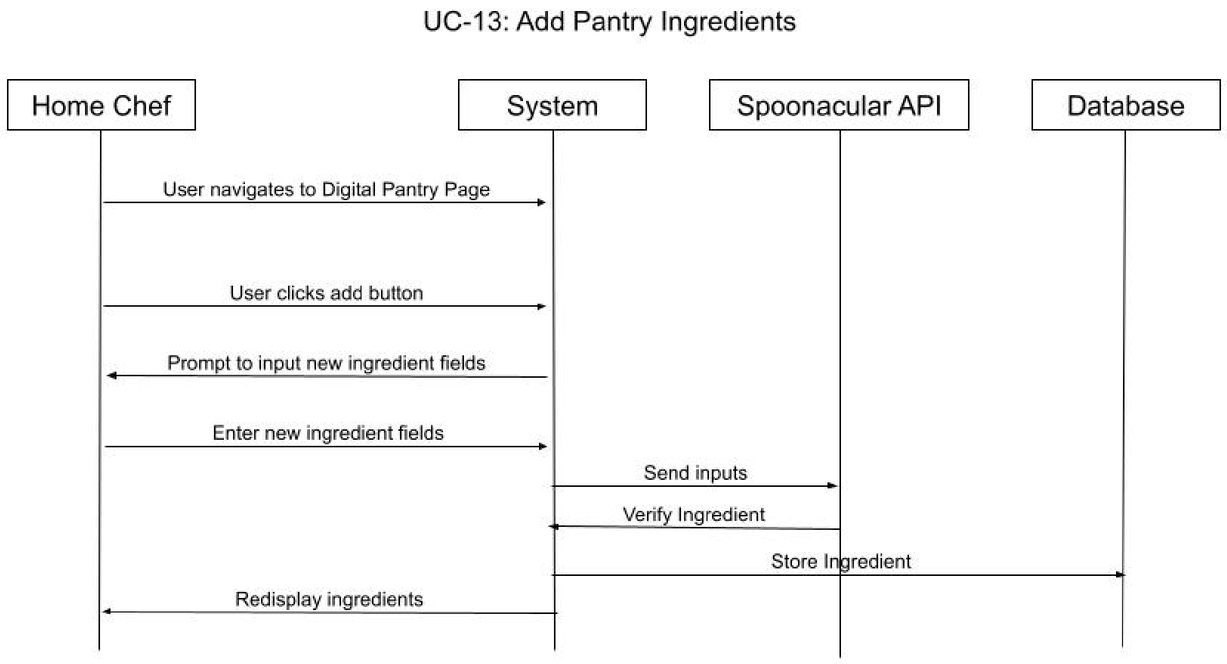

UC-13: Add Pantry Ingredients
Home Chef
System
Spoonacular API
Database
User navigates to Digital Pantry Page
User clicks add button
Prompt to input new ingredient fields
Enter new ingredient fields
Send inputs
Verify Ingredient
Store Ingredient
Redisplay ingredients

Figure 28: Add Pantry Ingredients System Sequence Diagram

Figure 28 is the system sequence diagram for adding pantry ingredients. In the Digital Pantry Page, the user clicks the add button and is prompted to enter new ingredient fields. These include the ingredient name, quantity, units, expiration date and warning thresholds. The ingredient is verified to be valid by the Spoonacular API. If valid, the database will store the ingredient and the Digital Pantry will be updated and redisplayed to the user.

System Sequence Diagram - Remove Pantry Ingredients

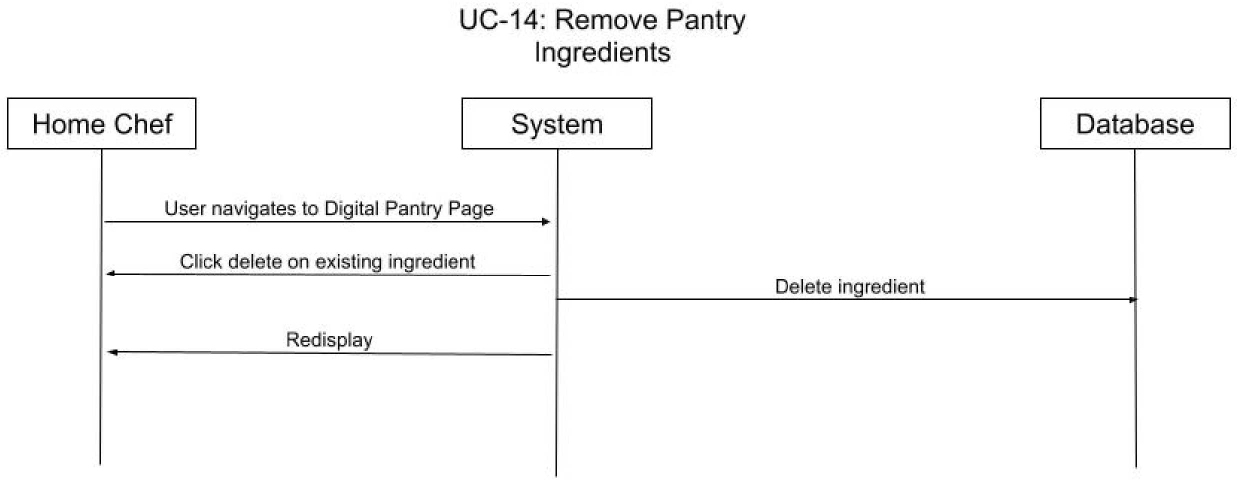

UC-14: Remove Pantry Ingredients
Home Chef
System
Database
User navigates to Digital Pantry Page
Click delete on existing ingredient
Delete ingredient
Redisplay

Figure 29: Remove Pantry Ingredients System Sequence Diagram

Figure 29 is the system sequence diagram for removing pantry ingredients. The user clicks on an existing ingredient in their Digital Pantry. The database will delete the ingredient and the system will redisplay the change to the user.

System Sequence Diagram - View List Ingredients

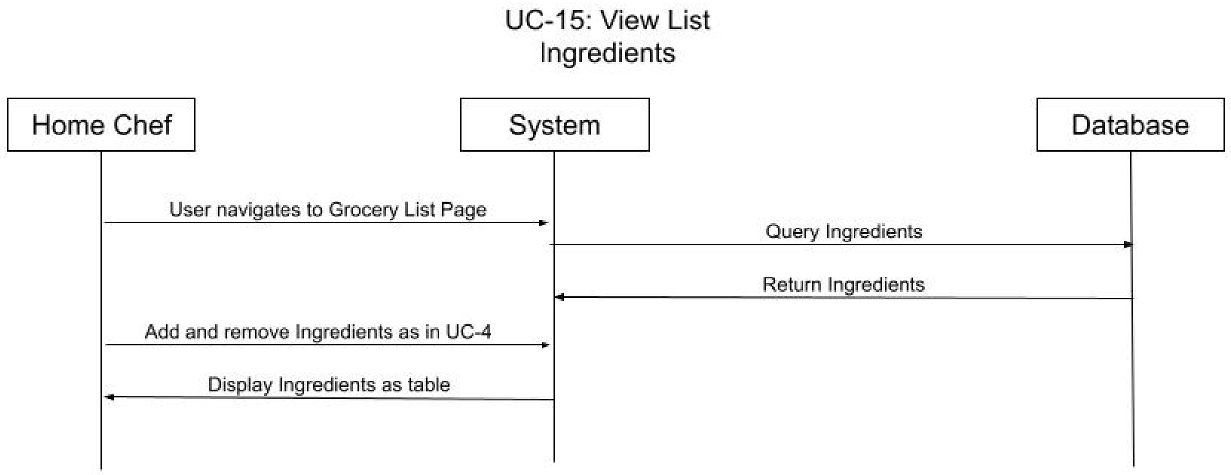

UC-15: View List Ingredients
Home Chef
System
Database
User navigates to Grocery List Page
Query Ingredients
Return Ingredients
Add and remove Ingredients as in UC-4
Display Ingredients as table

Figure 30: View List Ingredients System Sequence Diagram

Figure 30 is the system sequence diagram for viewing grocery list ingredients. In the Grocery List Page, the system will query the database for the Digital Pantry ingredients information. Ingredients are automatically generated to fill the threshold set by the user. The user can then add and remove additional ingredients as desired. The list of ingredients are displayed to the user with their names.

System Sequence Diagram - Add List Ingredients

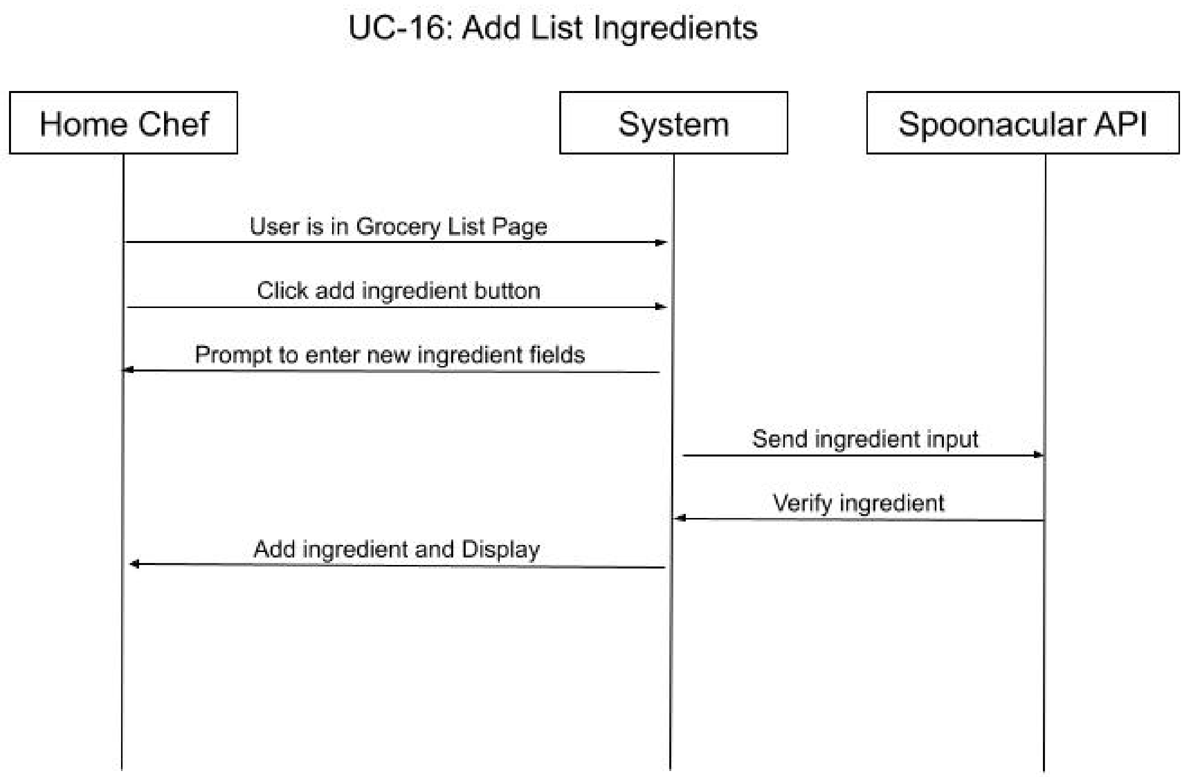

UC-16: Add List Ingredients
Home Chef
System
Spoonacular API
User is in Grocery List Page
Click add ingredient button
Prompt to enter new ingredient fields
Send ingredient input
Verify ingredient
Add ingredient and Display

Figure 31: Add List Ingredients System Sequence Diagram

Figure 31 is the system sequence diagram for adding grocery list ingredients. The user clicks the add button and is asked to enter ingredient fields. Upon adding, the ingredient is verified to be valid by Spoonacular. If valid, the ingredient will be added to the grocery list and displayed again.

System Sequence Diagram - Remove List Ingredients

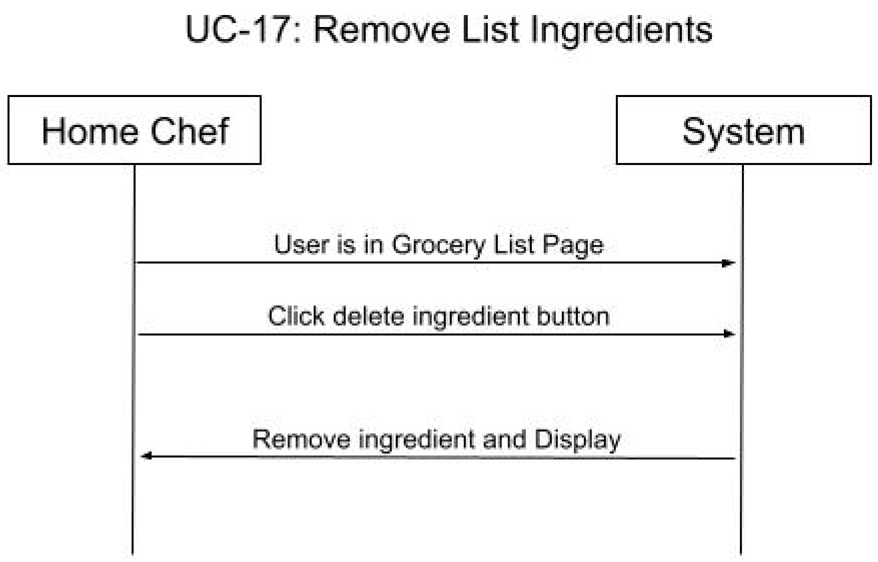

UC-17: Remove List Ingredients
Home Chef
System
User is in Grocery List Page
Click delete ingredient button
Remove ingredient and Display

Figure 32: Remove List Ingredients System Sequence Diagram

Figure 32 is the system sequence diagram for removing grocery list ingredients. By clicking the delete button for an existing ingredient in the grocery list, the ingredient will be removed and redisplayed.

System Sequence Diagram - Verify Ingredients

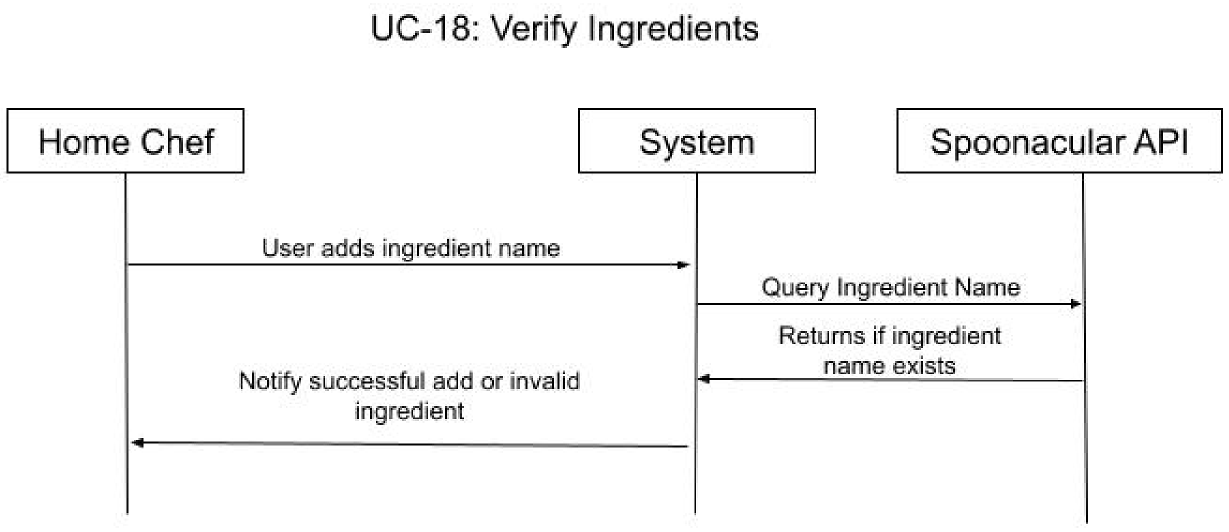

UC-18: Verify Ingredients
Home Chef
System
Spoonacular API
User adds ingredient name
Query Ingredient Name
Returns if ingredient name exists
Notify successful add or invalid ingredient

Figure 33: Verify Ingredients System Sequence Diagram

Figure 33 is the system sequence diagram for verifying ingredients. Whenever the user attempts to add an ingredient, the name of the ingredient is sent to the Spoonacular API. The API returns whether or not the ingredient name exists. If it does, the ingredient is successfully added for the user. Otherwise, the user is notified they have made an error.

System Sequence Diagram - Ingredient Threshold Warning

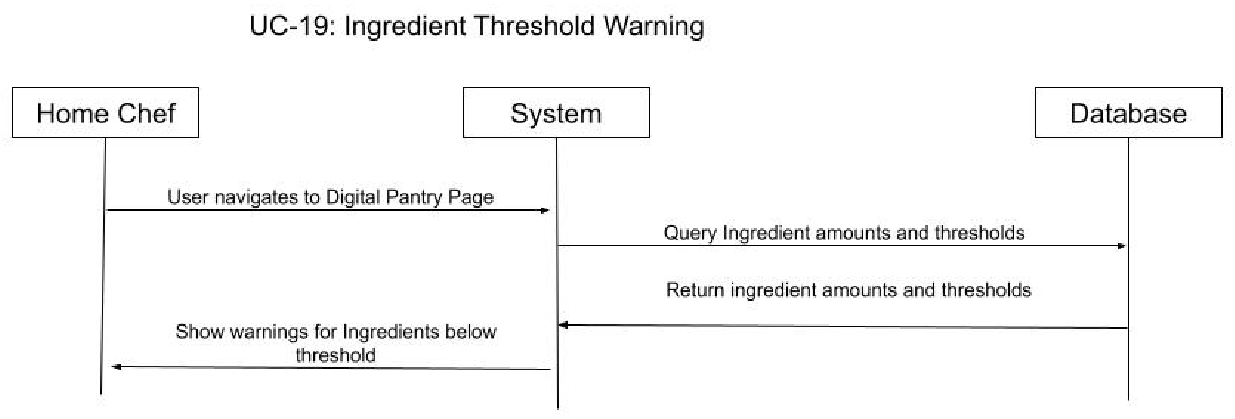

UC-19: Ingredient Threshold Warning
Home Chef
System
Database
User navigates to Digital Pantry Page
Query Ingredient amounts and thresholds
Return ingredient amounts and thresholds
Show warnings for Ingredients below threshold

Figure 34: Ingredient Threshold Warning System Sequence Diagram

Figure 34 is the system sequence diagram for showing the ingredient threshold warning. In the Digital Pantry Page, ingredients are queried from the database. The ingredient amounts and their thresholds are compared for every ingredient. If the amount is less than the threshold, then the user is warned that their amount is low in stock.

System Sequence Diagram - Show Missing Recipe Ingredients

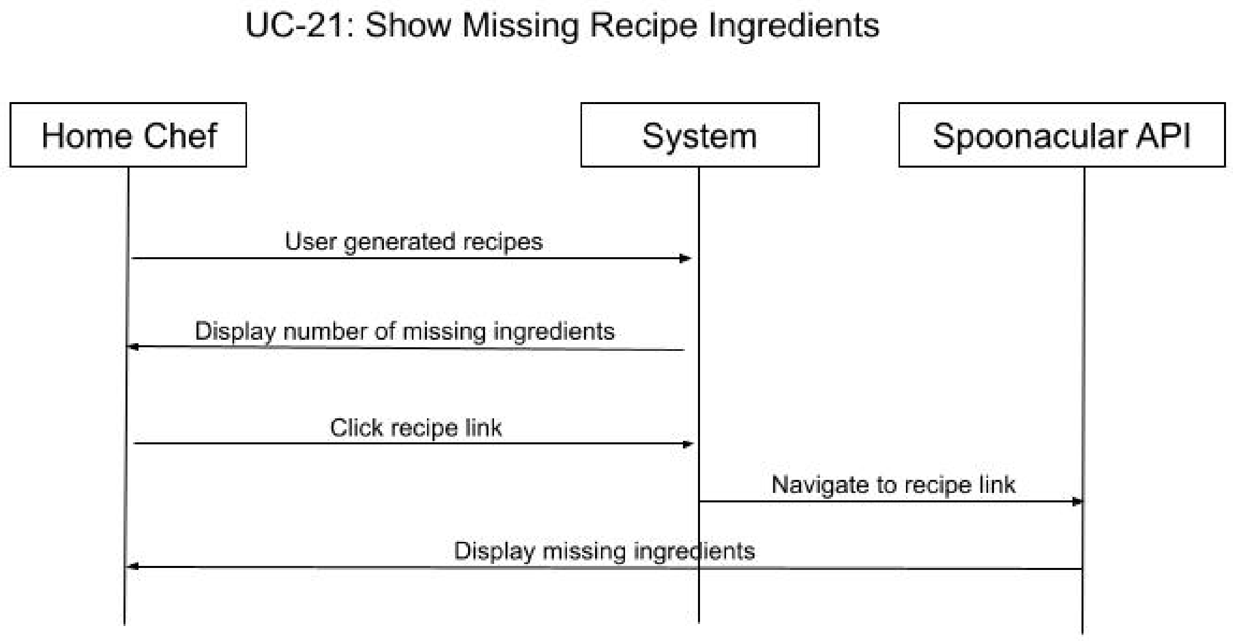

UC-21: Show Missing Recipe Ingredients
Home Chef
System
Spoonacular API
User generated recipes
Display number of missing ingredients
Click recipe link
Navigate to recipe link
Display missing ingredients

Figure 35: Show Missing Recipe Ingredients System Sequence Diagram

Figure 50 is the system sequence diagram for showing missing recipe ingredients. After generating a recipe, the system will display the number of missing ingredients. For more details, the user can click the recipe link to go to Spoonacular to see a more detailed list of their missing ingredients.

Effort Estimation Using Use Case Points

Unadjusted Use Case Weight

Use Case Weights

Use Case Classification	Class Description	Weight
Simple	1 participating actor 1 to 3 steps	5
Average	2 participating actors 4 to 5 steps	10
Complex	3 participating actors 6 or more steps	15

Table 14: Table for Use case weightings

Table 14 shows the classification and weightings for each potential use case depending on the number of participating actors as well as the number of steps to enact the use case.

