

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



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

<https://chefcart.herokuapp.com>

Project Report 2  
23 March 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

Cover Page	1
Individual Contributions	2
Table of Contents	3
Analysis and Domain Modeling	5
Conceptual Model & Concept Definitions	5
Association Definitions	6
Attribute Definitions	7
Traceability Matrix	8
Domain Model Diagram	10
System Operation Contracts	11
Data Model	13
Mathematical Model	14
Interaction Diagrams	16
Track Pantry	16
Create Grocery List	17
View Grocery Stores	18
Discover Recipes	19
Class Diagram and Interface Specification	20
Class Diagram	20
Data Types and Operation Signatures	22
Traceability Matrix	26
Algorithms and Data Structures	31
Algorithms	31
Data Structures	32
Concurrency	33
User Interface Design and Implementation	34

Design of Tests	46
Tests	46
Test Coverage	56
Integration Testing Strategy	56
Project Management and Plan of Work	57
Project Management	57
Use Case Status Updates	58
Projected Milestones	59
Responsibilities Breakdown	60
Plan of Work Gantt Chart	61
References	62

# Analysis and Domain Modeling

## Conceptual Model

### 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
Tracks the grocery store filters.	K	StoreFilter
Filters shown grocery stores based on user preferences.	D	StoreFilter
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

### 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
StoreFilter ⇔ Interface	Display filtered grocery stores.	FilterStores
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

### Attribute Definitions

Concept	Attribute	Description
AccountDetails	Username	Associated username of the account
	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_Unit	The unit of measurement for the specific ingredient
	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
	Closest_Expiration	The nearest expiration date for the Ingredients
	Last_Updated_Date	Date when Digital Pantry was last updated
GroceryList	Grocery_Ingredient_List	List of grocery ingredients
StoreFilter	Max_Distance	Maximum distance a user is willing to see stores.
	Sort_By	Current sorting mechanism for price and distance.
	Max_Price	Maximum price a user is willing to shop for groceries
RecipeFilter	Digital_Pantry	Using Digital Pantry ingredients to discover recipes.
	Sort_By	Current sorting mechanism for price, difficulty and percentage of ingredients missing.

## Traceability Matrix

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 Filter						X		X													
	Recipe Filter								X													
	Threshold Warning																			X		
	Ingredient Manager		X		X									X	X		X	X	X			

## Use Cases

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

**UC-2: Track Pantry** - Allows users to view and manipulate ingredients from their digital pantry.

**UC-3: Login** - Allows users to login in with their account credentials

**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.

**UC-5: View Grocery Stores** - Allows users to view grocery stores that have the ingredients for them to purchase and complete their grocery list.

**UC-6: Sort Grocery Stores** - Allows users to sort the list of stores based off of any set of user-specified preferences such as distance and price

**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



**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 such as price, prep time, difficulty, percentage of ingredients available

**UC-9: Viewing Recipe** - Allows users to view the recipe instructions and other relevant information such as required ingredients, price, difficulty, prep time, etc.

**UC-10: Account Editing** - Allows users to view account setting and edit them. The setting includes, dietary restriction, username, password, and location.

**UC-11: Logout** - Allows users to logout of any page and be sent back to login screen

**UC-12: View Pantry Ingredients** - Allows the user to view ingredients currently present in the digital pantry.

**UC-13: Add Pantry Ingredients** - Allows the user to add ingredients or add existing ingredient amounts to the digital pantry.

**UC-14: Remove Pantry Ingredients** - Allows the user to remove ingredients or remove existing amounts from the digital pantry.

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

**UC-16: Add List Ingredients** - Allows the user to add ingredients or add existing ingredient amounts to the grocery list.

**UC-17: Remove List Ingredients** - Allows the user to remove ingredients from the grocery list or remove existing ingredient amounts.