Use Cases Classifications

Use Case	Description	Weight
UC-1:Signup	Populate text fields and dropdowns, query and edit database, click button, redirect user. 5 steps, 1 actor	Average 10
UC-2: Track Pantry	Click button, populate fields, click button, query and edit database, confirm changes, navigate to main screen. 7 steps, 1 actor.	Complex 15
UC-3: Login	Populate text fields and click the button, query database, navigate to main screen, populate tables, display user preferences 6 steps, 1 actor	Complex 15
UC-4: Create Grocery List	Click button, query database, display. 3 Steps, 1 actor	Simple 5
UC-5: View Grocery Store	Click the button, sort. 2 steps, 1 actor	Simple 5
UC-6: Sort Grocery Store Items	Sort existing tables. 1 step, 1 actor	Simple 5

UC-7: Discover Recipes	Choose filters and sorts, add manual ingredients, query Spoonacular, return recipes to the user for display. 4 steps, 1 actor	Average 10
UC-8: Sort/Filter Recipes	Choose filter params, choose sorting params, display. 3 steps, 1 actor	Simple 5
UC-9: Viewing Recipe	Simple UI, display organized page to user. 1 step, 1 actor	Simple 5
UC-10: Account Editing	Navigate to tab, edit fields and dropdowns, query and update database. 3 steps, 1 actor	Simple 5
UC-11: Logout	Click the button. 1 step, 1 actor	Simple 5
UC-12: View Pantry Ingredients	Navigate to tab. 1 step, 1 actor	Simple 5
UC-13: Add Pantry Ingredients	Click button, edit fields and dropdowns, query and update database. 3 steps, 1 actor	Simple 5
UC-14: Remove Pantry Ingredients	Click button, confirm alert. 2 steps, 1 actor	Simple 5
UC-15: View List Ingredients	Navigate to tab. 1 step, 1 actor	Simple 5
UC-16: Add List Ingredients	Click button, edit fields and dropdowns, query and update database. 3 steps, 1 actor	Average 10
UC-17: Remove List Ingredients	Click button, confirm alert. 2 steps, 1 actor	Simple 5
UC-18: Verify Ingredients	Fill text field, click button, query Spoonacular, notify user. 4 steps, 1 actor	Average 10
UC-19: Ingredient Threshold Warning	Query database, compare to threshold values, return alert, confirm alert. 4 steps, 1 actor	Average 10
UC-20: Shop Missing Recipe Ingredients	Click button, redirect to tab, run query on three database tables, return temporary table to user, confirm additions for each	Complex 15

	row to grocery list table. 8 steps, 1 actor	
UC-21: Show Missing Recipe Ingredients	Click the button to generate recipes, choose recipe, navigate to Spoonacular site, check ingredients, match to digital pantry. 6 steps, 1 actor	Complex 15

Table 15: Table for Use case classifications

Table 15 shows the classified use cases according to Table 14 weightings. Each use case is listed out and a description for the number of steps each user is accompanying each use case.

Each use case is classified to be simple, average or complex.

As a result, we are able to compute the following:

UUCW(ChefCart) = 12*Simple + 5*Average + 3*Complex = 12*5 + 5*10 + 3*15 = 155

Technical Complexity Factor

Technical Factors

Technical Factor	**Characteristics**	**Weight**	**Complexity (Relative, 1-5)**	**Calculated Factor (W*C)**
T1	Ease of use for Home Chef (user)	2	5	2 * 5 = 10
T2	Concurrent use is not required (1 actor only)	0.5	0	0.5 * 0 = 0
T3	Reasonably efficient use of hardware resources	1	3	1 * 3 = 3
T4	No access for third party users	0.5	0	0.5 * 0 = 0
T5	Generality/portability for all users	1	5	1 * 5 = 5
T6	Reasonably fast performance, on the order of seconds/operation	1	4	1 * 4 = 4
T7	Users do not require training to use system	2	5	2 * 5 = 10

T8	Minimal developer maintenance	1	3	1 * 3 = 3

Table 16: Table for Calculated Factors for Technical Factors

Table 16 computes the Calculated Factors for each technical factor for ChefCart according to their characteristics, weights and complexities.

As a result, we can compute the following:

Technical Factor Total = 35
TCF = 0.6 + (Technical Factor Total / 100) = 0.6 + (0.35) = 0.95

Environmental Complexity Factors

We will assume that ECF = 1

Use Case Points

Below is the computation to calculate the Use Case Points.

UCP = UUCW * TCF * ECF
Unadjusted Use Case Weight = 155
Technical Complexity Factor = 0.95
Environmental Complexity Factor = 1

UCP = 155 * 0.95 * 1 = 147.25

ChefCart has a final Use Case Points (UCP) of 147.25 points.

Duration

Duration = UCP * PF
PF given as 28
147.25 * 28 = 4,123 hours

Final duration for ChefCart is 4,123 hours.

Domain Analysis

Conceptual Model

Concept Definitions

Concept Definitions

Responsibility	Type	Concept Name
Display options and data for the user to interact with ChefCart.	D	Interface
Manages connections with the database	D	DBConnection
Displays Account information to the user.	D	AccountDetails
Stores account information for login and dietary restrictions.	K	AccountDetails
Warns the user of ingredients that are below a certain amount in the Digital Pantry.	D	ThresholdWarning
Stores list of ingredient information for a user.	K	DigitalPantry
Display list of ingredients saved for the user account.	D	DigitalPantry
An ingredient object that tracks an ingredient for the user in their pantry.	K	Ingredient
Add, edit or remove ingredients for temporary use in recipe generation or grocery list generation.	D	IngredientManager
Generates a grocery list for the user.	D	GroceryList
Sorts shown grocery store ingredients based on user preferences.	D	StoreIngredientSorter
Tracks the recipe filters.	K	RecipeFilter
Filters recipes based on user preferences.	D	RecipeFilter
Used in the recipe generator to track usage of digital pantry ingredients in recipe generation.	K	DigitalPantry

Table 17: Table for Domain Concepts, Types and Responsibilities defined

Table 17 lists each domain concept, their type of Doing or Knowing and their responsibilities.

Association Definitions

Association Definitions

Concept Pair	Association Description	Association Name
AccountDetails ⇔ DBConnection	Retrieves account details from the database.	QueryDB
AccountDetails ⇔ Interface	Display account details.	DisplayAccount
ThresholdWarning ⇔ Interface	Display low ingredients in the Digital Pantry.	Warn
DigitalPantry ⇔ Interface	Display pantry ingredients and edit them.	DisplayPantry
DigitalPantry ⇔ Ingredient	Display pantry edits each ingredient object to be tracked.	Tracks
DigitalPantry ⇔ DBConnection	Save current pantry ingredients to the database.	SavePantry
IngredientManager ⇔ Interface	Add, edit or remove ingredients for temporary use in the grocery list or recipe generator and display.	CustomizeIngredients
GroceryList⇔ Interface	Display generated grocery list to user.	DisplayGroceries
StoreIngredientSorter ⇔ Interface	Display sorted ingredients.	SortIngredient
RecipeFilter ⇔ Interface	Display filtered recipes.	FilterRecipes
RecipeFilter⇔ AccountDetails	Filter out recipes that do not satisfy dietary restrictions.	RestrictRecipes
DigitalPantry ⇔ RecipeFilter	Use Digital Pantry Ingredients to help generate recipes.	UsePantry

Table 18: Table for Association Definitions of Related Pairs of Domain concepts

Table 18 lists the purpose of associating each Domain Concept and names them, thus defining each association. Only related domain concepts are associated.

Attribute Definitions

Attribute Definitions

Concept	Attribute	Description
AccountDetails	Email	Associated email of the account
	Password	Password of the account
	City	City of user
	State	State of user
	User_Dietary_Restrictions	Allows the user to pick several dietary restrictions
Ingredient	Ingredient_Name	The name of the ingredient
	Ingredient_Quantity	Integer corresponding to how much "Ingredient_Units" of the ingredient there is
	Ingredient_Units	Various units of measurement for the specific ingredient. (Extracted into one attribute to represent Weight, Quantity, or Volume)
	Expiration_Date	The expiration date of the ingredient
	Warning_Threshold	The minimum quantity the user wants to set before a notification is made to buy more of the ingredient
DigitalPantry	Pantry_Ingredient_List	List of pantry ingredients, referencing the concept of Ingredients
GroceryList	Grocery_Ingredient_List	List of grocery ingredients
StoreIngredientSorter	Sort_By	Current sorting mechanism for ingredient name, availability, rating, review, and price.
RecipeFilter	Digital_Pantry	Using Digital Pantry ingredients to discover recipes.
	Additional_Ingredient	Add additional ingredients to generate recipes.
	Filter_by	Filter the recipes by dietary restriction, intolerance, specific cuisines, and excluded ingredients
	Sort_By	Current sorting mechanism for missing ingredient, used ingredient, and ingredient name

Table 19: Table for defining Attributes on Domain Concepts

Table 18 defines the attributes associated with each domain concept as well as their purpose. If a concept is not listed, that is because it will not be needing to store any attributes.

Traceability Matrix

Traceability Matrix - Domain Concept

Use Cases		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Priority Weight		3	5	5	5	5	4	5	4	4	3	4	5	5	5	4	4	4	4	4	4	4
Domain Concepts	Interface		X			X		X		X		X	X			X					X	X
	DBConnection		X		X			X						X	X		X	X				X
	Account Details	X		X							X	X										
	Ingredients													X	X		X	X	X			
	Digital Pantry		X										X	X	X							
	Grocery List				X											X	X	X			X	
	Store Ingredient Filter						X		X													
	Recipe Filter								X													
	Threshold Warning																			X		
	Ingredient Manager		X		X									X	X		X	X	X			

Table 20: Table for Mapping Domain Concepts to Use Cases

Table 20 maps each use case to each Domain Concept such that domain concepts will cover every use case. From the table, we see that this is true and can ensure that each use case is accounted for by a domain concept. In addition we see that each domain concept is accounted for by at least one domain concept. There are no dangling domain concepts that serve no purpose.

Domain Model Diagram

Domain Model Diagram

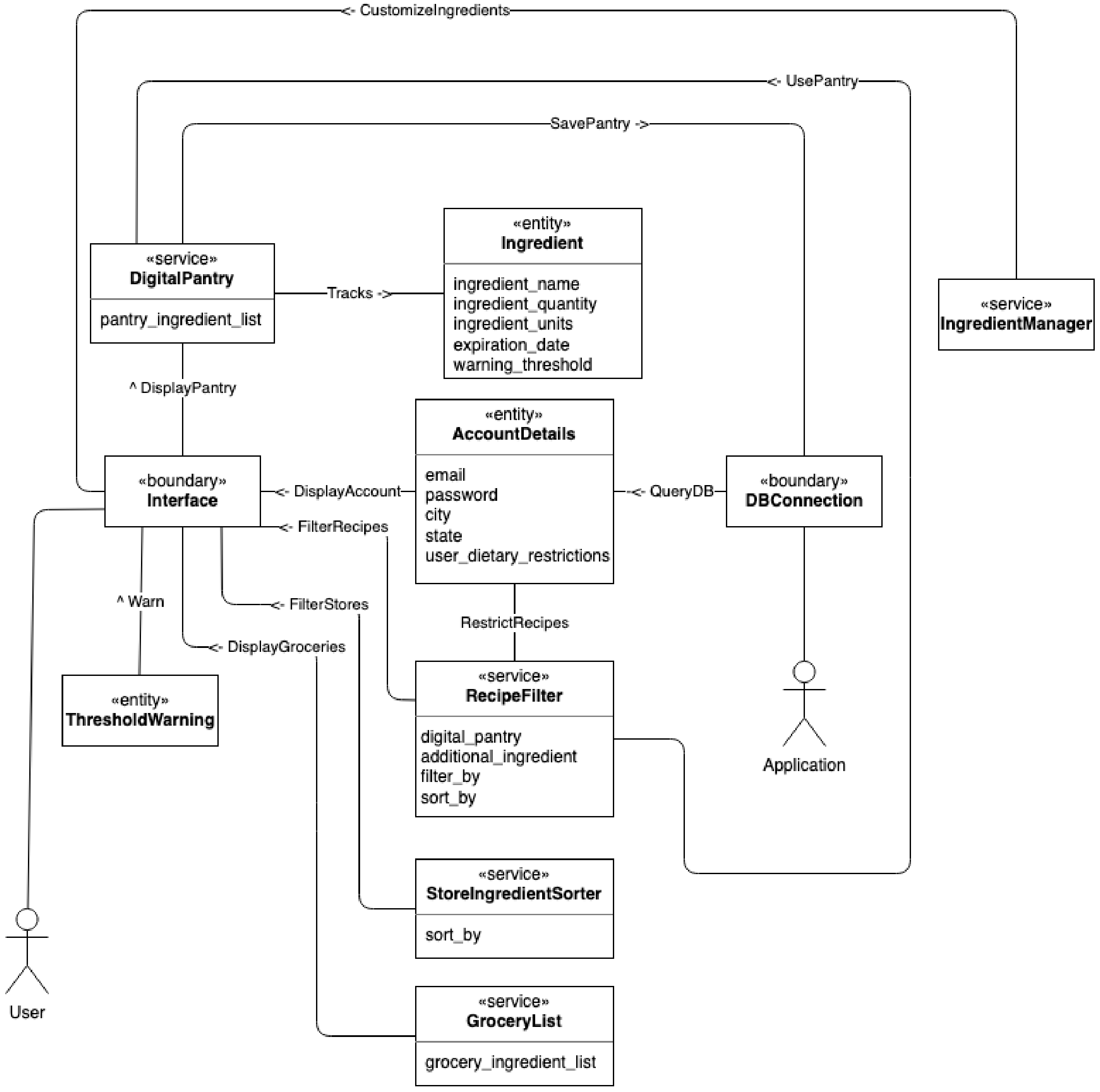


Figure 8: Diagram of the entire ChefCart Domain Model

Figure 8 is the diagram of the domain model. Each feature is shown as a box and is labeled with its type. The association definitions are then used to connect relevant domain concepts. Inside each concept are their attributes as defined before.

System Operation Contracts

System Operation Contract - Track Pantry

Operation	Track Pantry
Use Case:	UC-2
Preconditions	● The user has logged in and has navigated to the Digital Pantry tab in the interface. ● ThresholdWarning shows a warning for a particular ingredient if the Warning_Threshold amount is greater than the current ingredient amount or Ingredient_Unit attribute.
Postconditions	● Attributes for the ingredient concept will be updated and the list of ingredients in the Digital Pantry will be updated in the database. ● The interface will display the updated list to the user. ● The database and the interface will hold the same data.

Table 21: Track Pantry System Operation Contract

Table 21 lists the preconditions and postconditions for tracking the user pantry in the form of a System operation Contract.

System Operation Contract - Create Grocery List

Operation	Create Grocery List
Use Case:	UC-4
Preconditions	● The user has launched the ChefCart app. ● The user has successfully logged into ChefCart. ● After logging in, the user has navigated to the Grocery List Tab.
Postconditions	● The interface displays a list of ingredients that will be the user's created GroceryList

Table 22: Create Grocery List System Operation Contract

Table 22 lists the preconditions and postconditions for creating a grocery list in the form of a System operation Contract.

System Operation Contract - View Grocery Store

Operation	View Grocery Store
Use Case:	UC-5
Preconditions	● The user has launched and logged into ChefCart ● The user was on the grocery list page, generated a grocery list and clicked on the “Search” button.
Postconditions	● The displayed store, if any, will allow users to click on ingredients that are sold by it to be sent to the store website. ● The ingredients will be sorted and filtered according to the StoreIngredientFilter ● The store information will be displayed at the top.

Table 23: View Grocery Store System Operation Contract

Table 23 lists the preconditions and postconditions for viewing a grocery store and its ingredients in the form of a System operation Contract.

System Operation Contract - Discover Recipes

Operation	Discover Recipes
Use Case:	UC-7
Preconditions	● The user has launched and logged into ChefCart ● The user has navigated to the Recipe Generator page.
Postconditions	● The system will display any found recipes to the user. ● All recipes displayed in the interface satisfy the RecipeFilter. ● All recipes displayed in the interface satisfy AccountDetails User_Dietary_Restrictions. ● Recipes listed will use a high percentage of ingredients that were manually inputted in the previous page and the DigitalPantry ingredients if chosen through the Digital_Pantry attribute of the RecipeFilter.

Table 24: Discover Recipes System Operation Contract

Table 24 lists the preconditions and postconditions for finding recipes using relevant ingredients in the form of a System operation Contract.

Data Model:

ChefCart needs to save data to outlive a single execution. We chose to use a relational database powered by PostgreSQL, a free and open-source RDBMS. This will match well with our knowledge of relational databases and SQL. Persistent data is required for a few of our concept identities. These include AccountDetails for each user and DigitalPantry to store the list of ingredients each user has as well as an Ingredients table. Each user has a unique Account_ID and each user has only one Digital Pantry. It is assumed that users are uniquely identified by Account_ID as well as Digital_Pantry. Finally, Ingredients are uniquely identified by Ingredient_ID and Account_ID.

Entity Relationship Diagram

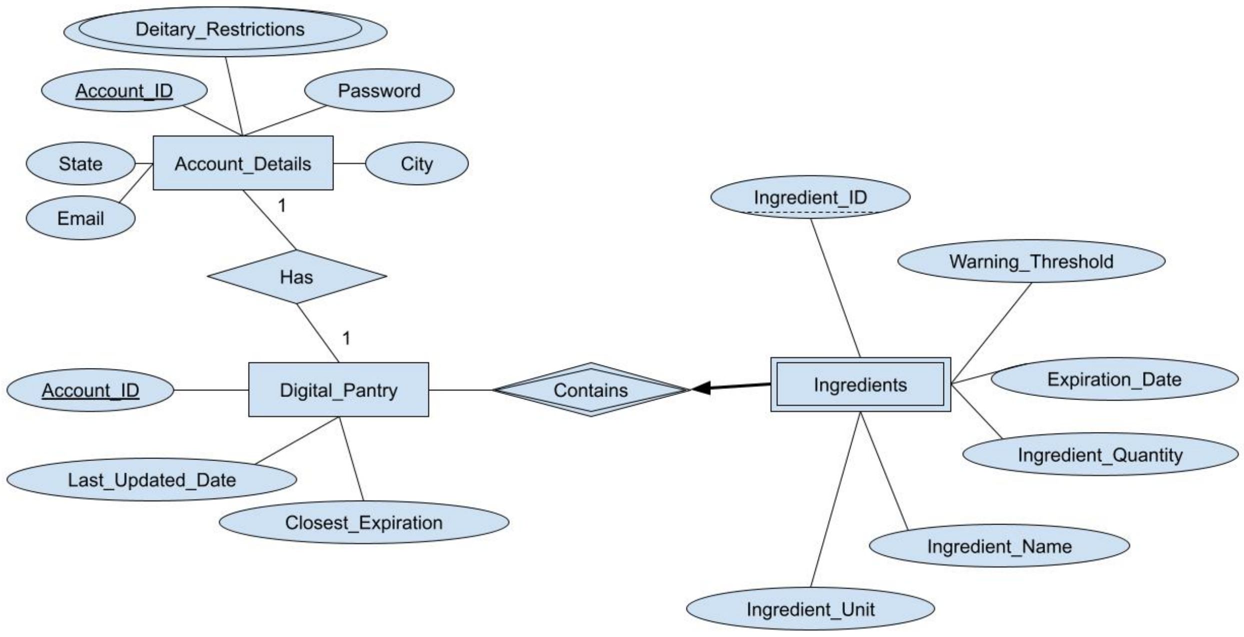


Figure 9: Entity Relationship Diagram of Database

Figure 9 is a diagram of the Entity Relationship Diagram for all the data that we want to store. We noticed that each account and Digital Pantry must be associated with each other in a one-to-one relationship, so the Digital_Pantry entity simply needs to be using an Account_ID as its primary key. We also noticed that the Ingredients concept needed to belong to exactly one Digital_Pantry and each Digital_Pantry can have many ingredients. Because of this relationship, we can represent Ingredients as a weak entity with Ingreient_ID being a partial key and the Account_ID being the other, which is taken from the Digital_Pantry.

Relational Schema

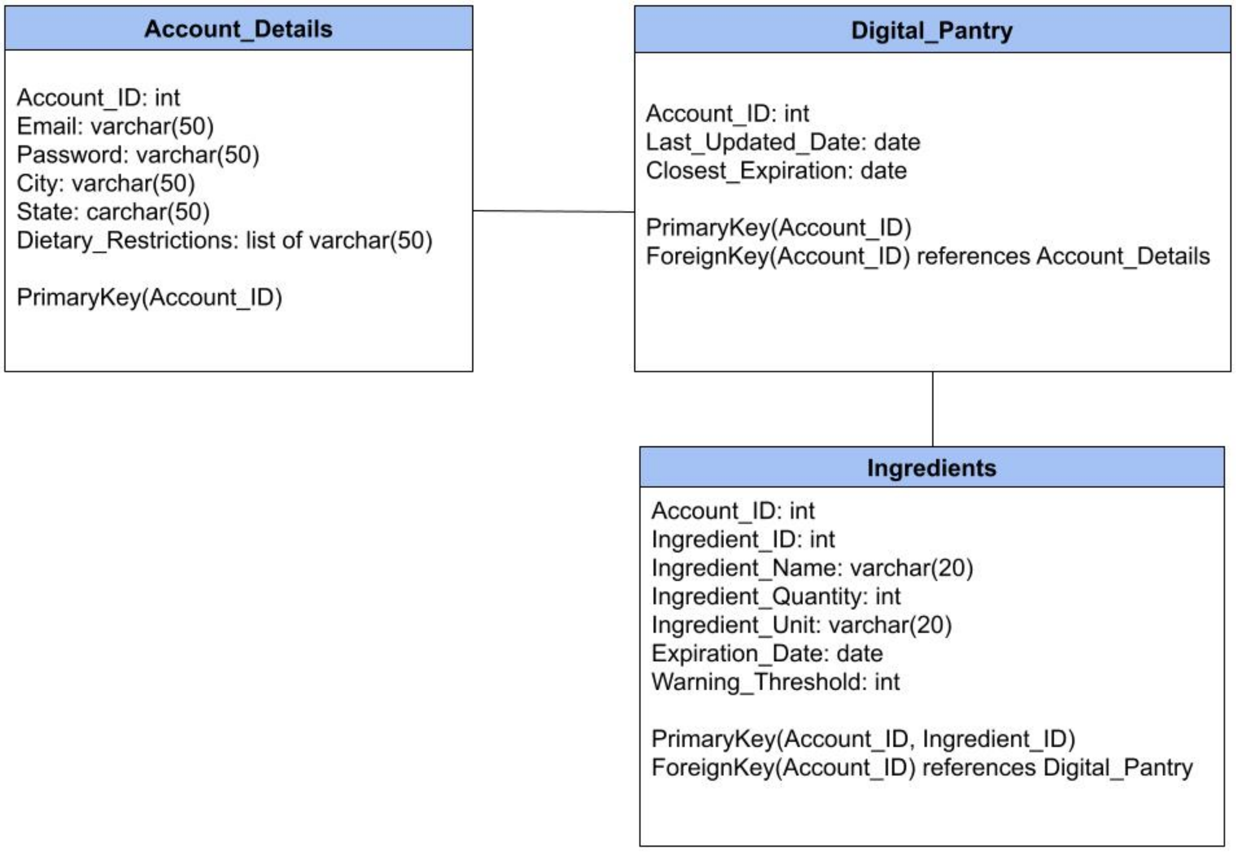


Figure 10: Relational Schema of Database

Figure 10 a diagram of the relational schema of our database that is a result of the above ER diagram. Other functionalities such as the Recipe Generator and Grocery List will query these tables as necessary to generate their respective results. However, we will not need to store information as generators will create results each time. Each attribute stored is shown and their primary keys and foreign key references as well.

Mathematical Model

We do not use a mathematical model in our project.

Interaction Diagrams

Track Pantry

Interaction Diagram - Track Pantry

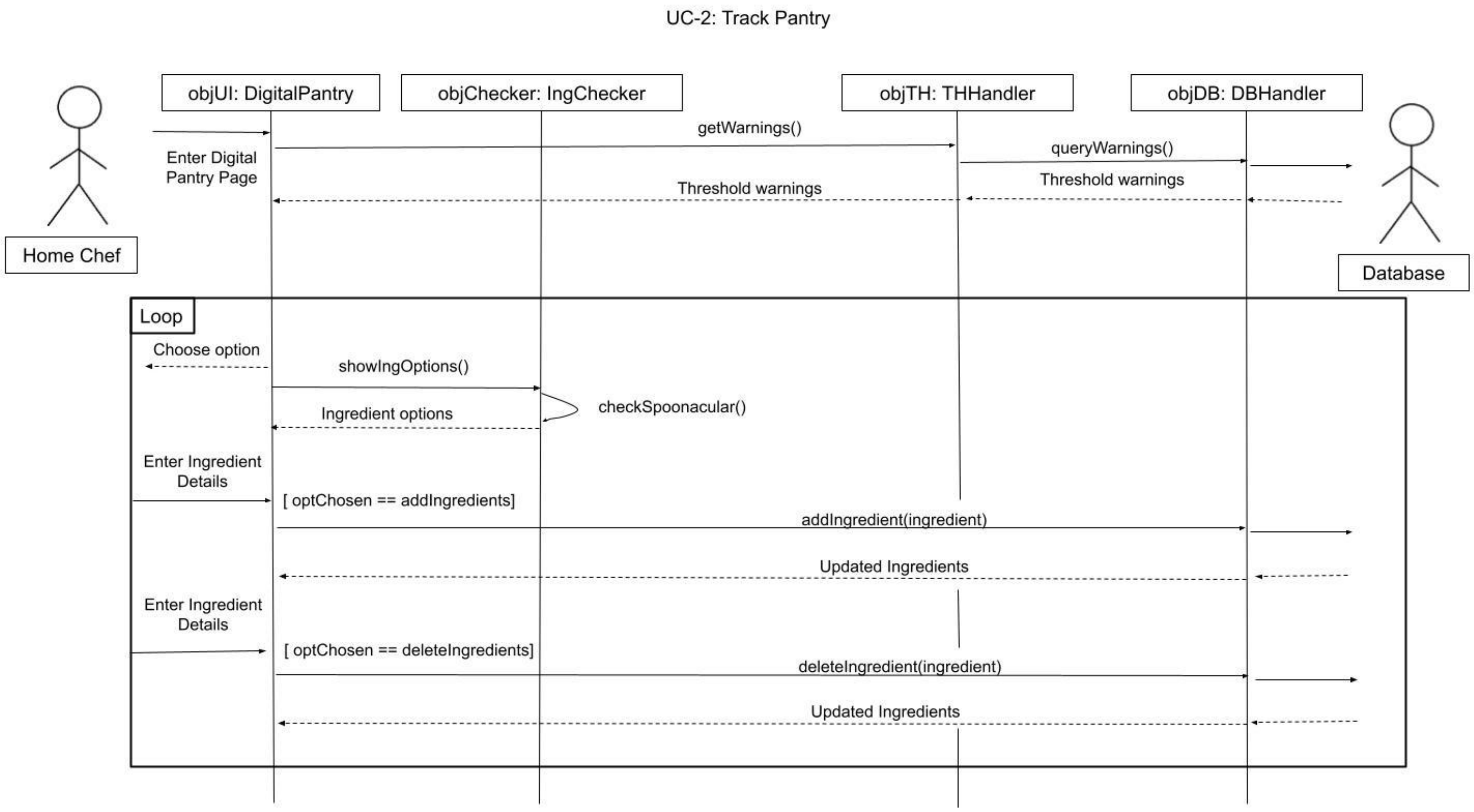


Figure 11: Interaction Diagram for Track Pantry

The UC-2: Track Pantry diagram, shows how the user can track their own digital pantry. Once the user is in the digital pantry section, the user can update their digital pantry. The user can add or remove ingredients from the digital pantry. While the user is inputting/updating the ingredient, we need to be able to show threshold warnings to the user and make sure that they are updating ingredients appropriately by using the IngChecker class to get the appropriate ingredient options from the Spoonacular API. We decided to remove the option to edit ingredients because it can be simplified to adding or deleting ingredients by amount instead. All the updated digital pantry will be stored in the database.

One of the design principles is the high cohesion principle. The high cohesion principle says that the responsibilities given to an element/object should be highly focused, manageable, and should not take on too many computational responsibilities. In the Track Pantry diagram, a specific task is assigned to each element. All the tasks are mainly focused on tracking the digital pantry.Another design principle that is demonstrated in our diagram is the creator principle. Creator principle decides on which class is responsible for creating objects. In our case, every time the user adds a new ingredient, a new ingredient object is created. The low coupling principle is also used. Low coupling determines how an object or element relies on another element or object. In this UC, there are many modules: Home Chef, spoonacular API, and database. Every time an ingredient is added, we need to make sure that the ingredient can be found in the Spoonacular API to determine if the ingredient exists and if it exists, it will be stored in the database. We ensured that each module has the minimum number of dependencies possible for successful implementation so that one module will affect at most one module.

A design pattern that is used in this interaction diagram is the command pattern. In the user interface, the user must input the fields to add, edit and delete ingredients. These ingredients fields need to be processed, packaged and persisted in the database. We encapsulate the client requests to make an update to their pantry ingredients with an ingredient struct using Go. In doing so, different clients can make different requests and the requests can easily be tracked, queued and logged.

Create Grocery List

Interaction Diagram - Create Grocery List

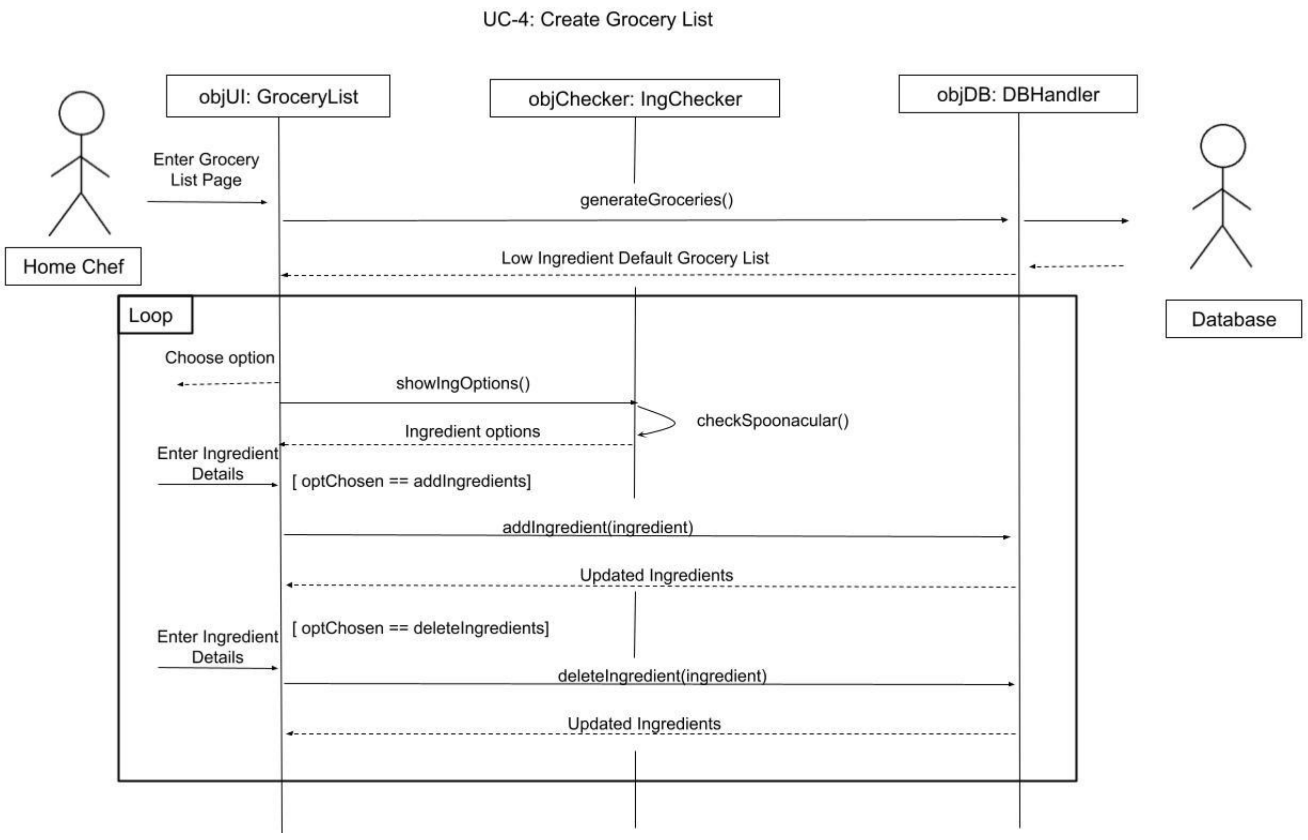


Figure 37: Interaction Diagram for Create Grocery List

The UC-4: Create Grocery List diagram above, shows how the user can create their own grocery list. Once the user is in the Grocery List section, the user can update their grocery list. The user can add or remove ingredients from the grocery list. While the user is inputting the ingredient, we need to make sure that they are inputting/updating the ingredients appropriately by using the IngChecker class to get the appropriate ingredient options from the Spoonacular API. Editing ingredients is not present in the interaction diagram, because it is included in both adding and deleting ingredients. The grocery list will also automatically generate ingredients by using the digital pantry. The information relating to the ingredients in the digital pantry is stored in the database and we can use the database to determine if the ingredients are low in quantity. If the quantity is low, then it will be added to the grocery list.

The first design principle that is important to this interaction diagram is the low coupling principle. To achieve low coupling, we focused on the goal that making changes to one module would not affect the operation of another. Since there are several modules interacting in this UC (Home chef, spoonacular, database) we had to make sure that each module can act independently. Additionally, we ensured that each module has the minimum number of dependencies possible for successful implementation. This way, changes to one module will affect at most one other module. Another design principle in this diagram is high cohesion. We accomplished high cohesion here by creating a specified 'workflow'. Through this workflow, a user using the project or a developer viewing the code would be able to follow logically from one module to another, and understand the entire interaction diagram. We built the code and the project around this workflow, to avoid fragmented and unorganized interactions.

A design pattern used in this diagram is the module pattern. The Grocery List needs to retrieve the Digital Pantry ingredients just like the Digital Pantry page. In this sense, the functionality to retrieve ingredients is the same. Thus, the DBHandler object is essential as a module that extracts out the functionality to communicate with the database. Here, the Grocery List uses this module to retrieve the current digital pantry ingredients from the database without any extra code. This allows any future features that need to interact with the database ingredients or the user account to simply need to call the DBHandler. There will not be any need to reinvent the wheel.

View Grocery Stores

Interaction Diagram - View Grocery Store

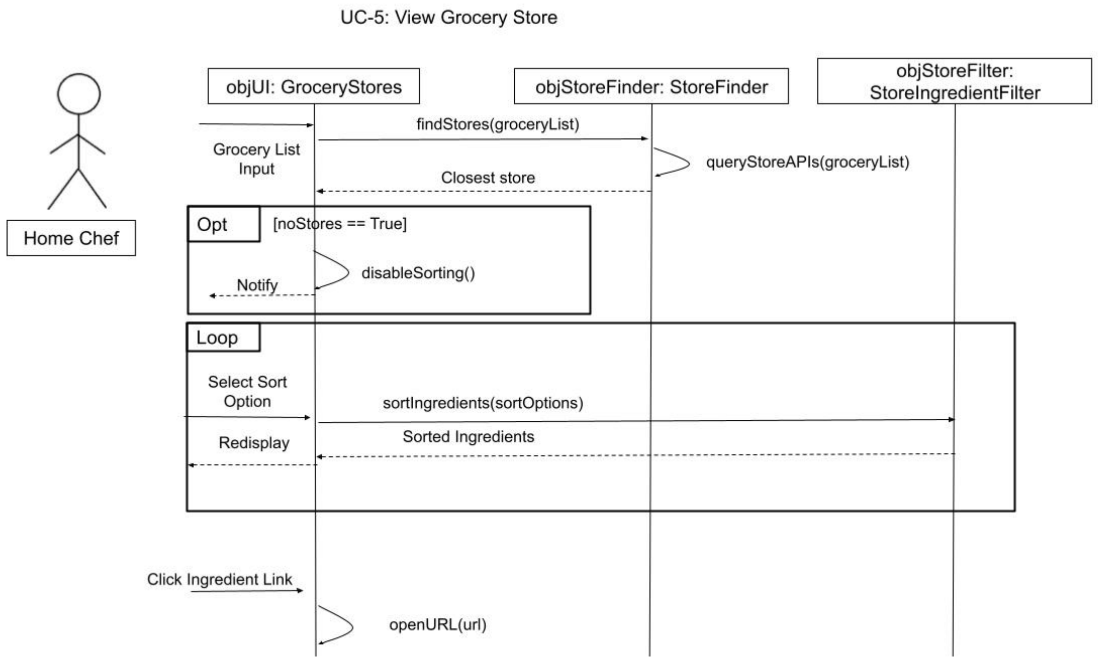


Figure 12: Interaction Diagram for View Grocery Store

The UC-5: View Grocery Store diagram below, shows how the user can generate a list of relevant items from the closest grocery store, based on the user's location preferences. Once the user decides to search for an ingredient in his/her Grocery List in store, he/she can generate a table of associated store items in the Grocery Store closest to him/her. Each store item in the table has its picture, name, store link, availability, price, rating, and number of reviews displayed. The user can sort the table by name, availability, price, rating, and number of reviews.

For this UC, our most important principle was the 'expert doer' or information expert principle. There are several pieces of information floating in this use case. We need the user's location, the location of grocery stores, the user's store preferences and filters, and other info about each specific store. In order to accomplish this, we created classes based on the information needed by each task. The 'Account' class contains all of the user's information, the 'Store' class contains all information about the stores, and we can use the methods of each class to access the attributes required for the execution of this use case.

The design pattern used in this interaction diagram is the dependency injection pattern. The StoreFinder object needs to be able to take a grocery list object in order to produce a store that is best suited for the user. In this way, the GroceryStores object is injected into the StoreFinder object as a dependency. This allows the separation of concerns between the grocery list and the act of finding a store. By representing the grocery list in an abstract way, any changes to the grocery list representation can still allow the StoreFinder to find stores based on the list.

Discover Recipes

Interaction Diagram - Discover Recipes

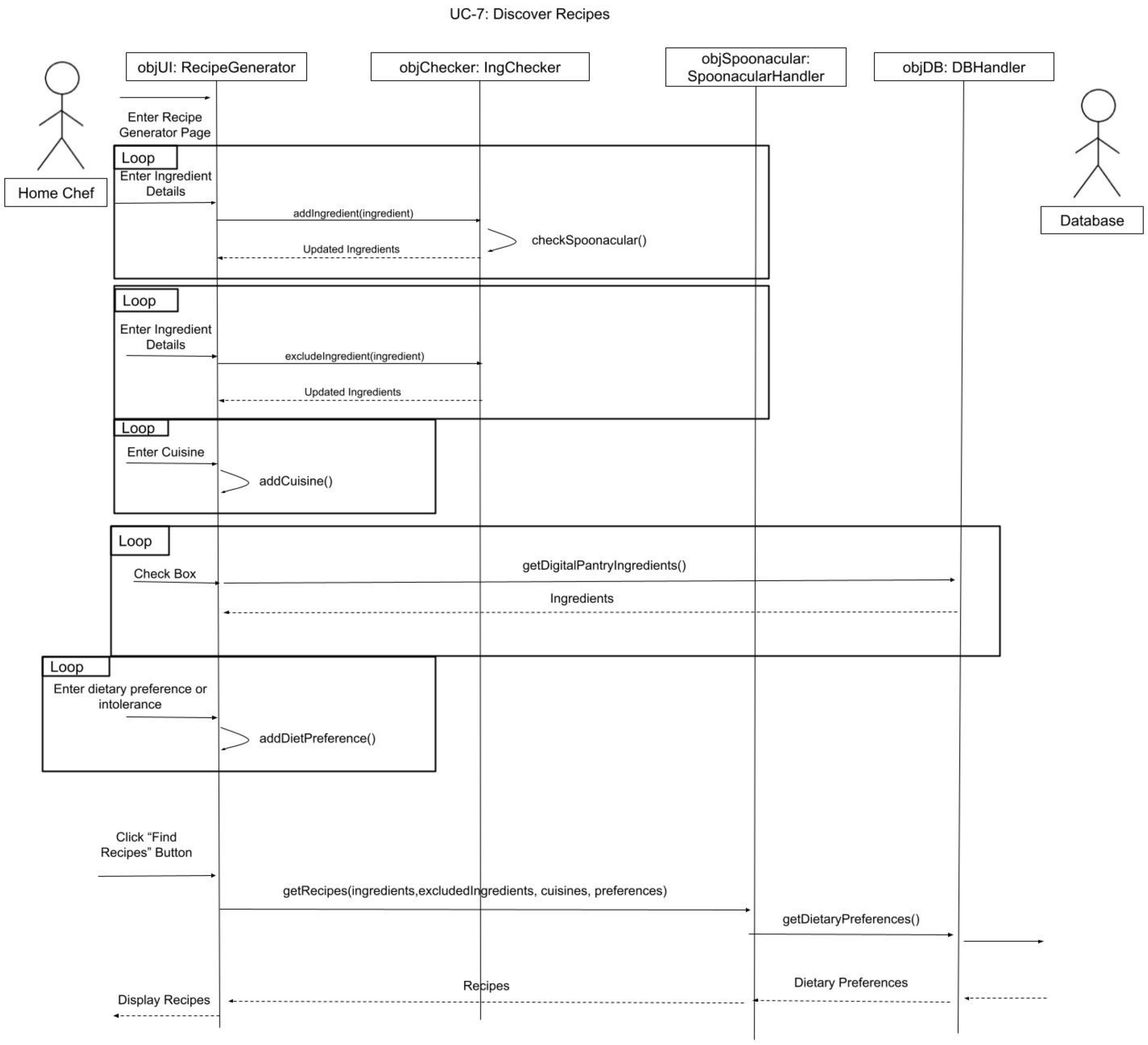


Figure 13: Interaction Diagram for Discover Recipe

The diagram above shows how the user can generate recipes. Once the user is in the Recipe Generator, they can generate a list of recipes by using the digital pantry or manual inputs. There are check boxes for the user to click if they want to use the particular ingredients already in their digital pantry in their recipe search. If the user chooses to input the ingredients, they can add or remove ingredients from the list before generating the recipe. While the user is inputting/updating the ingredient, we make sure that they are updating the ingredients appropriately by using the IngChecker class to get the appropriate ingredient options from the

Spoonacular API. After all the ingredients are finalized and the user clicks on the Discover Recipe button, a list of recipes will be generated, showing the number of searched ingredients used, and the number of ingredients missing from the digital pantry for each listed recipe.

Once again, we followed the design principles mentioned above such as expert doer, high cohesion, and low coupling in order to create the best interaction system between our classes and modules. Each section of the interaction has singular inputs and outputs, the classes are self contained with all the information required to execute their functions, and each class is designed based on the information required to complete a task. When we had a single class that was completing two tasks, we decomposed it into separate classes that can compound to a larger class. For example, manual inputs vs. automatic updates are not the same class, although they ultimately serve the same function by updating the grocery list.

The design pattern that is used in the interaction diagram is the Publisher-Subscriber. In the user interface, the user sends their inputs to the system. The user is the publisher. The publisher will send a message to the message broker and then it will be sent to the subscriber. In this case, the subscriber is the Spoonacular API and the database. It helps disassociate unrelated responsibilities, helps simplify/remove complex conditional logic, and allow seamless future adding of new cases. Later on, depending on the user input, there will be an output sent back to the user to tell the user if the action is successful. The user can see the generated recipes if the inputs are valid or nothing if the inputs are invalid or no matches have been found.

Class Diagram and Interface Specification

Class Diagram: The diagram below shows how our objects interact with each other. We used “Extends” arrows to show inheritance and “Use” arrows to show dependencies. In addition, we’ve specified various aggregation and composition relationships with relevant arrows, and with field types. Setters and getters are assumed and are elaborated on below.