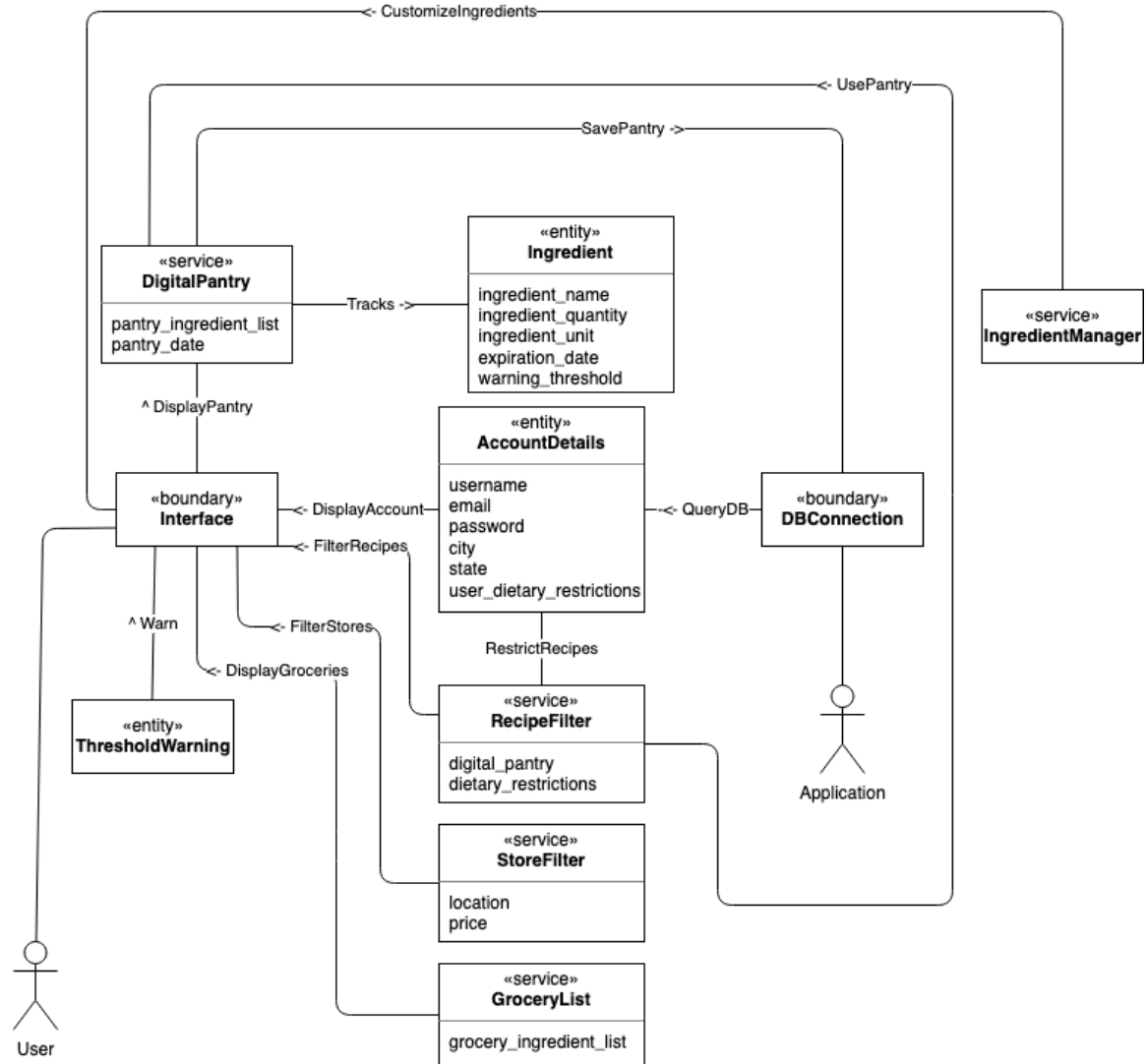
**UC-18: Search Ingredients** - Allows the user to search for an ingredient when trying to add it to either the grocery list or digital pantry.