Class Diagram

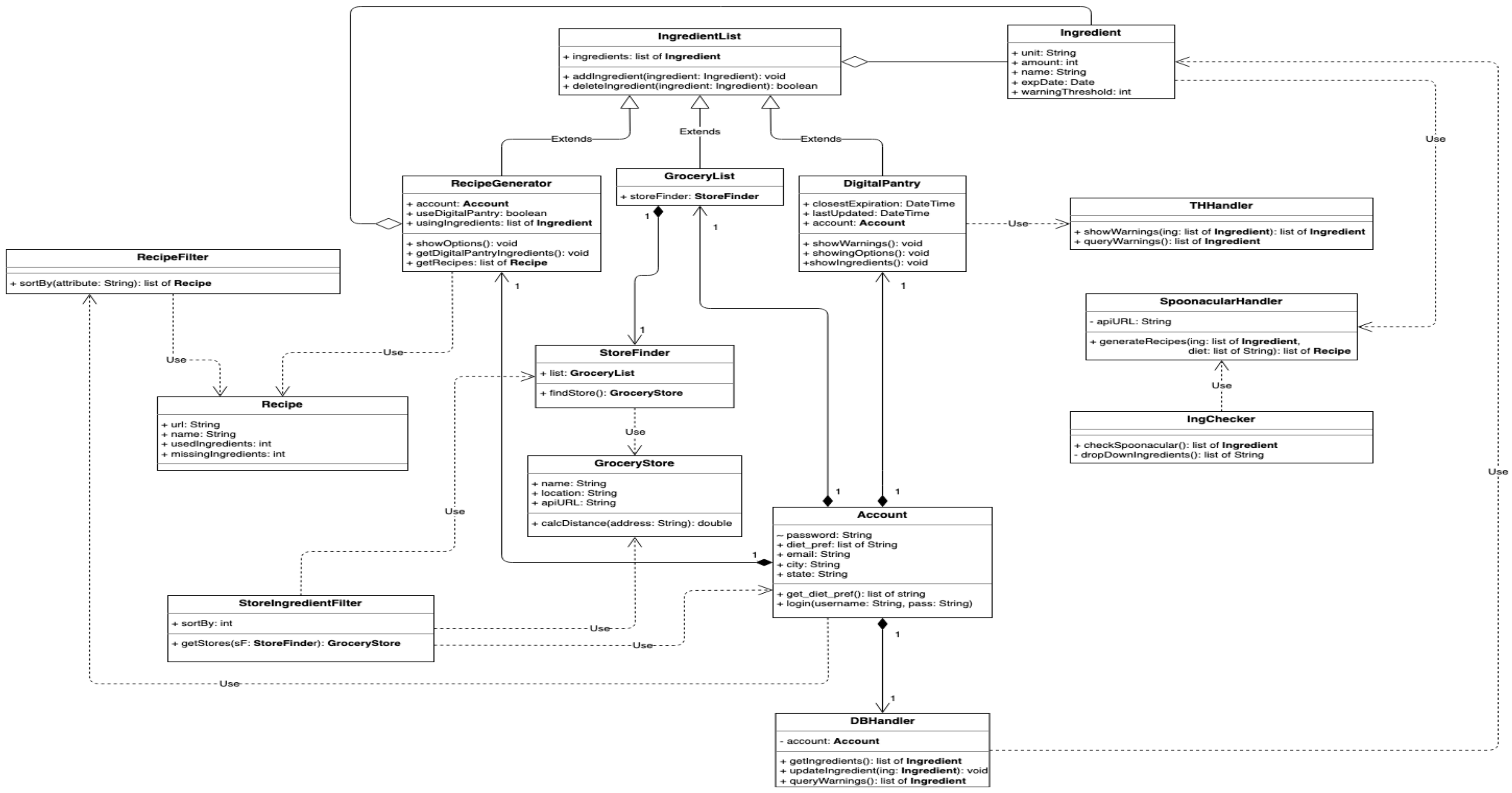
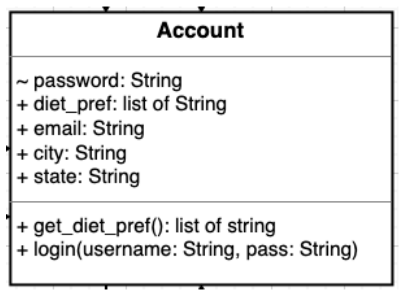

IngredientList
+ ingredients: list of Ingredient
+ addIngredient(ingredient: Ingredient): void
+ deleteIngredient(ingredient: Ingredient): boolean
Ingredient
+ unit: String
+ amount: int
+ name: String
+ expDate: Date
+ warningThreshold: int
Extends
Extends
Extends
RecipeGenerator
+ account: Account
+ useDigitalPantry: boolean
+ usingIngredients: list of Ingredient
+ showOptions(): void
+ getDigitalPantryIngredients(): void
+ getRecipes: list of Recipe
GroceryList
+ storeFinder: StoreFinder
DigitalPantry
+ closestExpiration: DateTime
+ lastUpdated: DateTime
+ account: Account
+ showWarnings(): void
+ showingOptions(): void
+showIngredients(): void
THHandler
+ showWarnings(ing: list of Ingredient): list of Ingredient
+ queryWarnings(): list of Ingredient
RecipeFilter
+ sortBy(attribute: String): list of Recipe
SpoonacularHandler
- apiURL: String
+ generateRecipes(ing: list of Ingredient, diet: list of String): list of Recipe
IngChecker
+ checkSpoonacular(): list of Ingredient
- dropDownIngredients(): list of String
StoreFinder
+ list: GroceryList
+ findStore(): GroceryStore
Recipe
+ url: String
+ name: String
+ usedIngredients: int
+ missingIngredients: int
GroceryStore
+ name: String
+ location: String
+ apiURL: String
+ calcDistance(address: String): double
Account
~ password: String
+ diet_pref: list of String
+ email: String
+ city: String
+ state: String
+ get_diet_pref(): list of string
+ login(username: String, pass: String)
StoreIngredientFilter
+ sortBy: int
+ getStores(sF: StoreFinder): GroceryStore
DBHandler
- account: Account
+ getIngredients(): list of Ingredient
+ updateIngredient(ing: Ingredient): void
+ queryWarnings(): list of Ingredient
Use
1

Figure 14: Class Diagram UML

Data Types and Operation Signatures

Note: Classes may have additional setter and getter methods listed in the Methods section.

Class: Account - This class keeps track of the account details of a user.


Account
~ password: String
+ diet_pref: list of String
+ email: String
+ city: String
+ state: String
+ get_diet_pref(): list of string
+ login(username: String, pass: String)

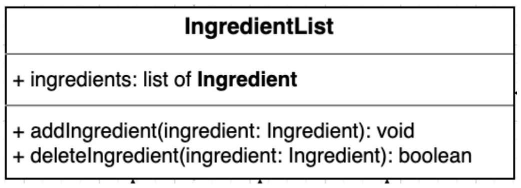
Attributes

- password: String - the password of the user
- dietPreferences: List of String - list of dietary preferences for the user such as allergies and the like
- email: String - the user email (also the username)
- city: String - the user’s city of residence
- state: String - the state where the user resides

Methods

- getDietPref(): List of String - retrieves the dietary preferences of the user
- login(username: String, pass: String): void - attempts to login the user with username and password as parameters
- Setters - Sets the value of the corresponding attribute
    - setPassword(password: String): void
    - setDietPref(preferences: List of String): void
    - setEmail(email: String): void
    - setCity(city: String): void
    - setState(state: String): void
- Getters - Gets the value of the corresponding attribute
    - getUsername(): String
    - getDietPref(): List of String
    - getEmail(): String
    - getCity(): String
    - getState(): String

Class: IngredientList - abstract class for other ingredient list structures.


IngredientList
+ ingredients: list of **Ingredient**
+ addIngredient(ingredient: Ingredient): void
+ deleteIngredient(ingredient: Ingredient): boolean

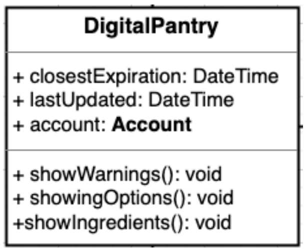
Attributes

- ingredients: List of Ingredient- the list of ingredients in the ingredient list

Method

- addIgredient(ingredient: Ingredient): void - adds an ingredient amount to the digital pantry
- deleteIngredient(ingredient: Ingredient): boolean - removes an amount of the ingredient from the digital pantry; returns true if successful, false if not.
- Getters - Gets the value of the corresponding attribute
    - getIngredients(): List of Ingredient - Gets the list of ingredients

Class: DigitalPantry - This class keeps track of the digital pantry for the user.



Attributes

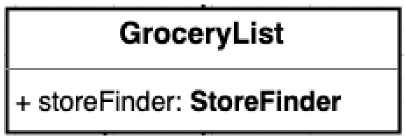
- Inherits everything from IngredientList
- closestExpiration: DateTime - the nearest expiration date and time for any ingredient
- lastUpdated: DateTime - the last updated date and time of the digital pantry
- account: Account - the account this pantry is associated with

Methods

- Inherits everything from IngredientList
- showWarnings(): void - retrieves and shows the under threshold ingredient warnings
- showIngredients(): void - displays the digital pantry to the user
- showIngOptions(): void - gets and shows the drop down menu of allowable ingredients.
- Setters - Sets the value of the corresponding attribute
    - setClosestExpiration(date: DateTime): void
    - setLastUpdated(date: DateTime): void

    - setAccount(account: Account): void
- Getters - Gets the value of the corresponding attribute
    - getClosestExpiration(): DateTime
    - getLastUpdated(): DateTime
    - getAccount(): Account

Class: GroceryList - This class helps the user create a grocery list and find stores.



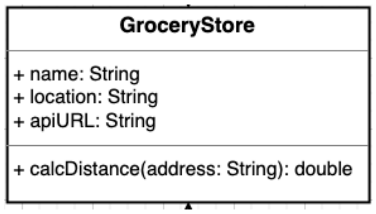
Attributes

- Inherits everything from IngredientList
- storeFinder: StoreFinder - Object for retrieving stores that have the grocery list's items

Methods

- Inherits everything from IngredientList

Class: GroceryStore - This class allows the user to find stores where they can shop based on their grocery list.



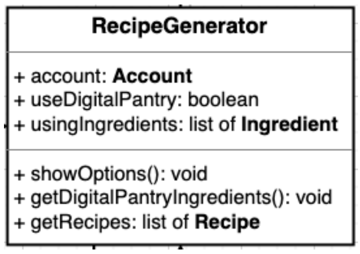
Attributes

- name: String - store's name
- Location: String - store's address
- apiURL: String - URL for store API for grocery list

Methods

- calcDistance(address: String): double: calculates store's distance from input address in miles
- Setters - Sets the value of the corresponding attribute
    - setName(name: String): void
    - setLocation(location: String): void
    - setapiURL(api: String): void
- Getters - Gets the value of the corresponding attribute
    - getName(): String
    - getLocation(): String
    - getapiURL(): String

Class: RecipeGenerator - This class generates recipes that the user can cook based on the ingredients they want to use.


RecipeGenerator
+ account: Account
+ useDigitalPantry: boolean
+ usingIngredients: list of Ingredient
+ showOptions(): void
+ getDigitalPantryIngredients(): void
+ getRecipes: list of Recipe

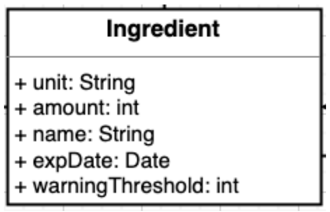
Attributes

- Inherits everything from IngredientList
- account: Account - tracks the user account to help retrieve dietary preferences
- useDigitalPantry: boolean - tracks if the user wants to use digital pantry ingredients to generate recipes.
- usingIngredients: List of Ingredients - the list of ingredients to be used in generating recipes.

Methods

- Inherits everything from IngredientList
- showOptions(): void - shows the drop down menu of allowable ingredients
- addIgredient(ing: Ingredient): void - adds an ingredient amount to the recipe generator
- deleteIngredient(ing: Ingredient): void - removes an amount of the ingredient from the recipe generator
- getIngredients(): List of Ingredient - retrieves digital pantry ingredients
- getRecipes(): List of Recipe - the list of recipes that will be generated by Spoonacular.
- Setters - Sets the value of the corresponding attribute
    - setAccount(account: Account): void
    - setUseDigitalPantry(using: boolean): void
    - setUsingIngredients(ingredients: List of Ingredient): void
- Getters - Gets the value of the corresponding attribute
    - getAccount(): Account
    - getUseDigitalPantry(): boolean
    - getUsingIngredients(): List of Ingredient

Class: Ingredient - This class stores ingredient information.



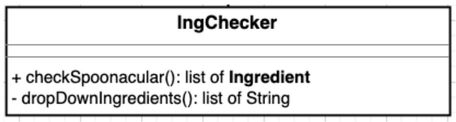
Attributes

- unit: String - the ingredient unit of measurement
- amount: int - the amount of this ingredient
- name: String - the name of the ingredient
- expirationDate: Date - the expiration date of the ingredient
- warningThreshold: int - the amount of the ingredient that the user would like to have in the pantry at all times

Methods

- Setters - Sets the value of the corresponding attribute
  - setUnit(unit: String): void
  - setAmount(amount: int): void
  - setName(name: String): void
  - setExpDate(date: Date): void
  - setWarningThreshold(threshold: int): void
- Getters - Gets the value of the corresponding attribute
  - getUnit(): String
  - getAmount(): int
  - getName(): String
  - getExpDate(): Date
  - getWarningThreshold(): int

Class: IngChecker - This class verifies and checks that each ingredient is valid.


IngChecker
+ checkSpoonacular(): list of **Ingredient**
- dropDownIngredients(): list of String

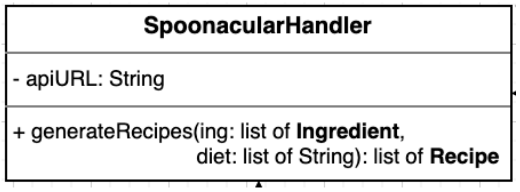
Attributes

- N/A

Methods

- checkSpoonacular(): List of String - checks spoonacular for a valid list of ingredients
- dropDownIngredients(): List<String> - called by the above function to help populate the drop down menu with ingredients

Class: SpoonacularHandler- This class generates the recipes according to the ingredients to be used and the list of dietary preferences.


SpoonacularHandler
- apiURL: String
+ generateRecipes(ing: list of **Ingredient**, diet: list of String): list of **Recipe**

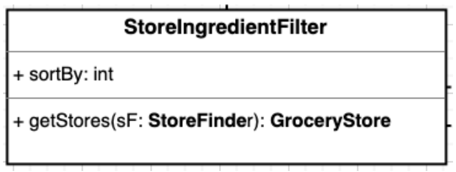
Attributes

- apiURL: String - the URL to access the Spoonacular API

Methods

- generateRecipes(ing: list of Ingredient, diet: list of String): list of Recipe

Class: StoreIngredientFilter - This class sorts store ingredients to display to the user.



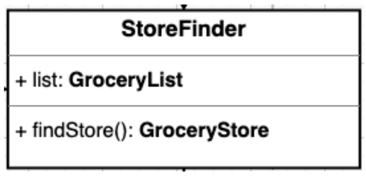
Attributes

- sortBy: int - an int representing how to sort ingredients (0 = name, 1 = availability, 2 = ratings, 3= reviews, 4 = price);

Methods

- getStore(sF: StoreFinder): GroceryStore returns the closest store.
- Setters - Sets the value of the corresponding attribute
  - setSortBy(sort: int): void
- Getters - Gets the value of the corresponding attribute
  - getSortBy(): int

Class: StoreFinder - This class finds the closest store that a user can shop at.



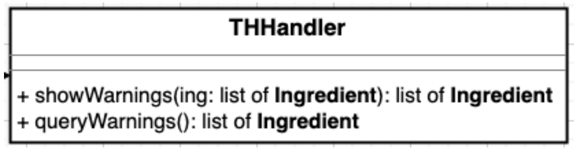
Attributes

- list: GroceryList - the relevant grocery list with which we can find stores that have its items in stock

Methods

- findStore(): GroceryStore - returns a store that have the items in the grocery list and is closest to the user
- Setters - Sets the value of the corresponding attribute
  - setList(groceryList: GroceryList): void
- Getters - Gets the value of the corresponding attribute
  - getList(): GroceryList

Class: THHandler - This class gets warnings for ingredients if they are below a threshold.



THHandler

+ showWarnings(ing: list of **Ingredient**): list of **Ingredient**
+ queryWarnings(): list of **Ingredient**

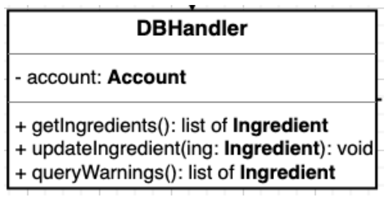
Attributes

- N/A

Methods

- showWarnings(ingredients: List of Ingredient): List of Ingredient - used to get the warnings to be showed for the returned ingredients
- queryWarnings(): List of Ingredient - This queries the database to get below threshold ingredients for the above function

Class: DBHandler - This class handles operations with the database for manipulating ingredients.



DBHandler

- account: **Account**

+ getIngredients(): list of **Ingredient**
+ updateIngredient(ing: **Ingredient**): void
+ queryWarnings(): list of **Ingredient**

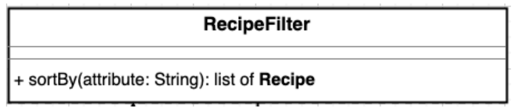
Attributes

- account: Account - This keeps track of the account that the class will query ingredients for

Methods

- getIngredients(): List of Ingredient - retrieves the list of ingredients in the digital pantry.
- updateIngredient(ingredient: Ingredient): void - updates the ingredient with the new value in the ingredient object and will perform addition and deletion accordingly
- queryWarnings(): List of Ingredient - queries the database for below threshold ingredients
- Setters - Sets the value of the corresponding attribute
  - setAccount(account: Account): void
- Getters - Gets the value of the corresponding attribute
  - getAccount(): Account

Class: RecipeFilter - This class filters recipes.



RecipeFilter

+ sortBy(attribute: String): list of **Recipe**

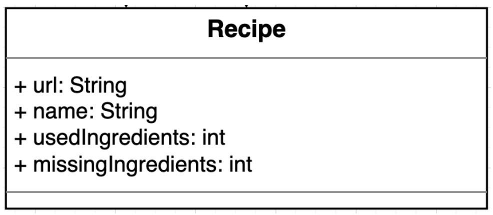
Attributes

- N/A

Methods

- sortBy(attribute: String): List of Recipe - sorts and returns the recipes sorted by an attribute

Class: Recipe - This class holds the information for a recipe.



Attributes

- url: String - the Spoonacular url to display all recipe details
- name: String - the name of the recipe
- usedIngredients: int - the number of used ingredients
- missingIngredients: int - the number of missing ingredients

Methods

- Setters - Sets the value of the corresponding attribute
  - setName(name: String): void
  - setURL(url: String): void
  - setUsedIngredients(num: int): void
  - setMissingIngredients(num: int): void
- Getters - Gets the value of the corresponding attribute
  - getName(): String
  - getURL(): String
  - getUsedIngredients(): int
  - getMissingIngredients(): int

Traceability Matrix

Traceability Matrix - Domain Concepts and Classes

Concept Name	Responsibility	Derived Classes
Interface	Display options and data for the user to interact with ChefCart.	All classes contribute to the interface.
DBConnection	Manages connections with the database	DBHandler - All database actions are accessed through the DBHandler class. Other classes will use methods from DBHandler to make changes to database
AccountDetails	Displays Account information to the user.	Account - Concept required a class to hold user data and to allow user data to be queried
AccountDetails	Stores account information for login and dietary restrictions.	Account - Concept required a class to hold user data and to allow user data to be queried
ThresholdWarning	Warns the user of ingredients that are below a certain amount in the Digital Pantry.	DigitalPantry - Class must contain the thresholds for all ingredients in the pantry THHandler - methods from this class will check and show warnings to the user for ingredients nearing the threshold.
DigitalPantry	Stores list of ingredient information for a user.	DigitalPantry - Class must contain the ingredient lists for users
DigitalPantry	Display list of ingredients saved for the user account.	DigitalPantry - Class must be able to display the ingredients in the list from the DigitalPantry concept
Ingredient	An ingredient object that tracks an ingredient for the user in their pantry.	Ingredient - Class to store the ingredient information accordingly.
IngredientManager	Add, edit or remove ingredients for temporary use in recipe generation or grocery list generation.	IngChecker - Users will likely attempt to add ingredients that are invalid or do not exist. This class is necessary to prevent the user from making harmful database changes through the ingredients.

		Ingredient - Will need this class to temporarily make the ingredient to be passed around.
GroceryList	Generates a grocery list for the user.	Ingredient - Now that ingredients from the digital pantry are being used in other concepts, we had a requirement to create a universal Ingredient class to be shared between concepts. GroceryList - List must contain Ingredient classes and Store classes to display for user
StoreIngredientFilter	Tracks the store ingredient filters.	StoreFinder - Used to find the nearest store with the ingredients in the grocery list. GroceryStore - The information contained in the grocery store that is found. StoreIngredientFilter - Helps sort the store ingredients by availability, name, price, ratings and reviews.
StoreIngredientFilter	Filters shown grocery stores based on user preferences.	StoreFinder - Used to find the nearest store with the ingredients in the grocery list. GroceryStore - Information is checked against StoreFinder to find matches
RecipeFilter	Tracks the recipe filters.	RecipeFilter - Since the class filters recipes, it has to intake the filters from the concept in order to search for matching parameters Recipe - The information held in this class will be matched against the filters
RecipeFilter	Filters recipes based on user preferences.	RecipeFilter - Holds the preferences Recipe - Matched against preferences RecipeFilter.filterRecipes() was created to compare the parameters from the two classes in order to find matches. SpoonacularHandler - Use this class to finally generate the recipes according to the filters using the Spoonacular API.

DigitalPantry	Used in the recipe generator to track usage of digital pantry ingredients in recipe generation.	RecipeGenerator - This class polls the digital pantry through DBHandler to find matching Recipe classes for the user.

Table 25: Traceability Matrix for Mapping Domain Concepts to Classes

This table maps line by line each domain concept to its relevant classes. Each class is taken into account and derives into a domain concept.

Design Patterns

For our ChefCart web app, we used the Publisher-Subscriber design pattern. The Publisher - Subscriber design pattern helps disassociate unrelated responsibilities, helps simplify/remove complex conditional logic, and allow seamless future adding of new cases. It also makes an event detector object tell multiple, decoupled event doer objects when events are available. The Publisher-Subscriber pattern is extremely useful because we can easily interact with our use cases. There is a user and the user can be the home chef or any other people who are interested in cooking. The user interacts with the user interface. The user interface interacts with the system and the system interacts with the database. The database contains the user's digital pantry, user information, and other important data. In a use case, the subscriber is the user who wants to either create the digital pantry, find recipes, create the grocery list, or edit other information. In a use case, the publisher is the database that holds all the information and the system that tells the interface what to do. In UC-2: Track Pantry, the user interacts with the system and the checker or Spoonacular APi also interacts with the system. The user and the checker do not interact with each other directly. Similar things will occur with the other use cases.

Another design pattern that we used was the object pool pattern. More specifically, we used the connection pool pattern. From the interaction diagram, we encapsulated the database handling object, or the DBHandler to be provided as a module. This allows many features to be able to access the database by simply using the module. However, the DBHandler features a connection pool to the database. This is ideal because many users can access ChefCart at the same time. This means that there may be a high traffic amount of requests to update the database. In that sense, we want to keep a number of connections, or a pool of connections to the database that can be constantly reused for any user. This allows for the concurrency of users to operate even with a large amount of traffic, so that speed and performance will not be a concern.

Object Constraint Language (OCL)

Below is a list of the Contracts in OCL for the most important use cases. These include logging in, showing the Digital Pantry, creating a Grocery List and finding a store and its items, and generating recipes.

<u>Login</u>
Context LoginService::login(loginController: LoginController): JWT
Inv: loginButton, JWTService, emailField, passwordField
Pre: username = emailField.text()
Pre: password = passwordField.text()
Pre: LoginService.LoginUser(username: String, password: String) = true
Post: result = WebServer.AuthUser()
Pre: username = emailField.text()
Pre: password = passwordField.text()
Pre: LoginService.LoginUser(username: String, password: String) = false
Post: Alert.alert("Invalid Credentials")

<u>DigitalPantry</u>
Context DigitalPantryService
Inv: addButton, WebServer, UserInfo
Pre: WebServer.AuthUser()
Post: WebServer.GetPantry()

Context DigitalPantryService::WebServer.GetPantry(): List of Ingredient
Inv: UserInfo
Pre: WebServer.AuthUser()
Post: DigitalPantryService.Display(ingredients: List of Ingredient): void

<u>GroceryList</u>
Context GroceryListService
Inv: addButton, WebServer, UserInfo
Pre: WebServer.AuthUser()
Pre: WebServer.GetPantry()
Post: result = List of Ingredient where ingredient.Quantity < ingredient.Threshold

Context GroceryListService::ClosestStore(city: String, state: String): (StoreInfo,
List of Ingredient)
Inv: UserInfo, List of Ingredient
Pre: WebServer.AuthUser()
Post: DigitalPantryService.Display(ingredients: List of Ingredient): void
Post: DigitalPantryService.Display(storeInfo: StoreInfo): void

<u>RecipeGenerator</u>

Context RecipeGeneratorService
Inv:WebServer, UserInfo, additionalIngredientsField, cuisinesField, intolerancesField, dietsField
Pre: WebServer.AuthUser()
Pre: WebServer.GetPantry()
Post: RecipeGeneratorService.GenerateRecipes(ingredients: List of String,
    additionalIngredients: List of String, diets: List of String,
    intolerances: List of String, cuisines: List of String): List of Recipe
Context RecipeGeneratorService
Inv: addButton, WebServer, UserInfo
Pre: WebServer.AuthUser()
Pre: WebServer.GetPantry()
Post: RecipeGeneratorService.GenerateRecipes(ingredients: List of String,
    additionalIngredients: List of String, diets: List of String,
    intolerances: List of String, cuisines: List of String): List of Recipe

Context RecipeGeneratorService::GenerateRecipes(ingredients: List of String,
    additionalIngredients: List of String, diets: List of String,
    intolerances: List of String, cuisines: List of String): List of Recipe
Inv:WebServer, UserInfo, additionalIngredientsField, cuisinesField, intolerancesField, dietsField
    SpoonacularAPIURL
Pre: WebServer.AuthUser()
Pre: WebServer.GetPantry()
Post: Router.POST(endpoint: String, function(context: Context)): List of Recipe

System Architecture and System Design

UML Package Diagram

ChefCart utilizes different APIs to develop different features. Using the domain routing logic, the results of each of the Digital Pantry, Recipe Generator, and the Grocery List features will be displayed to the user in the Client GUI. Feature pairs such as the Digital Pantry and the Recipe Generator, the Grocery List and the Digital Pantry and the Recipe Generator need to communicate with each other. A grocery list needs to know what is in the pantry and so does the recipes generator. Each feature is connected to the API they need to call and the database to retrieve ingredients used in the Digital Pantry. Our web server, database and client GUI are separated so that each feature can be easily tested modularly.

UML Package Diagram

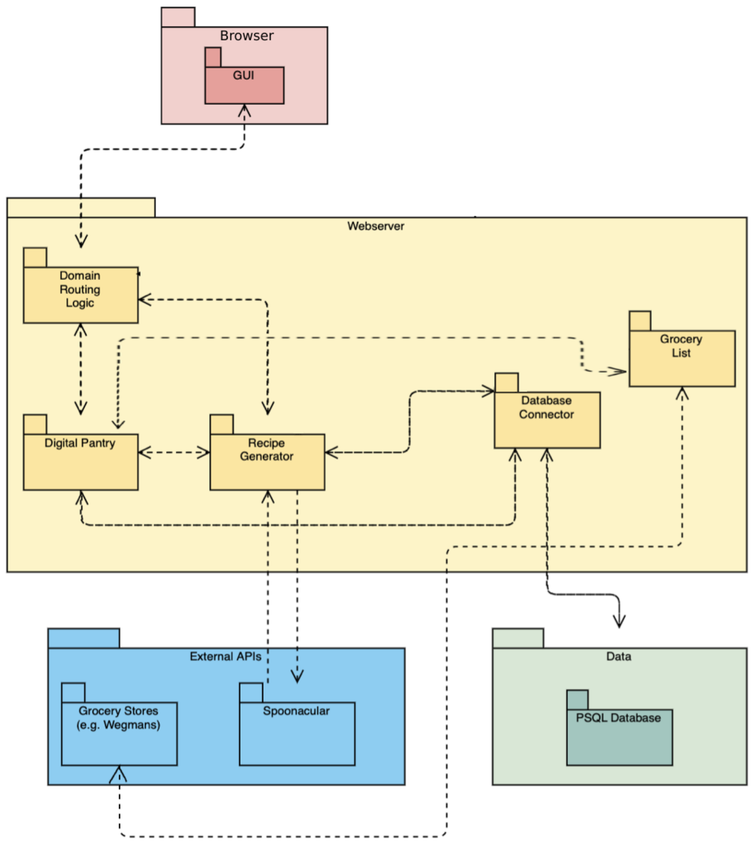


Figure 7: UML Package Diagram

Figure 7 is the UML Package Diagram. Important high level features are represented here. These include external APIs, the database, the webserver and its features, and the browser.

Architectural Styles

In our design, we utilize two main architectural styles:

1. Event-driven
2. Model-view-controller.

The event-driven architecture is commonly used in modern applications. It is a software architecture and model for application design. It consists of three parts: event producer, event router, and event consumer. In the event producer, the user inputs and updates information through our web application. Then, the data will be sent to the event router. The event router will filter and push the event to the appropriate consumer. In the event consumer, our application will generate (recipes, stores), update, remove, or add content (ingredients, user's personal information) depending on the user input.

The model-view-controller is commonly used for developing the user interface. It is an application design model composed of three interconnected parts. The three parts are model, view, and controller. In the view section, the users are able to view information such as ingredients, recipes, and personal information depending on what page they are on. After the user inputs, update or add/remove ingredients or other information, the controller will update the model and view accordingly. The model contains the data, mainly ingredients, used by the application.

Mapping Subsystems to Hardware

In our application, there are three subsystems.

1. Website Hosting
2. Database Content
3. Client Access

All three subsystems need to run on multiple computers. The first subsystem, Website Hosting, will be using an external server like Amazon Web Services (AWS) and Heroku. AWS and Heroku are the two most commonly used cloud services that allow us to deploy, monitor, and scale web apps. The second subsystem, Database Content, is hosted on an external database like PSQL and will be hosted on a server to be accessible by ChefCart. The third subsystem, Client Access, will allow the clients to access their personal computer or mobile device.

Connectors and Network Protocols

Our team will be implementing a REST API to server HTTP requests. This is common in the modern industry when designing web applications with clients and servers. HTTP will help us separate the client from the server and will allow for the client-side to send requests every now and then from the server so that it can return the appropriate information for the client-side to use. This makes sense because the user will be manipulating the digital pantry only ever so often, whose update information is passed through as an HTTP request to propagate the change in our database. In addition, we will use protobuf messages or Protocol Buffers in order to send messages to pass information between the client and the server. Since our intended language is going to be Go, this method is handled well in Go and is well documented for it. Protobuf messages allow flexible creation and receiving of messages as data can be directed into any of several data structures of our choosing.

Global Control Flow

Execution Orderness:

ChefCart is largely event-driven rather than procedure-driven. Some aspects of ChefCart will be linear, but each user can generate their actions in different orders. In particular, all users must login with their account before accessing the system features. Once logged in, users can then choose any of the features whether it be the digital pantry, recipe generator, or the grocery list pages. Users have access to each of these features in a nonlinear fashion. In addition, when manipulating the digital pantry, the user may choose to add, edit or delete ingredients in any order of their choosing. In each of the recipe generator and the grocery list pages, the user must go through these features in a linear fashion. When using the grocery list feature, they first must generate their grocery list before looking at potential stores and they must click to search ingredients in a store before they can view its detailed information. As for the recipe generator, a user must input ingredients before they can start searching for recipes that mostly use them and they must click on a recipe before being able to view the recipe instructions.

Time Dependency:

ChefCart does not have timers in its system. However, some of the aspects of ChefCart change based on external APIs. For example, when calling the grocery store API to retrieve available ingredients that a user might buy, the price of the ingredients and the stock is dependent on the state of the store at that time. Other than that, there are no timers.

Hardware Requirements

ChefCart utilizes a database to store data and uses Wi-Fi to have the database communicate with the end-user client. Users should be able to access ChefCart by running it on iOS 10 and above, Android 8.0 Oreo and above, Windows 7 and above, and MacOS Catalina (10.15) and above as well as the major Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari browsers. These will cover most devices on the market today. Our application requires a color display with a minimum resolution of 1330 x 700 pixels and will work up to 1920 x 1080 pixels. In order to communicate with our servers, users need a 1 Mbps download and upload speed. Users will not need storage on their device to access ChefCart as it will be a web application. As long as the user has Wifi, they can access the web app. The requirements of the database storage space will expand with the number of users and the server requirements will not need to scale since we are not opting for microservice architecture.

Algorithms and Data Structures

Algorithms

We did not use any mathematical models in Report #1.

For any user who wants to make their own digital pantry, they can input ingredients into the digital pantry. The algorithm will keep track of the digital pantry for the user. The user has three options: update, remove, and add ingredients. The user can decide to add, remove, or update the ingredient in the user interface. If the user chooses to update or add a new ingredient, then the user will input the ingredient name, weight, and volume in the corresponding text field. There will be a method that uses the Spoonacular API to check if the ingredient is appropriate/correct. Also, there will be a method that checks if the user input a positive value or 0 (n/a, do not know) for the quantity, weight, volume. This method is used whenever an ingredient is added or updated. If any of the input is incorrect, the ingredient will not be stored and a warning will be returned to the user. If all of the inputs are correct, the ingredient will be stored in the PostgreSQL database. When the user chooses to delete an ingredient, the system will go in the PostgreSQL database and remove the ingredient.

When the user wants to find a recipe to use, the user can use the find recipe. This algorithm will generate a list of recipes and display them to the user. The user interface will ask the user to input some values in the text field so that the system can generate the recipes. The user interface will ask for the ingredients that the user wants to use to generate the recipes. The ingredients can be from the digital pantry, manually input, or both. If the user chooses to use the digital pantry, the system will find the ingredients that are stored in the PostgreSQL database. If the user decides to manually input the ingredients, the system will use the Spoonacular API to associate the user's text inputs with Spoonacular ingredients (invalid inputs will simply be ignored). The user can also choose to filter the recipes. The filters include diets, intolerances, and specific cuisines. There will be no method to check the diets, intolerances, and specific cuisines because it will be a drop down. The user is unable to input an invalid input. Once the user clicks the submit button to generate recipes, the system will use the Spoonacular API to generate the recipes that use all of the ingredients. There is also a method that gets the user's dietary restriction(s), which is stored in the PostgreSQL database, and prevents any inappropriate recipe from getting output to the user by using the dietary restriction(s) to check if any of the recipes have ingredients that should not be used. The algorithm will first generate the recipes by comparing each ingredient to the recipe ingredient. Then, if the user chooses to use the filter, the algorithm will filter out the unnecessary recipes by comparing the user's filer with recipe fields: cuisines, diets, and intolerance.

The user would want to buy more ingredients when running out of ingredients. The algorithm creates the grocery list and finds the closest store for the user to purchase the ingredients and display them to the user. There will be a method that uses the digital pantry to check if the ingredients are below a certain threshold. If an ingredient is below the threshold, it will then be added to the grocery list automatically. The user can choose to add additional ingredients to the list. There will be a method that checks if the user inputs the correct ingredient. If the user decides to manually input the ingredients, then the method will call the Spoonacular API to check the user's text inputs with Spoonacular ingredients. A warning will appear if the user input is invalid. The user can click on the delete button to remove the ingredient from the list. Since the grocery list is a one time use, the data will not be stored in the PostgreSQL database. Once the user clicks on the search button, the system will find and display the store to the user. There will be a method that calculates the distance from the user's location to the store and then it will compare all the local store distances to see which store is the closest. Afterward, the system will display the store information and different ingredient brands that the user can choose from. After clicking on the specific ingredient brand that the user wants, the system will send the user to the store website to purchase the ingredient.

The system can serve multiple users concurrency easily because it is a web app. Any user with the link can access the web app. All the backend database and api calls are handled on a pre-request basis. So if for some reason we had insanely high load there would be a bit of delay. For future iterations we would move to a micro service architecture in order to prevent this issue.

Data Structures

ChefCart uses complex data structures as part of its many functions and features. Arrays are used commonly to store local memory on the client side for displaying data to the user. For example, we would need an array to display all the ingredients that a user has in the digital pantry. Situations where we need to use an array for displaying include the digital pantry, grocery list ingredients, recipe ingredients, grocery stores and generated recipes. These arrays will be either populated by the database, our APIs (both spoonacular and stores), or generated on the web application itself for temporary use. In order to help us process and store the data nicely, we use the classes mentioned in the UML class diagrams. These classes store necessary data for ease of use when coding the functionality to display results to the user. Other features that use arrays are when we display the list of Store Ingredients that the user can shop from and the list of cuisines, dietary preferences and intolerances that the user wants to use to filter out recipes when generating recipes. Many of these data structures are coded in Go, so we use Go's structs to act as our classes in ChefCart to implement the UML class diagrams.

In order to send information to be passed as parameters to external APIs such as Spoonacular and store APIs such as Walmart, we need to be able to package the information. Object relational mapping, or ORM is used to convert the ingredient data stored in the databases to Go structs. Then, these structs need to be formatted into Protocol Buffer messages to be passed to our server side methods via gRPC, which is Go's implementation of remote procedure calls. These will then be used to communicate with the external APIs. In addition, results are sent back to the client side with the same Protocol Buffer messages.

We decided to use arrays because the data that we are showing to the user are simply lists that need to be sorted and filtered. As we are not dealing with particularly large datasets, the simplicity and flexibility of arrays are sufficient. We do not need the extra performance that other data structures provide. We decided to use an RPC framework to offload client side processing to server side processing instead. By doing so, we needed to use Protocol Buffer messages for communicating in Go to make the full use of this possibility.

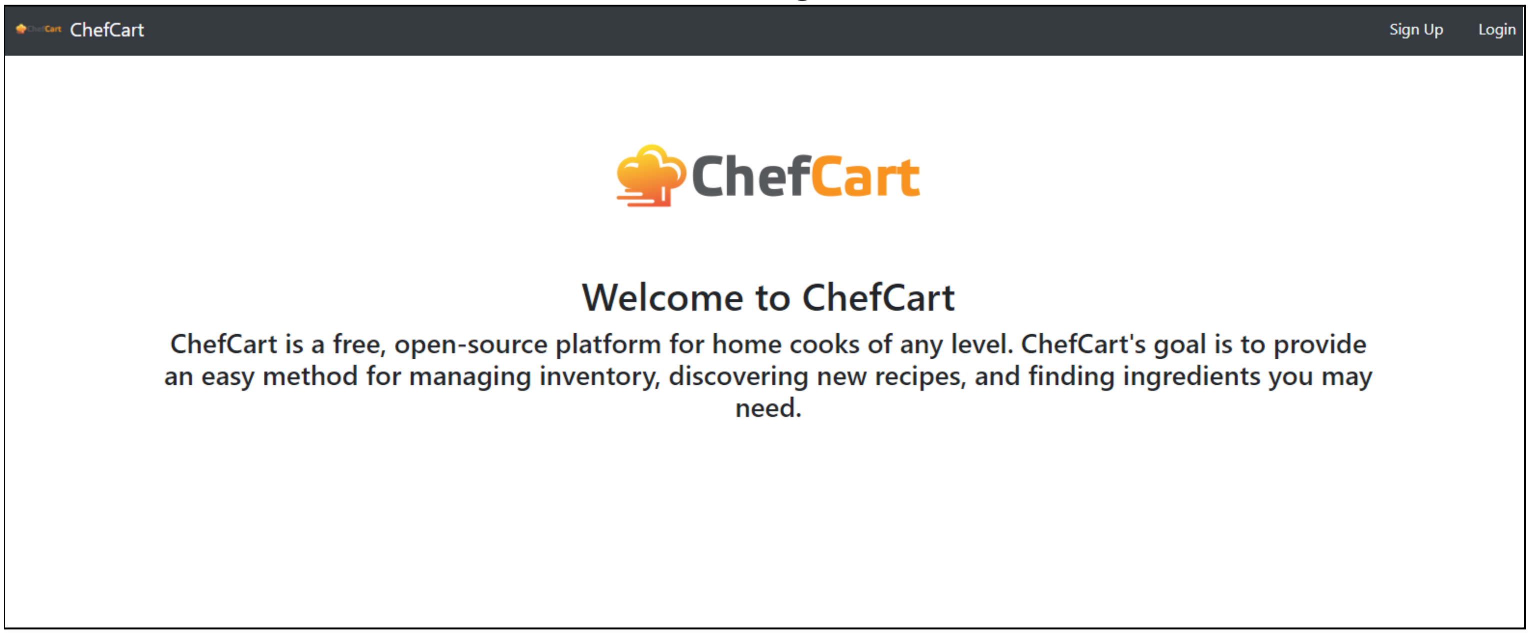
Concurrency

Multiple concurrent users are still handled in ChefCart. Many users can login at a time and still be able to use the application. Each client can be considered a separate thread. The server side deals with requests from each client as they come in. Each client request is processed and dealt to serve the client on a need by need basis. Each web page that is being used by every user that calls the backend database and makes api calls will be handled on a per-request basis. There may be a little bit of delay for high loads. In the future, we would like to prevent this issue by using a micro service architecture.

User Interface Design and Implementation

Account Handling subproject:

Main Page



Welcome to ChefCart

ChefCart is a free, open-source platform for home cooks of any level. ChefCart's goal is to provide an easy method for managing inventory, discovering new recipes, and finding ingredients you may need.

Figure 15: UI Design of Main Page

Description:

- This is the Main Page where ChefCart welcomes the user and the user may choose to sign up for a new account or login to an existing account.
- If the user does not have a ChefCart account, then the user will click on the Sign up button to create an account. After clicking the button, it will lead the user to the Sign Up Page.
- If the user does have a ChefCart account, then the user will click on the Login button to login the account. After clicking the button, it will lead the user to the Login Page.

Sign Up Page

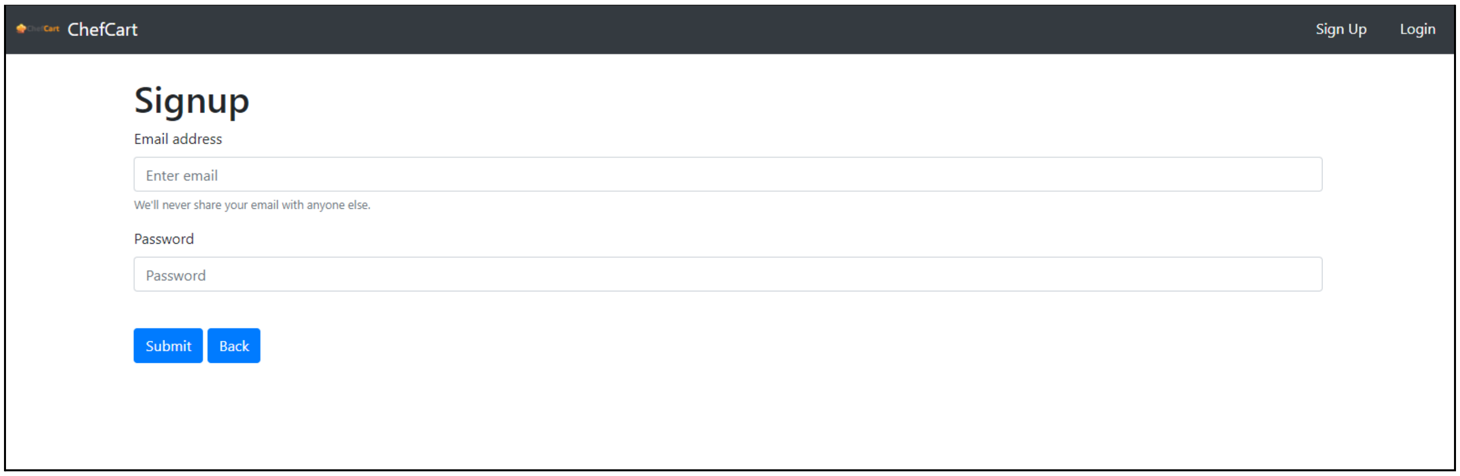


Figure 16: UI Design of Sign Up Page

Description:

- This is the Sign Up Page where the user can create a new account.
- When creating a new account, the user is prompted to enter a username and password. This information will be stored in the database for when the user logs in a separate time. Upon clicking the Submit button, the user will be logged in and can begin using ChefCart.
- If the user clicks the back button, then it will allow the user to go back to the Main Page.
- If the user clicks the Login button, then it will allow the user to go to the Login Page.

Login Page

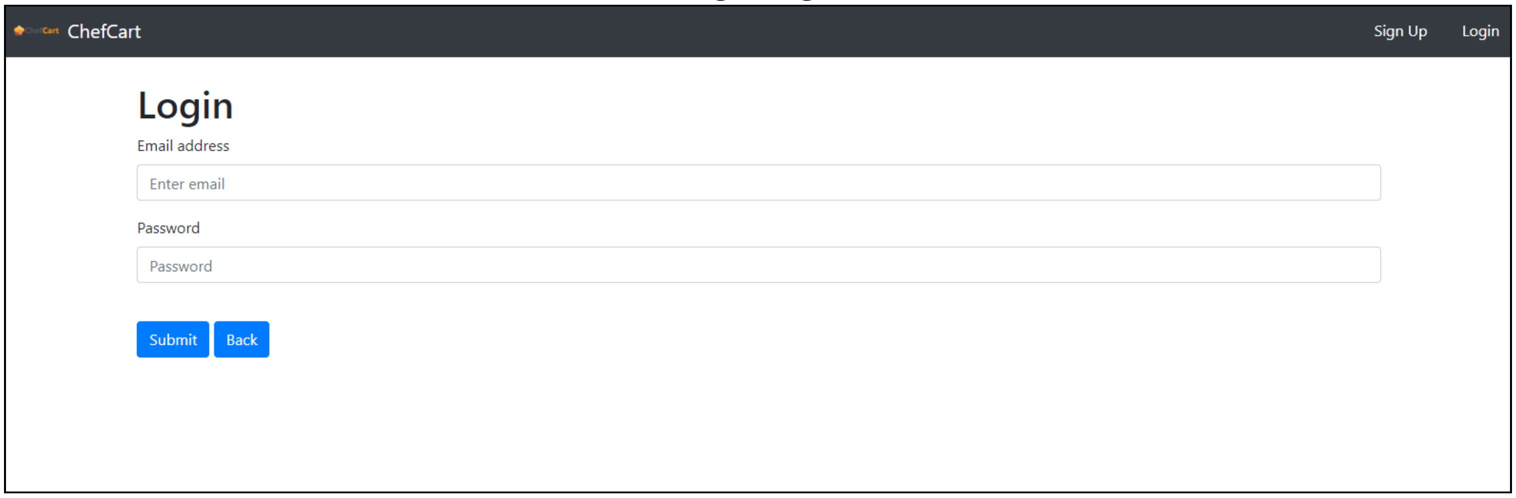


Figure 17: UI Design of Login Page

Description:

- This is the Login Page where the user can login to access ChefCart functions.
- If the user will input their username which will be their email and password in the respective fields. After inputting the username and password, the user will click the Submit button to sign in. It will lead the user to the Home Page.
- If the user clicks the Back button, then it will allow the user to go back to the Home Page.
- If the user clicks the Sign Up button, then it will allow the user to go to the Sign Up Page.