**UC-19: Ingredient Threshold Warning** - Allow the user to be warned when the ingredient amount is below a threshold

**UC-20: Shop Missing Recipe Ingredients** - Allow the user to add missing recipe ingredients to their grocery list after viewing a recipe.

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

## Domain Model Diagram



## System Operation Contracts

Operation	Track Pantry
Use Case:	UC-2
Preconditions	<ul style="list-style-type: none"> <li>• The user has logged in and has navigated to the Digital Pantry tab in the interface.</li> <li>• ThresholdWarning shows a warning for a particular ingredient if the Warning_Threshold amount is greater than the current ingredient amount or Ingredient_Unit attribute.</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• Attributes for the ingredient concept will be updated and the list of ingredients in the Digital Pantry will be updated in the database.</li> <li>• The interface will display the updated list to the user.</li> <li>• The database and the interface will hold the same data.</li> </ul>

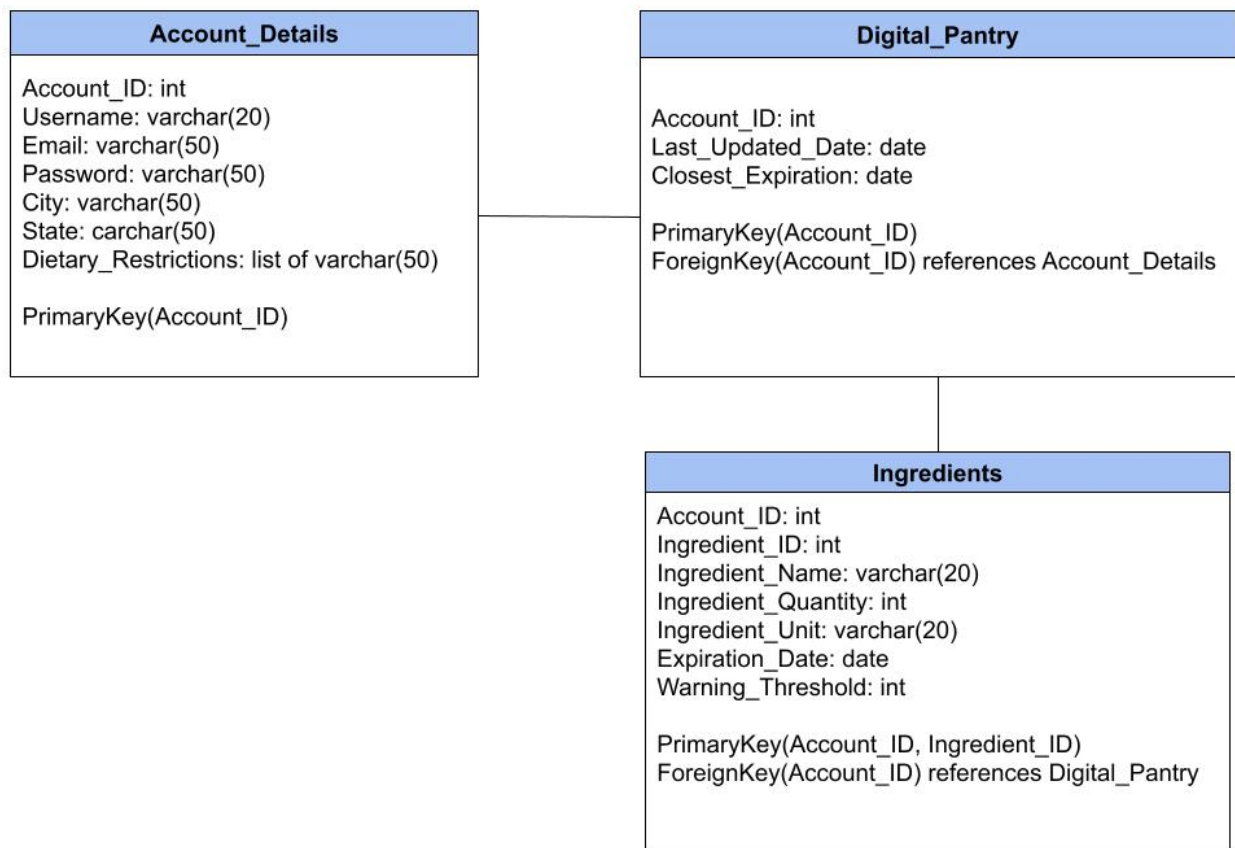
Operation	Create Grocery List
Use Case:	UC-4
Preconditions	<ul style="list-style-type: none"> <li>• The user has launched the ChefCart app.</li> <li>• The user has successfully logged into ChefCart.</li> <li>• After logging in, the user has navigated to the Grocery List Tab.</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• The interface displays a list of ingredients that will be the user's created GroceryList</li> </ul>