Home Page

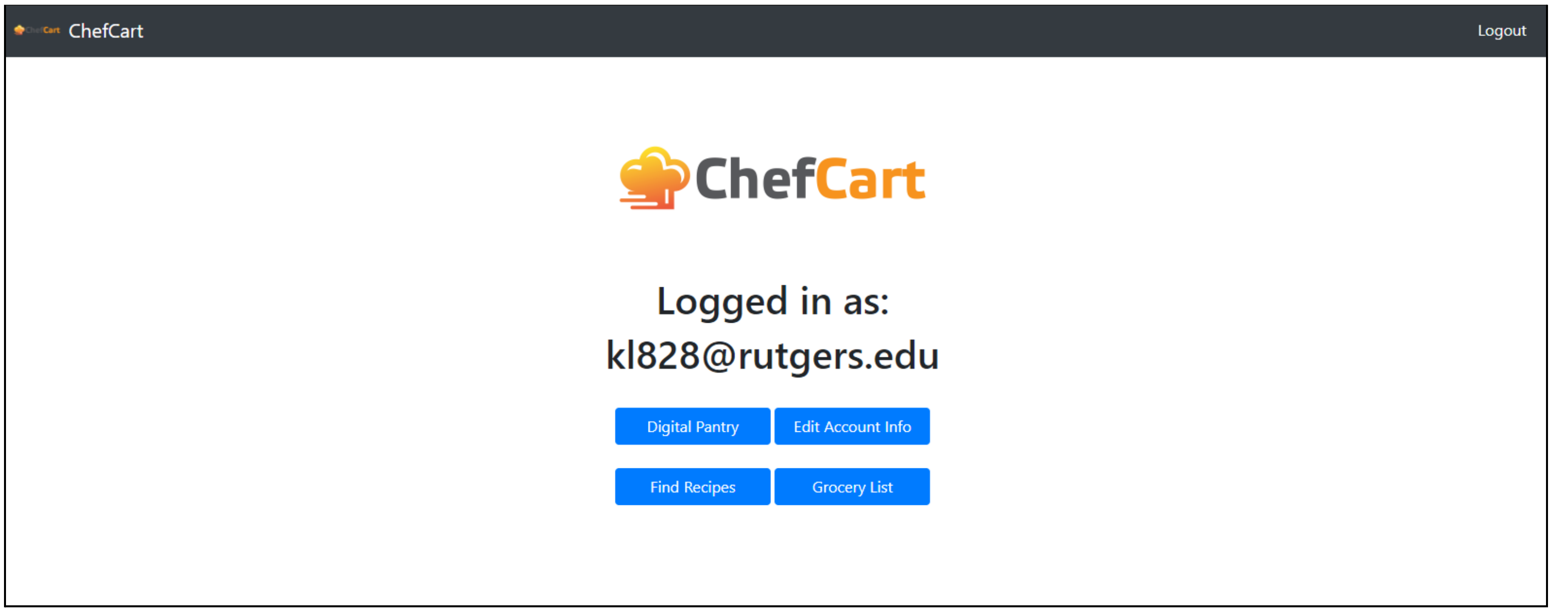
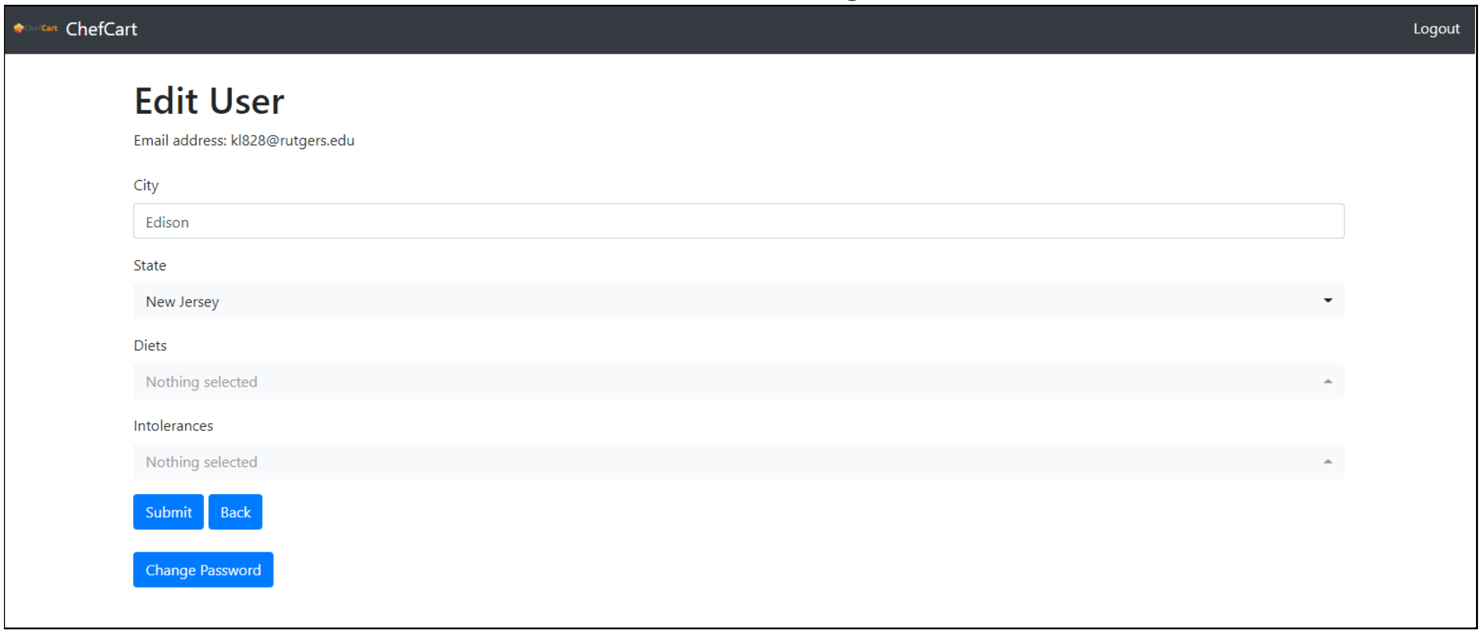


Figure 18: UI Design of Home Page

Description:

- This is the Home Page upon login where the user can navigate to each of ChefCart's features. Instead of having four tabs where the user can navigate between each function, we decided to leave that navigation to this main page instead.
- There are four buttons that the user can click on.
  - Digital Pantry button will lead to the Digital Pantry Page
  - Find Recipes button will lead to the Grocery List Page
  - Edit Account Info button will lead to the Account Page
  - Grocery List button will lead to the Grocery List Page
- If the user clicks on the Logout button, then the user will be directed to the Main Page.

Edit Account Page



Edit User

Email address: kl828@rutgers.edu

City: Edison

State: New Jersey

Diets: Nothing selected

Intolerances: Nothing selected

Figure 19: UI Design of Edit Account Page

Description:

- When in the Account Page, the user can edit their accounts. The Account Page consists of:
  - City - The user can change their city by inputting a new city.
  - State - The user can click on the drop down icon and a list of states will appear. The user can change their state by clicking on one of the states in the list.
  - Diets - The user can click on the drop down for adding any diets that they would like to have satisfied during the use of ChefCart.
  - Intolerances - The user can click on the drop down menu for adding any intolerances that they would like to have satisfied during the use of ChefCart.
- The user can save all the account changes by clicking on the Submit button.
- If the user clicks on the Back button, then the user will go back to the Home Page.
- If the user clicks on the Change Password button, then the user will go to the Change Password Page.
- If the user clicks on the Logout button, then the user will be directed to the Main Page.

Change Password Page

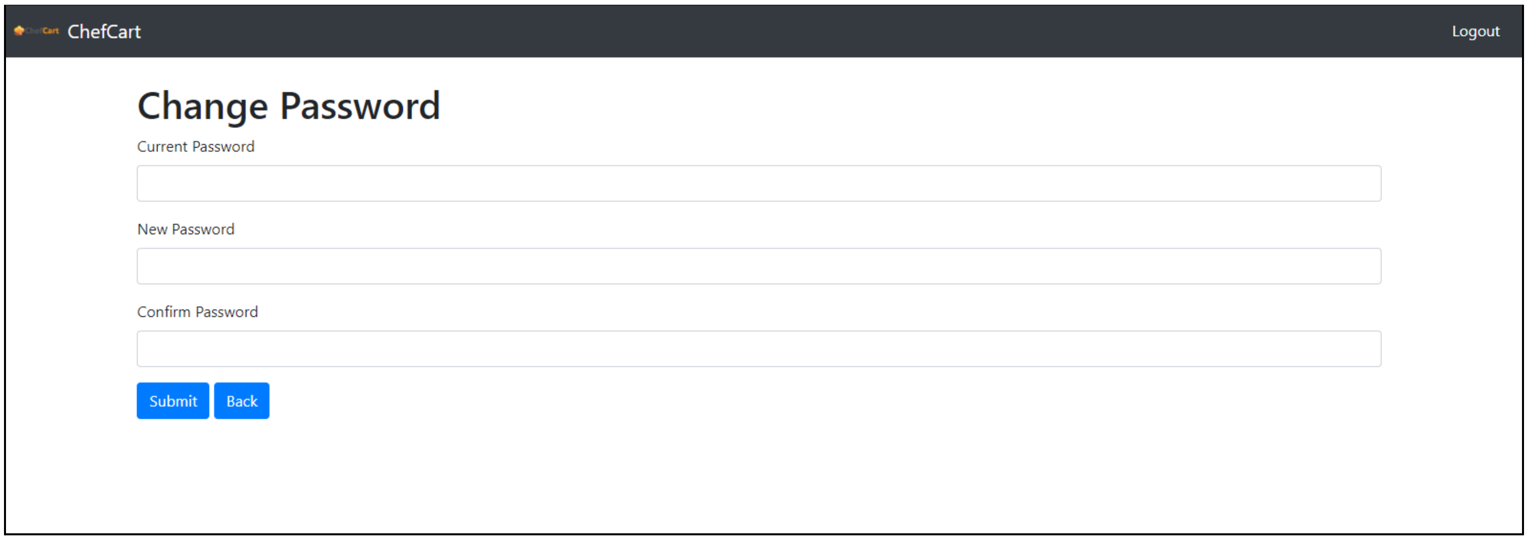


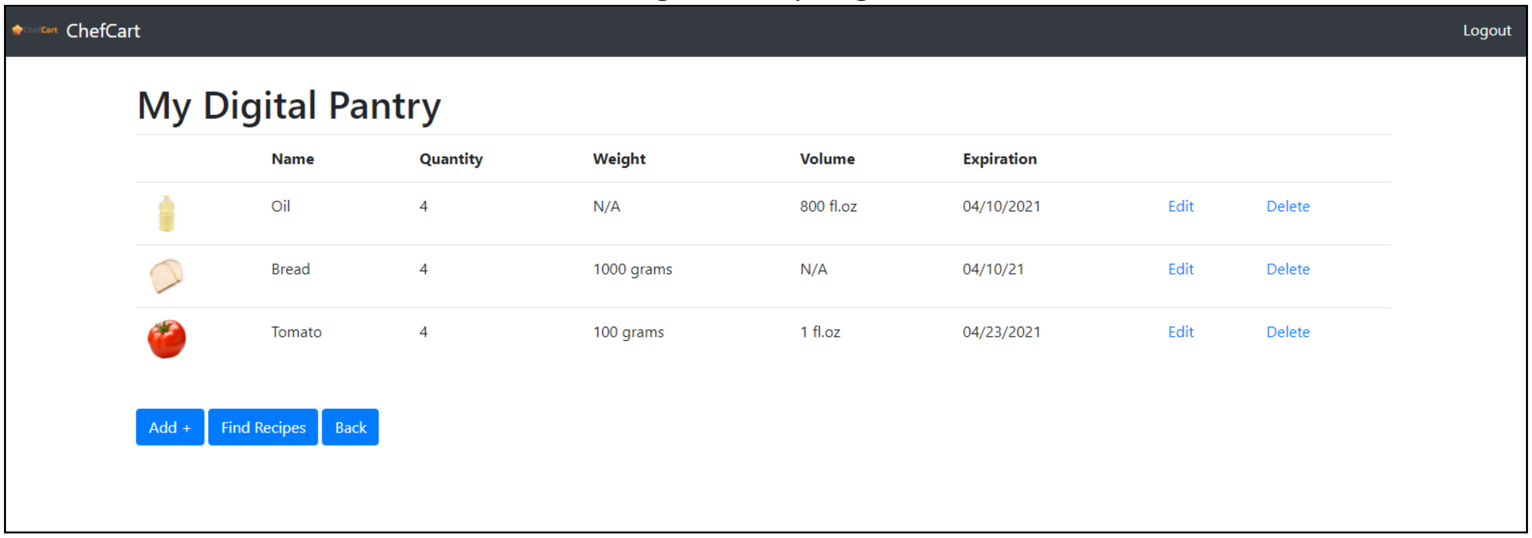
Figure 20: UI Design of Change Password Page

Description:

- When changing the password, the user is prompted to to enter the current password, new password, and confirm password. This information will be stored in the database for when the user logs in a separate time. Upon clicking the Submit button, the new password will be saved.
- If the user clicks on the Back button, the user will go back to the Home Page.
- If the user clicks on the Logout button, the user will be directed to the Main Page.

<u>Digital Pantry subproject:</u>

Digital Pantry Page


ChefCart

Logout

My Digital Pantry

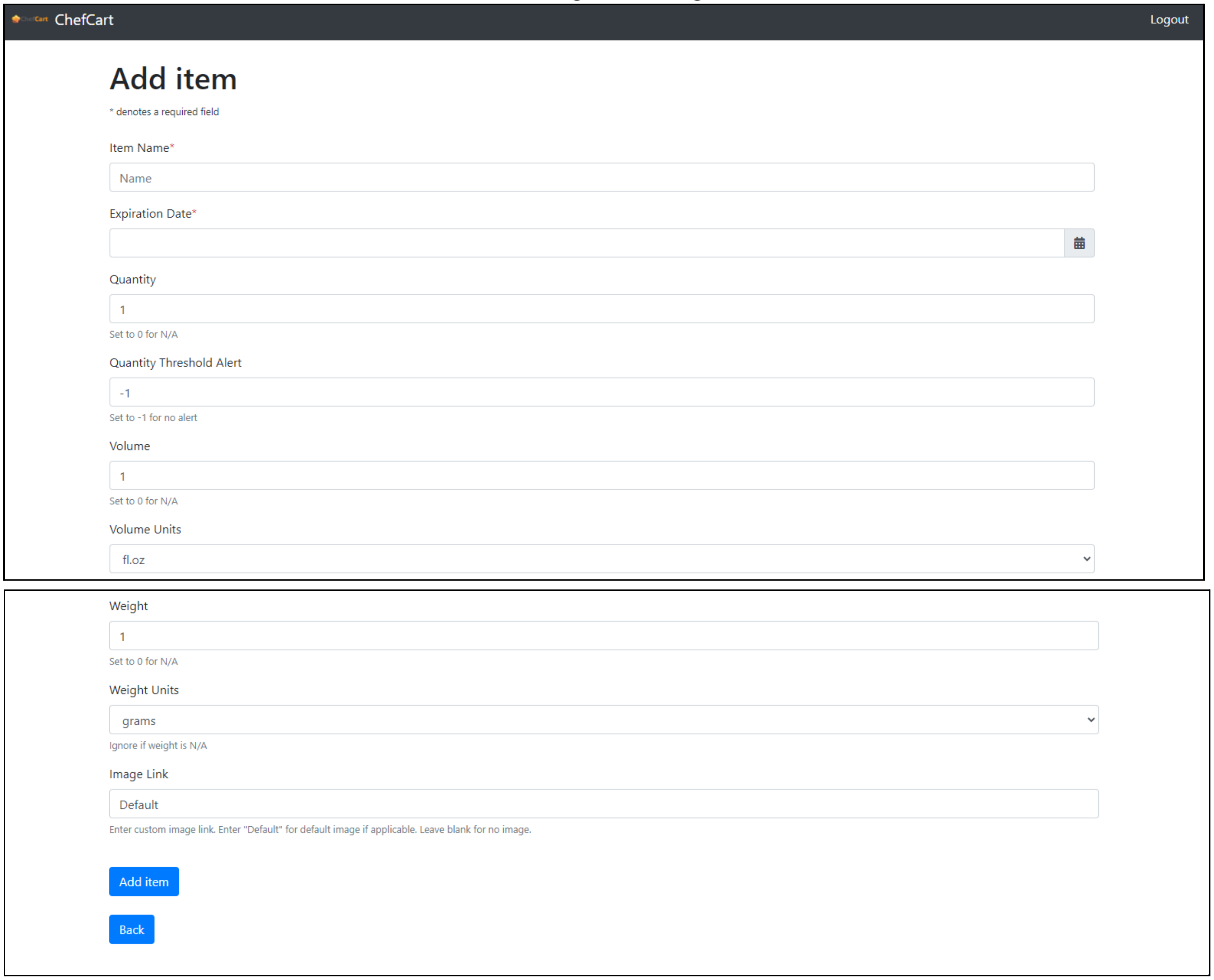
	Name	Quantity	Weight	Volume	Expiration		
	Oil	4	N/A	800 fl.oz	04/10/2021	Edit	Delete
	Bread	4	1000 grams	N/A	04/10/21	Edit	Delete
	Tomato	4	100 grams	1 fl.oz	04/23/2021	Edit	Delete

Figure 21: UI Design of Digital Pantry Page

Description:

- When in the Digital Pantry Page, the user will be able to see what is in their digital pantry along with key information: ingredient name, quantity, weight, volume, and expiration date.
- In the Digital Pantry Page, the user can click on …
  - The Add+ button opens up the Add Item page, it allows users to manually add ingredients and information. Once the ingredient is added, it will go back to the Digital Pantry Page and the Digital Pantry Page will refresh, showing the new ingredient(s) included.
  - The Find Recipes button opens the Find Recipe Page.
  - The Edit button for each existing ingredient allows users to edit the amount of said ingredient in the Digital Pantry.
  - The Delete button for each existing ingredient allows users to delete the said ingredient from the Digital Pantry.
- If the user clicks on the Logout button, the user will be directed to the Main Page.

Add Ingredient Page



ChefCart

Logout

Add item

* denotes a required field

Item Name*

Name

Expiration Date*

Quantity

1

Set to 0 for N/A

Quantity Threshold Alert

-1

Set to -1 for no alert

Volume

1

Set to 0 for N/A

Volume Units

fl.oz

Weight

1

Set to 0 for N/A

Weight Units

grams

Ignore if weight is N/A

Image Link

Default

Enter custom image link. Enter "Default" for default image if applicable. Leave blank for no image.

Add item

Back

Figure 22: UI Design of Add Ingredient Page

Description:

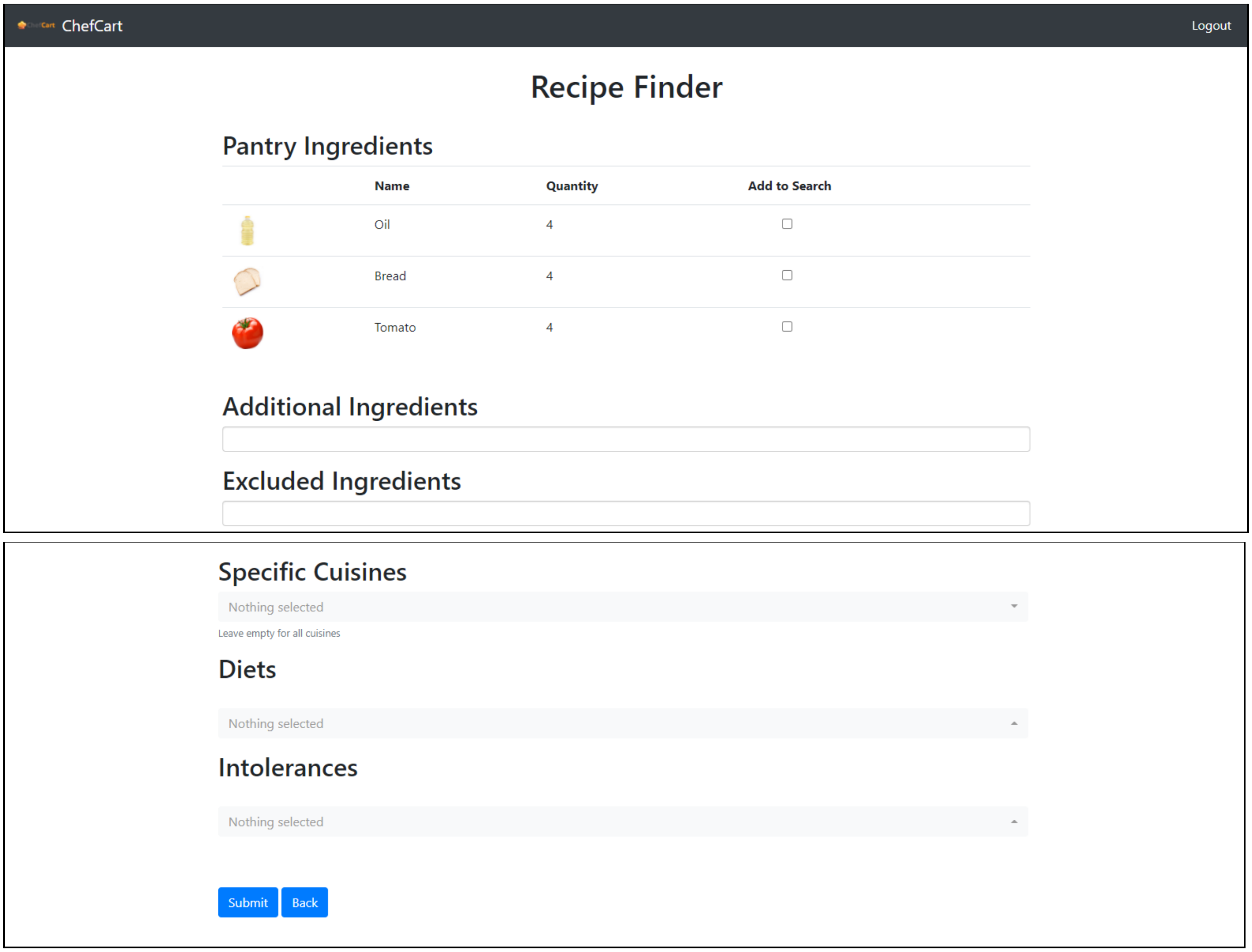
- The Add Ingredient Page is where the user can enter the fields for a new ingredient.
- In this page, the user can input a new ingredient's fields.
  - Item Name - The user enters the name of the ingredient.
  - Quantity - The user enters the quantity of the ingredient they have in stock.
  - Volume - The user enters the volume of the ingredient they have in stock.

- Volume Units - The user enters the unit of volume for the ingredient.
- Weight - The user enters the weight of the ingredient they have in stock.
- Weight Units- The user enters the unit of weight of the ingredient.
- Expiration Date - The user enters the expiration date of the ingredient using a date picking interface.
- Image link - The user enters a link for the image of the item that would like to view it as. If left as default, it will select an image for the user.

- Once the desired fields are entered, the user can either add the ingredient by clicking on the Add Item button or cancel by clicking the back button.
- If the user clicks on the Logout button, the user will be directed to the Main Page.

<u>Recipe Generator subproject:</u>

Find Recipe Page


ChefCart

Logout

Recipe Finder

Pantry Ingredients

	Name	Quantity	Add to Search
	Oil	4	☐
	Bread	4	☐
	Tomato	4	☐

Additional Ingredients

Excluded Ingredients

Specific Cuisines

Nothing selected

Leave empty for all cuisines

Diets

Nothing selected

Intolerances

Nothing selected

Submit Back

Figure 23: UI Design of Find Recipe Page

Description:

- The Find Recipe page shows the user what ingredients to include when discovering recipes.
- The user can check or uncheck various existing ingredients in the Digital Pantry signifying if they want recipes that use the selected ingredients. Interacting these ingredients keeps the user in the Find Recipe Page.

- The user can also specify additional ingredients by typing in a designated text field. All inputs are allowed; invalid food types will simply be ignored when searching for recipes.

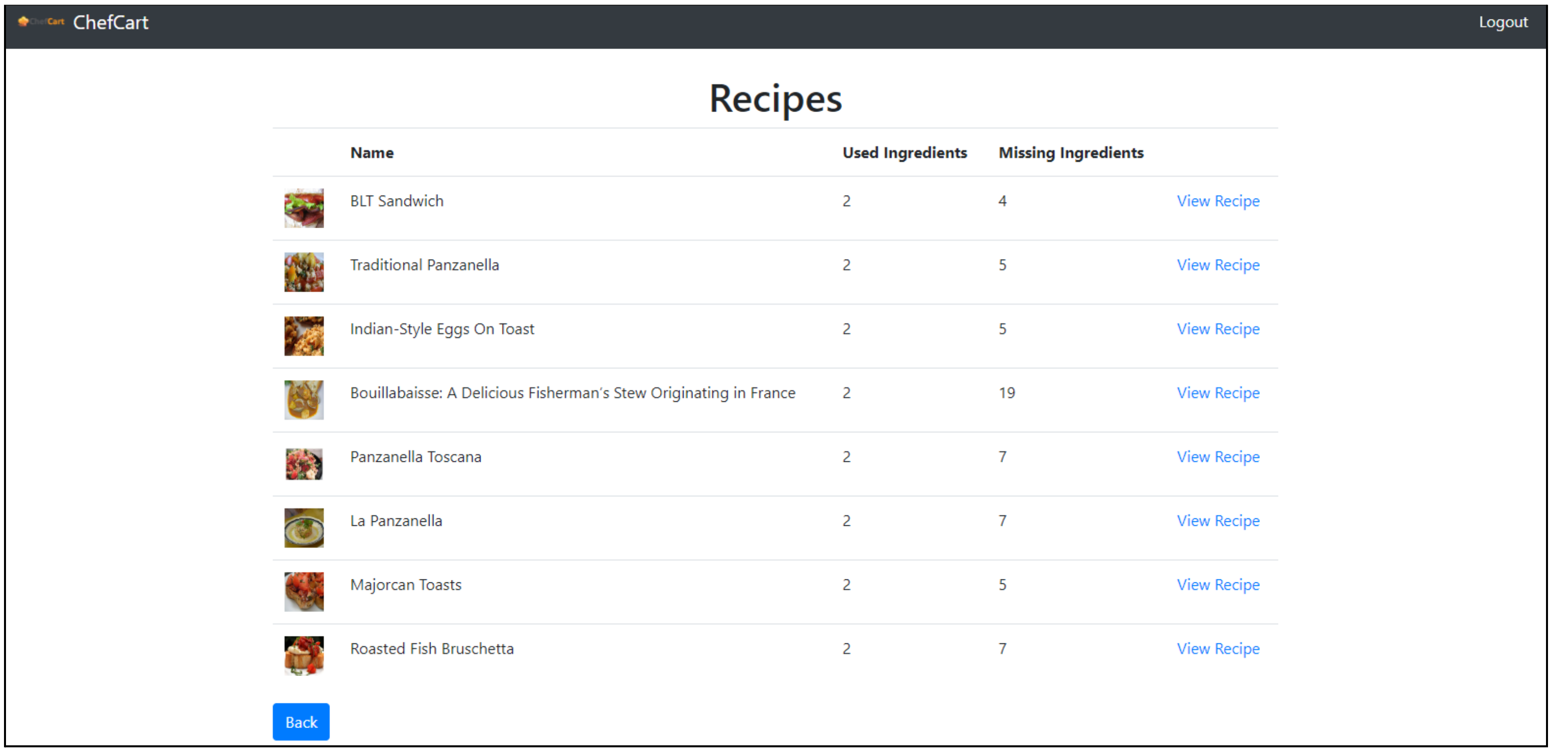
- The user can also select from a list of cuisines, diets and intolerances to further specify the recipe search. If no options are selected, the generator will output all recipes that use all of the ingredients.

- Once the user has entered all the information, the user can click the Submit button to go to the resulting recipes list on the Recipes Page.

- The user can click the Back button to go back to the previous page (this could be the Main Page or the Digital Pantry page).

- If the user clicks on the Logout button, the user will be directed to the Main Page.

Recipes Page



Recipes

	Name	Used Ingredients	Missing Ingredients	
	BLT Sandwich	2	4	View Recipe
	Traditional Panzanella	2	5	View Recipe
	Indian-Style Eggs On Toast	2	5	View Recipe
	Bouillabaisse: A Delicious Fisherman's Stew Originating in France	2	19	View Recipe
	Panzanella Toscana	2	7	View Recipe
	La Panzanella	2	7	View Recipe
	Majorcan Toasts	2	5	View Recipe
	Roasted Fish Bruschetta	2	7	View Recipe

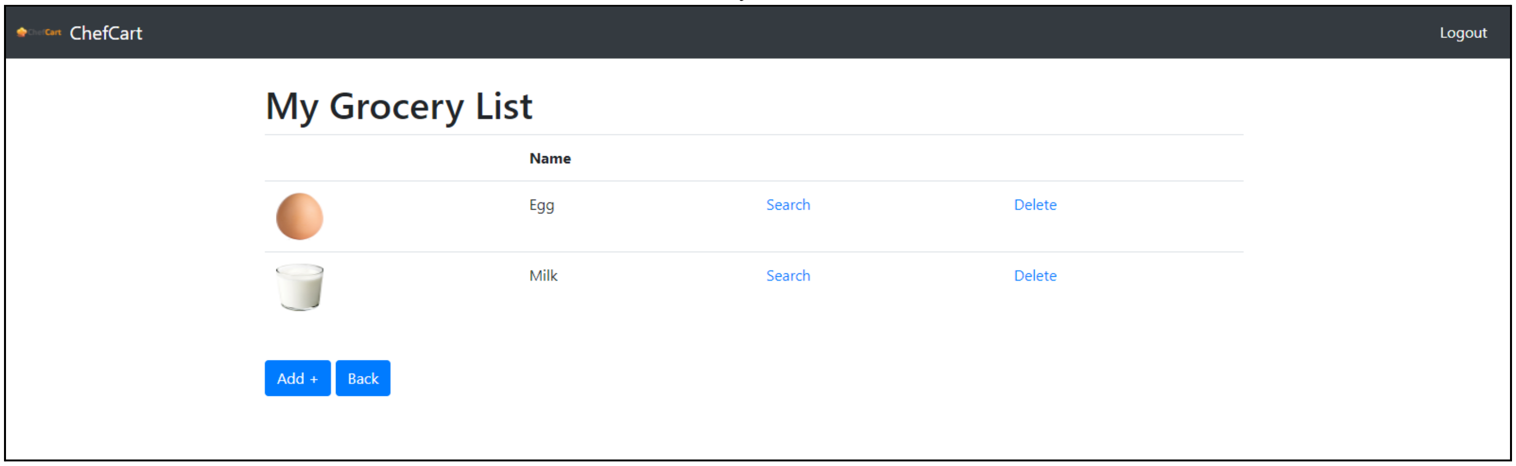
Figure 24: UI Design of Recipes Page

Description:

- The user can view the list of recipes available to them. The list contains the recipe name, used ingredients, and missing ingredients.
- By clicking on the View Recipe, it will display the recipe information to the user.
  - This action will send the user to the Spoonacular website to view the recipe.
  - Due to the limited amount of Spoonacular API calls, we will only be providing the user number of ingredients used and missing.
  - Information such as instructions, price and prep time are provided by Spoonacular.
- The user can click on the name, used ingredient, or the missing ingredients header to sort the recipes. It can be ascending or descending order.
- By clicking on the Back button, it allows the user to return back to the Recipe Generator when they click it.
- If the user clicks on the Logout button, the user will be directed to the Main Page.

<u>Grocery List subproject:</u>

Grocery List



My Grocery List

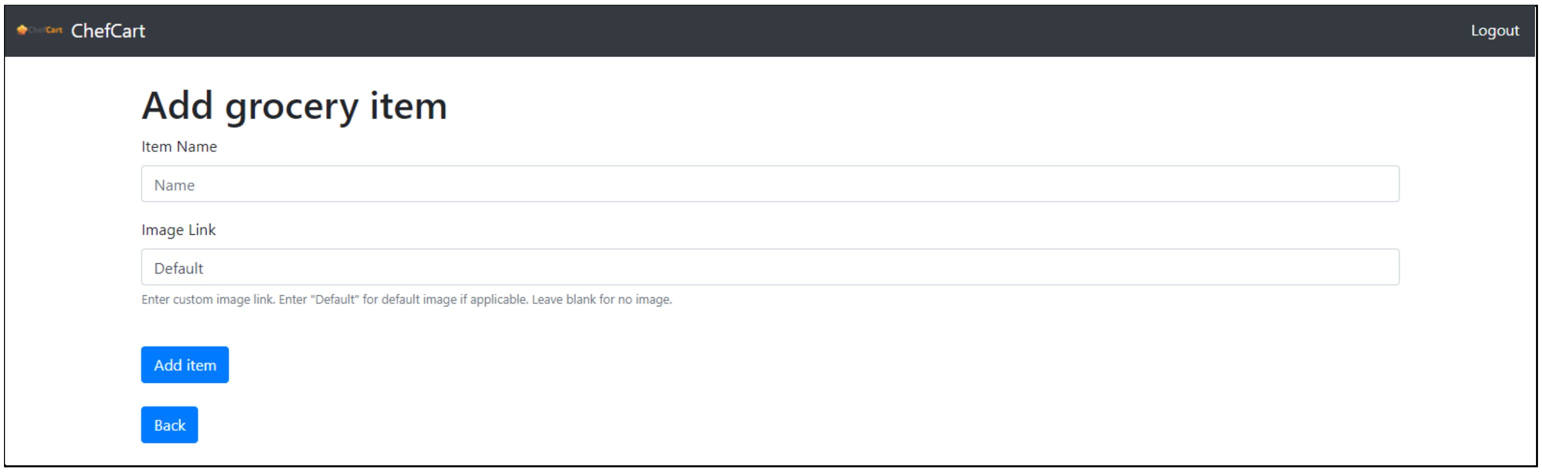
	Name		
	Egg	Search	Delete
	Milk	Search	Delete

Figure 25: UI Design of Grocery List Page

Description:

- When in the Grocery List, the user will be able to see what is in their grocery list.
- The user has three main actions in the Grocery List:
  - By clicking on the Add Ingredient button, it allows the user to go to the Add Grocery Item Page and users can manually add the ingredients to the list. Once the ingredient is added, the user will be back at the Grocery List Page and the grocery list will be refreshed, showing the new ingredient(s) included.
  - The Delete button allows the user to manually delete an ingredient from their grocery list. Upon clicking the Delete button, the ingredient will be deleted and the grocery list will refresh.
  - By clicking on the Search button, it will lead to the Search Result Page. (Shown below)
- If the user clicks on the Logout button, the user will be directed to the Main Page.

Add Grocery Item Page



Add grocery item

Item Name

Name

Image Link

Default

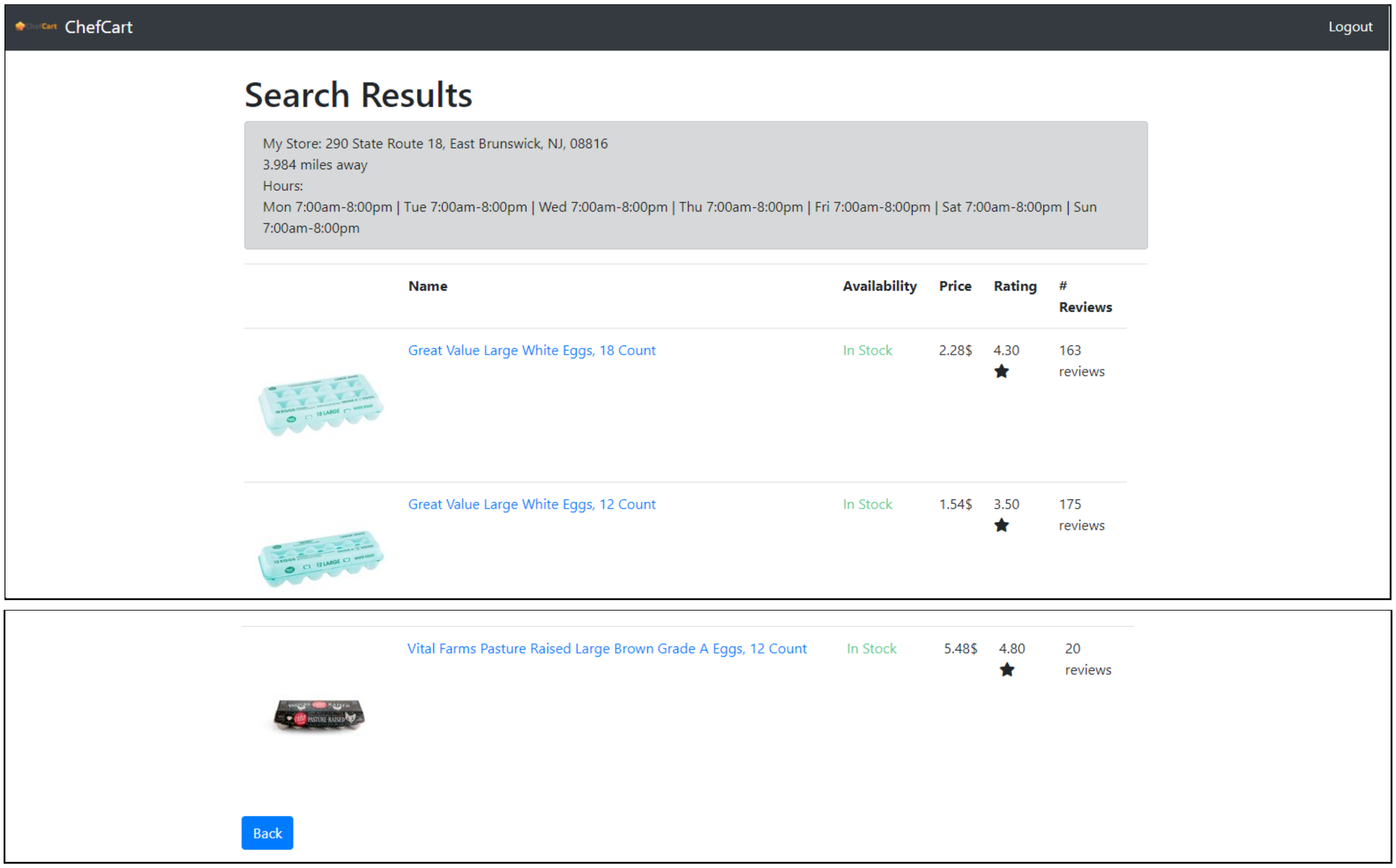
Enter custom image link. Enter "Default" for default image if applicable. Leave blank for no image.

Figure 26: UI Design of Grocery Item Page

Description:

- The Add Grocery Item Page is where the user can enter the fields for a new grocery ingredient.
- In this page, the user can input a new ingredient's fields.
  - Item Name - The user enters the name of the ingredient.
  - Image link - The user enters a link for the image of the item that would like to view it as. If left as default, it will select an image for the user.
- Once the desired fields are entered, the user can either add the ingredient by clicking on the Add Item button or cancel by clicking the back button.
- If the user clicks on the Logout button, the user will be directed to the Main Page.

Search Result Page



Search Results

My Store: 290 State Route 18, East Brunswick, NJ, 08816
3.984 miles away
Hours:
Mon 7:00am-8:00pm | Tue 7:00am-8:00pm | Wed 7:00am-8:00pm | Thu 7:00am-8:00pm | Fri 7:00am-8:00pm | Sat 7:00am-8:00pm | Sun 7:00am-8:00pm

	Name	Availability	Price	Rating	# Reviews
	Great Value Large White Eggs, 18 Count	In Stock	2.28$	4.30 ★	163 reviews
	Great Value Large White Eggs, 12 Count	In Stock	1.54$	3.50 ★	175 reviews
	Vital Farms Pasture Raised Large Brown Grade A Eggs, 12 Count	In Stock	5.48$	4.80 ★	20 reviews

Figure 27: UI Design of Search Result Page

Description:

- The user can view the different ingredient brands/types available to them. The list contains the ingredient name, availability, price, rating, and reviews.

- By clicking on the name, availability, price, rating, or review header, the user can sort the corresponding column in ascending or descending order.

- By clicking on the Back button, it allows the user to return back to the Grocery List.

- By clicking on the ingredient name, it will display the ingredient and store information to the user.
    - This action will send the user to the store website to view the ingredient. The information includes price, store location, rating, and how to order the ingredient.

- If the user clicks on the Logout button, the user will be directed to the Main Page.

Design of Tests

Test Cases

Test Case Identifier: TC-1 Use Case Tested: UC-1: Signup Pass/Fail Criteria: The test will pass if the user is able to successfully create a new account. The test case will fail if the user cannot make a new account. Input Data: username (email) and password	
Test:	**Expected Result:**
Test 1: The user inputs an existing email to create an account.	ChefCart will fail to create the new account and show a "User Already Exists" message.
Test 2: The user inputs their credentials. The email is not already in use for a ChefCart account.	ChefCart will succeed in creating a new account for the user using said credentials. These credentials will be saved to the database. The user will be logged in immediately.

Test Case Identifier: TC-2 Use Case Tested: UC-2: Track Pantry, UC-12: View Pantry Ingredients Pass/Fail Criteria: The test will pass if the user is able to see a list of ingredients in their digital pantry. Input Data: session data	
Test Procedure:	**Expected Result:**
Test 1: The user clicks on the "Digital Pantry" button on the home page.	ChefCart will redirect to a page that shows and lists all of the user's pantry ingredients, along with their quantities. This page will also have buttons allowing the user to add, remove and edit ingredients.

Test Case Identifier: TC-3 Use Case Tested: UC-3: Login Pass/Fail Criteria: The test will pass if the user is able to successfully log into the web app. The test case will fail if the user inputs an invalid username or password. Input Data: username (email) and password	
Test Procedure:	**Expected Result:**
Test 1: The user enters a username that is not associated with a ChefCart account.	ChefCart fails to let the user login and shows an "Invalid Credentials" message.
Test 2: The user inputs a username associated with a ChefCart account and the correct associated password and clicks the login button.	The user is logged in and ChefCart displays a welcome page where the user can choose which features they would like to access.
Test 3: The user enters a username associated with a ChefCart account, but different password from one the one associated with the account.	ChefCart fails to let the user login and shows an "Invalid Credentials" message.

Test Case Identifier: TC-4 Use Case Tested: UC-4: Create Grocery List Pass/Fail Criteria: The test will pass if the user is able to see that the automatically generated grocery list restocks their pantry on page entry. The test will fail if the automatically generated grocery list does not restock every ingredient. Input Data: User inputted ingredients, edits and Digital Pantry data.	
Test Procedure:	**Expected Result:**
Test 1: The user clicks the Grocery List button on the home page.	A grocery list is displayed in the form of a table of ingredients with a purchase that replenishes the amount of that ingredient in the digital pantry.

Test Case Identifier: TC-5

Use Case Tested: UC-5: View Grocery Stores

Pass/Fail Criteria: The test will pass if the user can view a list of stores that have the items on the user's grocery list.

Input Data: session data, user specific grocery list

Test Procedure:	**Expected Result:**
Test 1: Once on the Grocery List page, the user clicks on the Search link in a particular item's list entry.	A list of store items relevant to the list item at the nearest store is generated, with information for each store item such as image, name, store link, availability, rating, and number of reviews.

Test Case Identifier: TC-6

Use Case Tested: UC-6: Sort Grocery Stores Items

Pass/Fail Criteria: The test will pass if the grocery store items are sorted according to the relevant criteria, when that criteria is toggled. Otherwise, it will have failed.

Input Data: Sorting options for sorting ascending or descending for item availability, price, rating, name, and number of reviews.

Test Procedure:	**Expected Result:**
Test 1: Sort items by name ascending by toggling the arrow button.	All the items are displayed by increasing items.
Test 2: Sort items by name descending by toggling the arrow button.	All the items are displayed by decreasing name.
Test 3: Sort items by price ascending by toggling the arrow button.	All the items are displayed by increasing prices.
Test 4: Sort stores by price descending by toggling the arrow button.	All the stores are displayed by decreasing prices.
Test 5: Sort items by availability ascending by toggling the arrow button.	All the items are displayed by increasing availability.
Test 6: Sort stores by availability descending	All the stores are displayed by decreasing

by toggling the arrow button.	availability.
Test 7: Sort items by rating ascending by toggling the arrow button.	All the items are displayed by increasing rating.
Test 8: Sort stores by rating descending by toggling the arrow button.	All the stores are displayed by decreasing ratings.
Test 9: Sort items by number of reviews ascending by toggling the arrow button.	All the items are displayed by increasing the number of reviews.
Test 10: Sort stores by number of reviews descending by toggling the arrow button.	All the stores are displayed by decreasing the number of reviews.

Test Case Identifier: TC-7

Use Case Tested: UC-7: Discover Recipes

Pass/Fail Criteria: The test will pass if the program is able to generate a list of recipes that uses all of the ingredients entered.

Input Data: digital pantry ingredients, cuisine input, intolerance input, additional ingredients

Test Procedure:	**Expected Result:**
Test 1: The user enters the Recipe Generator page by clicking on the “Find Recipes” button on either the home page, or the Digital Pantry page. The user enters digital pantry ingredients they want to use, cuisine input, intolerance input, and additional ingredients, and then presses the “Submit” button with valid ingredients.	ChefCart will attempt to find recipes that use all of the ingredients, and list them out in a separate page. If there are no recipes, “No Recipes found” will be displayed to the user.
Test 2: The user enters and performs the same steps as in Test 1 until the adding of ingredients. The user inputs at least one invalid ingredient and then presses the “Submit” button with a mixture of ingredients recognized, and not recognized by the Spoonacular API.	ChefCart will attempt to find recipes that use all of the entered ingredients and will ignore invalid ingredients, and list them out in a separate page. If there are no recipes, “No Recipes found” will be displayed to the user.
Test 3: The user does not enter any ingredients and presses the “Submit” button.	ChefCart will notify that the user must select at least one ingredient before discovering recipes.

Test Case Identifier: TC-8

Use Case Tested: UC-8: Sort/Filter Recipes

Pass/Fail Criteria: Test case passes if list of recipes is correctly sorted (ascending or descending based on context) by the column of which the column header clicked by the user belongs to.

Input Data: none

Test Procedure:	**Expected Result:**
Test 1: The user clicks the "Used Ingredients" header when the "Used Ingredients" column is sorted in descending order.	The list of recipes is sorted by "Used Ingredients", in ascending order.
Test 2: The user clicks the "Used Ingredients" header when the "Used Ingredients" column is sorted in ascending order.	The list of recipes is sorted by "Used Ingredients", in descending order.
Test 3: The user clicks the "Missing Ingredients" header when the "Missing Ingredients" column is sorted in descending order.	The list of recipes is sorted by "Missing Ingredients", in ascending order.
Test 4: The user clicks the "Missing Ingredients" header when the "Missing Ingredients" column is sorted in ascending order.	The list of recipes is sorted by "Missing Ingredients", in descending order.

Test Case Identifier: TC-9

Use Case Tested: UC-9: Viewing Recipe, UC-21: Show Missing Recipe Ingredients

Pass/Fail Criteria: The test will pass if the user is able to view the recipes and all the information relating to it.

Input Data: recipe link

Test Procedure:	**Expected Result:**
Test 1: The user clicks on a recipe link from a generated recipe list from a recipe search.	ChefCart will redirect the user to a Spoonacular page that will display all the information relating to the recipe. For

	example, ChefCart will show the user the prep time, ingredients needed, instructions, estimated price of meal, and other facts about the recipe. It will also show the missing recipe ingredients.

Test Case Identifier: TC-10 Use Case Tested: UC-10: Account Editing Pass/Fail Criteria: The test will pass if the user is able to edit their account in the Edit Account Page. Input Data: session data, old password, new password, city, state, diets, intolerances	
Test Procedure:	**Expected Result:**
Test 1: The user clicks the “Edit Account Info” button on the home page.	ChefCart will redirect the user to a page where he/she can revise or update his/her credentials and preferences.
Test 2: Once on the “Edit Account Page”, the user clicks the “Change Password” button.	ChefCart will redirect the user to a page where the user can change his/her password.
Test 3: Once on the “Change Password” page, the user enters their correct current password, but the new passwords (one for entering and one for confirming) don’t match, and presses the “Submit” button to trigger the account info change.	ChefCart will display an error message at the bottom of the “Change Password” page, saying that the “passwords don’t match”.
Test 4: Once on the “Change Password” page, the user enters their incorrect current password, and presses the “Submit” button to trigger the account info change.	ChefCart will display an error message saying “Invalid Credentials”.
Test 5: Once on the “Change Password” page, the user enters their correct current password, and their new password twice, and presses the “Submit” button to trigger the account info change.	ChefCart takes the user to the home page, indicating that the password change was successful.
Test 6: Once on the “Edit Account” page, the user enters a string value for his/her city, and selects from various lists, his/her state, diets	ChefCart takes the user to the home page, indicating that the account changes were successful.

and intolerances, after which, the user presses the “Submit” button to trigger the account info change.	

Test Case Identifier: TC-11 Use Case Tested: UC-11: Logout Pass/Fail Criteria: The test will pass if the user is able to successfully log out of the web app. Input Data: none	
Test Procedure:	**Expected Result:**
Test 1: The user clicks on the logout button.	The user is logged out of ChefCart and is returned to the login page.

Test Case Identifier: TC-12 Use Case Tested: UC-13: Add Pantry Ingredients Pass/Fail Criteria: The test will pass if the user is capable of adding a new ingredient to the digital pantry or add an amount for an existing ingredient in the digital pantry and is saved. The test will fail if the ingredient is unable to be added and the no addition was made to the digital pantry. Input Data: New ingredient name, quantity, weight, volume, expiration, and warning threshold or new existing ingredient quantity, weight and volume.	
Test Procedure:	**Expected Result:**
Test 1: The user adds a new valid ingredient to the digital pantry by inputting the ingredient name, quantity, weight, volume, expiration, and warning threshold.	The digital pantry is redisplayed to show the added ingredient and the changes are saved in the database.
Test 2: The user attempts to add an invalid ingredient that does not exist according to the Spoonacular API or other invalid inputs in the fields.	The user is returned to the Digital Pantry page and is notified that the food item name is invalid. For all other invalid field inputs, the user is not able to add this ingredient.
Test 2: The user adds an amount to an existing ingredient to the digital pantry by inputting	The digital pantry is redisplayed to show the edited ingredient and the changes are saved in

the new quantity, weight, volume.	the database.

Test Case Identifier: TC-13

Use Case Tested: UC-14: Remove Pantry Ingredients

Pass/Fail Criteria: The test will pass if the user is able to remove an ingredient from the digital pantry or lower an existing ingredient amount.

Input Data: session data, updated existing ingredient quantity, weight and volume.

Test Procedure:	**Expected Result:**
Test 1: The user presses the delete icon right next to an ingredient row.	ChefCart will delete that ingredient from the user's Digital Pantry and the changes will be reflected in the window. The changes will be saved to the database.
Test 2: The user edits an existing ingredient by inputting a new ingredient quantity, weight and volume.	ChefCart will reflect the removal of the ingredient amount as part of the edit and redisplay it. The changes will be saved to the database.

Test Case Identifier: TC-14

Use Case Tested: UC-15: View List Ingredients

Pass/Fail Criteria: This test will pass if the ingredients in the grocery list are displayed in a tabular form to the user. It will fail otherwise.

Input Data: Session data, Digital Pantry data

Test Procedure:	**Expected Result:**
Test 1: The user enters the Grocery List page.	The user is shown a list of ingredients that are recommended for them to purchase based on the amounts in the Digital Pantry in a tabular form.
Test 2: The user adds and/or removes ingredients in the grocery list.	The changes are displayed to the user again in the same tabular form.

Test Case Identifier: TC-15 Use Case Tested: UC-16: Add List Ingredients Pass/Fail Criteria: The test will pass if the user can add an ingredient to the grocery list either new or more of an existing one. The test will fail if this addition is not reflected in the page. Input Data: New ingredient quantity, volume and weight or existing ingredient quantity, volume and weight.	
Test Procedure:	**Expected Result:**
Test 1: The user adds a new ingredient to the grocery list with a quantity, volume and weight.	The new ingredient is added to the grocery list.
Test 2: The user adds an amount to an existing ingredient with a new quantity, volume and weight.	The ingredient is updated with the corresponding amount in the grocery list.

Test Case Identifier: TC-16 Use Case Tested: UC-17: Remove List Ingredients Pass/Fail Criteria: The test will pass if the user can remove an ingredient or an existing ingredient amount from the grocery list. The test will fail if this addition is not reflected in the page. Input Data: May use existing ingredients new quantity, volume, and weight.	
Test Procedure:	**Expected Result:**
Test 1: The user removes an ingredient from the grocery list by clicking the delete button.	The ingredient is removed from the grocery list.
Test 2: The user removes an amount from an existing ingredient with a new quantity, volume and weight.	The ingredient is updated with the corresponding amount in the grocery list.

Test Case Identifier: TC-17 Use Case Tested: UC-18: Verify Ingredients Pass/Fail Criteria: The test will pass if the user successfully finds the ingredient. The test will fail if the user does not find the ingredient. Input Data: ingredient name	
Test Procedure:	**Expected Result:**
Test 1: The user inputs an ingredient name that the Spoonacular API can recognize.	The ingredient will be found and displayed to the user.
Test 2: The user inputs an ingredient name that the Spoonacular API cannot recognize.	There will be no ingredient display to the user.

Test Case Identifier: TC-18 Use Case Tested: UC-19: Ingredient Threshold Warning Pass/Fail Criteria: The test will pass if the system displays the threshold warning when the quantity is below a certain threshold. Input Data: session data	
Test Procedure:	**Expected Result:**
Test 1: The user lowers the quantity of an ingredient so that the quantity is lower than the threshold.	ChefCart will display a threshold warning to let the user know that they are running low on certain ingredients.
Test 2: The user increases the quantity of an ingredient so that the quantity is greater than the threshold.	ChefCart will not display the threshold warning.

Test Case Identifier: TC-19 Use Case Tested: UC-12: View Pantry Ingredients Pass/Fail Criteria: The test will pass if the user can view the Digital Pantry ingredients they have saved in their account Input Data: session data, navigation to Digital Pantry Page	
Test Procedure:	**Expected Result:**
Test 1: The user navigates to the digital pantry	Users can see their Digital Pantry ingredients listed in a tabular fashion. Columns include the ingredient name, quantity, weight, volume, expiration date and buttons to add, edit or delete ingredients.

Algorithm Test Cases

Note that the algorithms for the Digital Pantry and most of the Grocery List algorithms are accounted for in the Use Case Test Cases.

Test Case Identifier: TC-20 Algorithm Tested: Finding the Closest Store Pass/Fail Criteria: This algorithm will pass if it returns the store and its information that contains the grocery ingredients be searched for that is closest to the user. If no store is found, the user will be notified that no store is able to sell the product they are looking for. Input Data: session data, user state and city	
Test Procedure:	**Expected Result:**
Test 1: Find a store that sells a certain ingredient such as eggs. Create an account that has a city and state to be set to that store's city and state. Generate the grocery store for the ingredients.	The account will generate the grocery store that was researched beforehand. The store has that ingredient in stock for the user to potentially purchase.

Test Case Identifier: TC-21 Algorithm Tested: Generating Recipes Pass/Fail Criteria: This algorithm will pass if the returned recipes includes ingredients that the user has asked to include, excludes ingredients the user has asked to exclude, includes cuisines the user has asked for and removes recipes that violate dietary intolerances and does not satisfy dietary preferences. Input Data: session data, included ingredients, excluded ingredients, cuisines, dietary intolerances, dietary preferences.	
Test Procedure:	**Expected Result:**
Test 1: The user will input ingredients they want to include in a recipe, excluded ingredients, cuisines, dietary intolerances and dietary preferences. Each of these fields will be verified by the Spoonacular API. If any field is not valid according to the Spoonacular API query, it will not be considered when generating recipes.	Generated recipes will be shown to the user satisfying all their search parameters. The number of missing ingredients and number of used ingredients that the user has asked for are also included. Every recipe will not contain excluded recipes and will always contain at least one ingredient listed in the included ingredients. At least one recipe will be a part of the included cuisines. No recipe will be generated to violate dietary restrictions and preferences.

Nonfunctional requirement Test Cases

Test Case Identifier: TC-22 Nonfunctional Requirement Tested: REQ-16 Pass/Fail Criteria: This test case will pass if over 99% of user made changes are persisted in the database each time. (This happens only in the Digital Pantry). This test will fail otherwise. Input Data: session data, ingredient name, quantity, weight, volume, expiration date and warning threshold.	
Test Procedure:	**Expected Result:**
Test 1: The ingredients inputted are valid according to TC-12. Perform 1000 ingredient additions, 1000 ingredient edits and 1000 ingredient deletes. Verify each change in the database and tally successes.	Over 2970 of all changes are persisted in the database.

Test 2: The ingredients inputted are invalid according to TC-12.	The change is ignored and the database is unchanged and not corrupted.

Test Case Identifier: TC-23 Nonfunctional Requirement Tested: REQ-17 Pass/Fail Criteria: This will pass if ChefCart can be run on the devices listed in REQ-17. Otherwise, it will fail. Input Data: new device, new account email and password	
Test Procedure:	**Expected Result:**
Test 1: Create a new account on each device listed in REQ-17. Use a simulated version of the device where possible. Add, edit and delete ingredients to the Digital Pantry. Generate grocery lists, and generate recipes.	Account creation works to save the new account to the database. The user is able to see all aspects of the user interface and interact with them. Each change in the digital pantry is reflected in the database. Generating grocery lists and generating recipes works as in the use case test cases.

Test Case Identifier: TC-24 Nonfunctional Requirement Tested: REQ-18 Pass/Fail Criteria: This test case will pass if ChefCart is launched within 3 seconds. Input Data: Launch application	
Test Procedure:	**Expected Result:**
Test 1: Start a timer and launch ChefCart at the same time. Perform this 100 times.	The amount of time it takes to see the main screen is less than 3 seconds on average.

Test Case Identifier: TC-25 Nonfunctional Requirement Tested: REQ-19 Pass/Fail Criteria: This test case will pass if page transitions are less than 1 second each. Input Data:	
Test Procedure:	**Expected Result:**
Test 1: Start a timer and perform page transitions. Perform this 100 times.	The amount of time it takes to see the next screen is less than 1 second on average.

Test Case Identifier: TC-26 Nonfunctional Requirement Tested: REQ-20 Pass/Fail Criteria: This test case will pass if data is generated within 3 seconds and people can read the data within 1 meter. Input Data: recipe generator parameters	
Test Procedure:	**Expected Result:**
Test 1: Generate data using the recipe generator as it is the most intensive data generating feature. Start a time and generate recipes 100 times.	Recipes are generated and displayed within 3 seconds.
Test 2: Recruit 10 people with near average eyesight. Have them read aloud the text on ChefCart from 1 meter away.	Every person reads the words perfectly at this distance.

Test Case Identifier: TC-27 Nonfunctional Requirement Tested: REQ-21 Pass/Fail Criteria: This test case will pass if all User Interface interactable elements are accessible to the user without confusion. Input Data: new screen size

Test Procedure:	Expected Result:
Test 1: Starting from a large display size of 1920 by 1080 pixels, reduce the ChefCart display down by 50 pixels at a time for length and width until reaching 920 by 1120 by 280 pixels. From there, reduce 1120 pixels 100 pixels at a time to 520 pixels. Scroll to find and interact with each button element and interactable element as in the user interface. All elements include links, buttons, and text fields.	The user is able to find and use each interactable element.

User Interface Requirement Test Cases
Note that the following User Interface requirements are tested for in the use case test case procedures and therefore are not included here:

- UIREQ-2
- UIREQ-6
- UIREQ-7
- UIREQ-8
- UIREQ-9
- UIREQ-10
- UIREQ-11
- UIREQ-12
- UIREQ-13
- UIREQ-14
- UIREQ-15
- UIREQ-16
- UIREQ-17
- UIREQ-18
- UIREQ-19
- UIREQ-20

Test Case Identifier: TC-28 User Interface Requirements Tested: UIREQ-1 Pass/Fail Criteria: This test case will pass if can see the main page and is able to choose between logging in and signing up for an account. Input Data: Launch application	
Test Procedure:	**Expected Result:**
Test 1: Launch application	The user sees the main page and is able to choose between logging in and signing up.

Test Case Identifier: TC-29 User Interface Requirements Tested: UIREQ-3 Pass/Fail Criteria: This test case will pass if the user is able to go to the Home Page for all screens after the login page and then sees buttons to go any of the Edit Account Page, Digital Pantry Page, Grocery List Page, and Find Recipes Page. Input Data: Login credentials. Click to Home Page.	
Test Procedure:	**Expected Result:**
Test 1: Login and go through every page. Test to verify that there is a button to navigate to the Home Page.	The user sees the Home Page button at each page after logging in and is navigated to it. They see buttons to navigate to the Edit Account Page, Digital Pantry Page, Grocery List Page, and Recipe Generator Page.
Test 2: Login and go to the Home Page. Click to go to the Edit Account Page.	The user is navigated to the Edit Account Page.
Test 3: Login and go to the Home Page. Click to go to the Digital Pantry Page.	The user is navigated to the Digital Pantry Page.
Test 4: Login and go to the Home Page. Click to go to the Grocery List Page.	The user is navigated to the Grocery List Page.
Test 5: Login and go to the Home Page. Click to go to the Recipe Generator Page.	The user is navigated to the Recipe Generator Page.

Test Case Identifier: TC-30 User Interface Requirements Tested: UIREQ-4 Pass/Fail Criteria: This test case will pass if the user is able to navigate to each of the Edit Account Page, Digital Pantry Page, Grocery List Page, and Find Recipes Page from the Home Page. Input Data: Click button to navigate.	
Test Procedure:	**Expected Result:**
Test 2: Login and go to the Home Page. Click to go to the Edit Account Page.	The user is navigated to the Edit Account Page.

Test 3: Login and go to the Home Page. Click to go to the Digital Pantry Page.	The user is navigated to the Digital Pantry Page.
Test 4: Login and go to the Home Page. Click to go to the Grocery List Page.	The user is navigated to the Grocery List Page.
Test 5: Login and go to the Home Page. Click to go to the Find Recipes Page.	The user is navigated to the Find Recipes Page.

Test Case Identifier: TC-31

User Interface Requirements Tested: UIREQ-5

Pass/Fail Criteria: This test case will pass if there is a button to the previous screen at every page beyond the Main Page.

Input Data: Click button to go to previous page

Test Procedure:	**Expected Result:**
Test 1: Go to every page. Find the corresponding back button. Click the button.	The user is navigated to the previous page after clicking the back button. Pages such as the Login Page will navigate back to the Main Page. Pages of the Edit Account Page, Digital Pantry Page, Grocery List Page, and Find Recipes Page can logout to go to the Login Page. These are exceptions that work in a hierarchical fashion.

Test Coverage

Our test cases cover all use cases from UC-1 to UC-19. In addition, we wrote test cases to cover our unique algorithms, nonfunctional requirements and User Interface requirements that were not already covered in the use case test cases.

At this point, all of the core functionality is already implemented. For the rest of the semester, we will continue to add and adjust these use cases as more functionality is implemented. Each test case will test for some use cases and will ensure that there is a procedure for handling different types of user input, whether it be a failure result or success result. Most of these successes and failures will notify the user. We want to make sure that all failures and success pathways for each input are tested for. If there are an unreasonable number of inputs that a user can input, then we will limit the number of inputs for the user to limit the possibility of failure. For example, when inputting the amount of an ingredient, a user is unable to input a character that is not a number into the field. Rather than notifying the user with an alert, we simply remove their ability to perform certain inputs.

Integration Testing Strategy

We chose to use bottom-up integration as our strategy for integration testing. This strategy tests the units at the lowest level of the hierarchy and then combines them so that we will start out with the least amount of dependencies. These lower level units are verified for integrity and usage so that when we combine them to test higher level units, we can rule out failures that resulted from the lower level units themselves. This will help pinpoint failures for debugging at every step of our application. By building a good foundation, we can be sure that our higher level code in the future will stand strong. Each additional feature, if it has a bug, can have that bug be easily pinpointed to the integration, since we can recursively trust that building block units work for all test cases. As an example, what we intend to do is test to make sure that ingredients can be created and passed to and from the database. Then, we test to ensure that these ingredient objects have the required functionality to be used in the Spoonacular API and grocery store APIs for searching. From these guarantees when manipulating the ingredients, we can then individually implement the Digital Pantry, Grocery List, and Recipe Generator higher level components and test them under the assumption that Ingredient object classes work correctly.

Project Management

In order to efficiently manage and merge individual contributions, we are utilizing a branch & pull request approach with a github repository. By doing so, we ensure that the main working codebase does not become accidentally corrupted and that all code changes are up to the standard we set for ourselves. But of course not everyone knows how to use GitHub so we have also been uploading code on Google Drive so that other people can view and upload them on GitHub. In order to compile the final report, we are using a Google Doc in order to allow for real-time contributions from all group members. Thus far, our only issues encountered have been with setting up accounts and managing shared credentials. Since some of the APIs we are using require payment after a specific number of calls, we all needed to create accounts so that until we have finalized and tested our code we do not need to incur costs.

From a coordination aspect, we are leveraging Discord to work with each other asynchronously. We also have group calls weekly or biweekly as needed to touch base and create a plan of work for the immediate future. Finally, before submitting reports we have group members read through them and make edits for consistency in wording and overall structure.

The main issue we have encountered is that the individual subteams tend to make assumptions regarding the architecture and workflow of other subteams, which results in the teams completing unnecessary work, or creating features that cannot be used. It was easier to make assumptions than to double check with another team member and wait on their response. To solve this problem, we implemented “work hours”, and an “expected response time”. Essentially, during work hours the group members are expected to reply to questions in a specified amount of time. This way, we can encourage conversation between the teams and ensure that rather than making assumptions, teams can save more time by asking good questions.

History of Work, Current Status, and Future Work

History of work

At the present, we have completed all core functionality. This encompases user signup, user login, user account editing, digital pantry management (additions, deletions, and edits), recipe lookup, grocery list management, and grocery list item lookups. We had extensive changes in our use case points and technical complexity as we carried on with the work, and we had to recalculate some of our UUCW and UCP.

Key accomplishments

Looking back to our projected milestones from report 2, the group has reached the project milestones faster than expected. At present, the only remaining task is to integrate all components of the project into the webapp, and test the completed webapp. We did not expect to reach this position until 4/21, so we are approximately one week ahead of schedule. We will use this extra time to commit to more extensive tests of the final project, so that we have a bug-free and optimally functioning final demo. Additionally, we may use the opportunity to address some of the concerns raised by the professor and TAs during our first demo. These include:

- Threshold alerts item-by-item instead of for the whole pantry
- Expiration alerts
- Security fixes for user inputs
- Binary option for pantry ingredients like salt, pepper, etc.

We are also looking back on our recent work and highlighting these major goals that the group has accomplished during this project:

- Integrating w/ the spoonacular API
- Integrating w/ the walmart API
- Utilizing gRPC, Go, and VueJS to create a responsive webapp
- Utilizing geonames API to perform closest store search for users

Future Plans

Many of our future plans come from notes raised by the professor and TAs during the first demo.

- Allow users to save recipes to the web app and add notes on deviations from the recipe.
- Store user history and recent searches in local storage in order to allow users to quickly recall their last search. We are not implementing this function during the semester as we do not have the budget to use database tools like GCP BigQuery or AWS Athena. Future students could also elaborate on this option and find other ways to store user data, perhaps using cookies or local downloads.
- Expansion:
    - Find other grocery stores with APIs and integrate their API calls into ChefCart to offer users greater flexibility in shopping options
    - Integrate other recipe APIs like Spoonacular to widen the recipe databases.

References

Marsic, Ivan (2020), RU ENG ECE 14:332:452 Software Engineering, https://www.ece.rutgers.edu/~marsic/Teaching/SE/index.html, 2/9/21

Spoonacular (2021), Spoonacular API, https://spoonacular.com/food-api, 2/9/2021

Walmart (2021), Walmart Developer Portal, https://developer.walmart.com/, 2/9/2021

Wegmans (2021), Wegmans Developers, https://dev.wegmans.io/, 2/9/2021

PostgreSQL (2021), PostgreSQL, https://www.postgresql.org/docs/13/app-psql.html, 2/15/2021