Operation	View Grocery Stores
Use Case:	UC-5
Preconditions	<ul style="list-style-type: none"> <li>• The user has launched and logged into ChefCart</li> <li>• The user was on the grocery list page, generated a grocery list and clicked on the "Choose Store" button.</li> </ul>
Postconditions	<ul style="list-style-type: none"> <li>• The displayed stores, if any, will allow users to click on them to view their information in the Store Page.</li> <li>• All stores displayed in the interface satisfy the StoreFilter.</li> </ul>

Operation	Discover Recipes
Use Case:	UC-7
Preconditions	<ul style="list-style-type: none"><li>• The user has launched and logged into ChefCart</li><li>• The user has navigated to the Recipe Generator page.</li></ul>
Postconditions	<ul style="list-style-type: none"><li>• The system will display any found recipes to the user.</li><li>• All recipes displayed in the interface satisfy the RecipeFilter.</li><li>• All recipes displayed in the interface satisfy AccountDetails User_Dietary_Restrictions.</li><li>• 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.</li></ul>

## 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 username and each user has only one Digital Pantry. Below is a diagram of the relational schema of our database. 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.



## Mathematical Model

We do not use a mathematical model in our project.

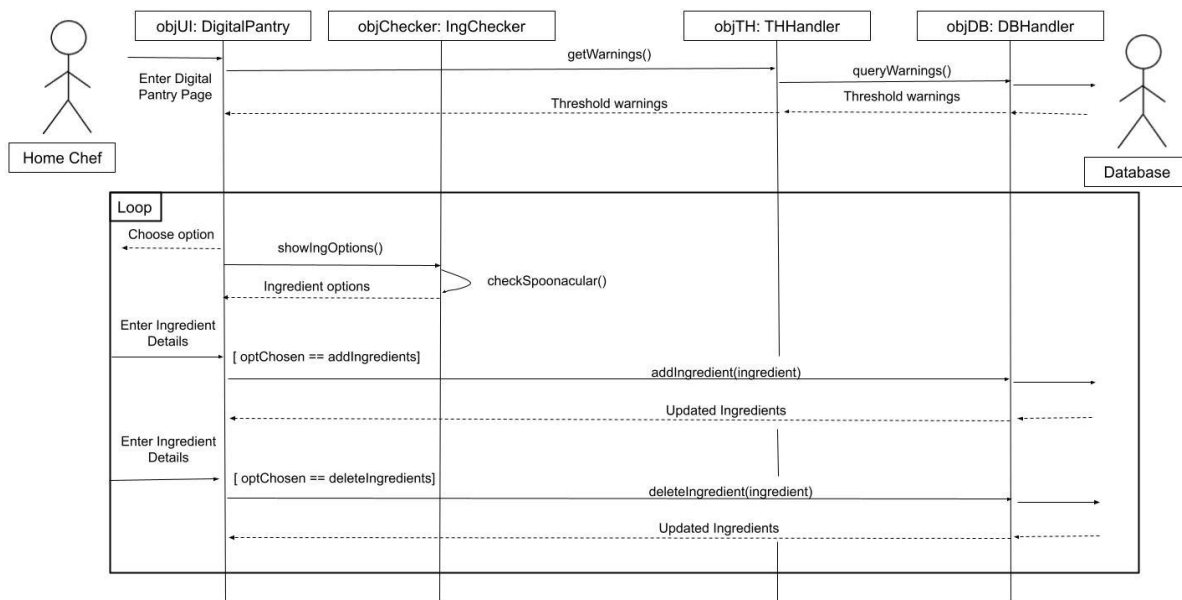
# Interaction Diagrams

## Track Pantry

**Description:** The UC-2: Track Pantry diagram below, 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.

**Design Principle:** 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.

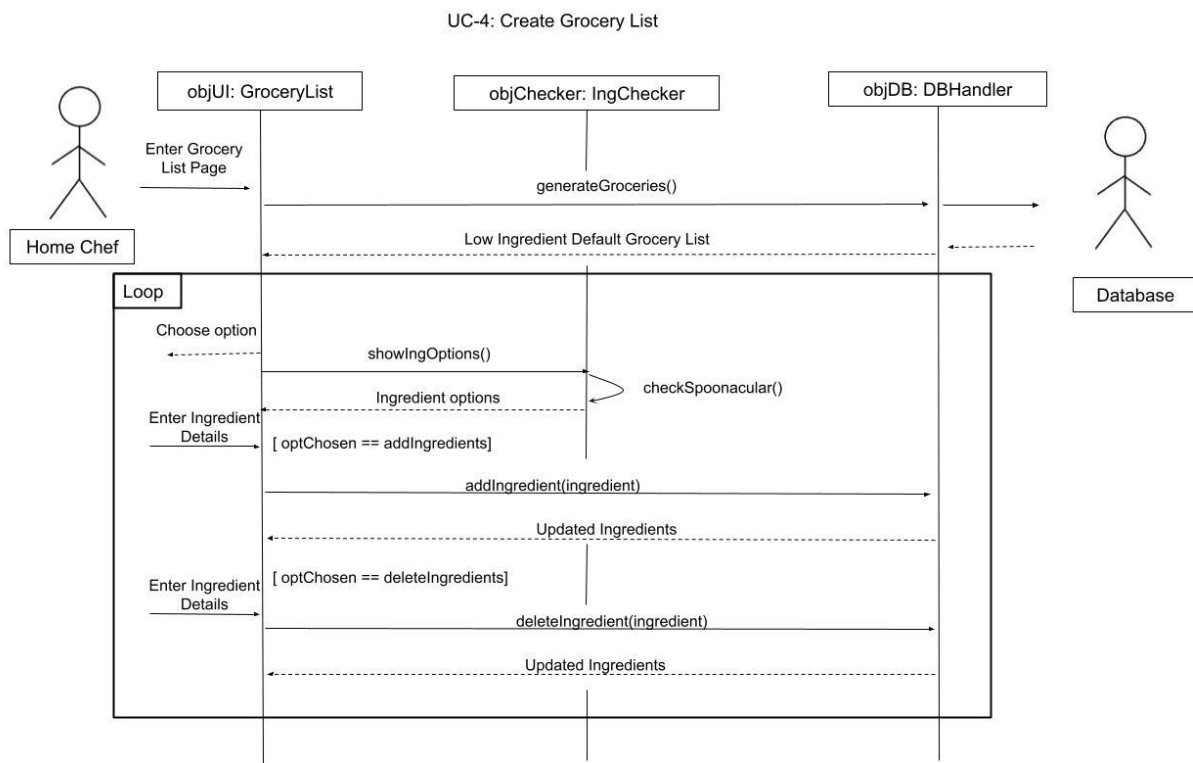
UC-2: Track Pantry



## Create Grocery List

**Description:** The UC-4: Create Grocery List diagram below, 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. We decided to remove the option to edit ingredients because it can be simplified to adding or deleting ingredients by amount instead. 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.

**Design Principles:** 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.



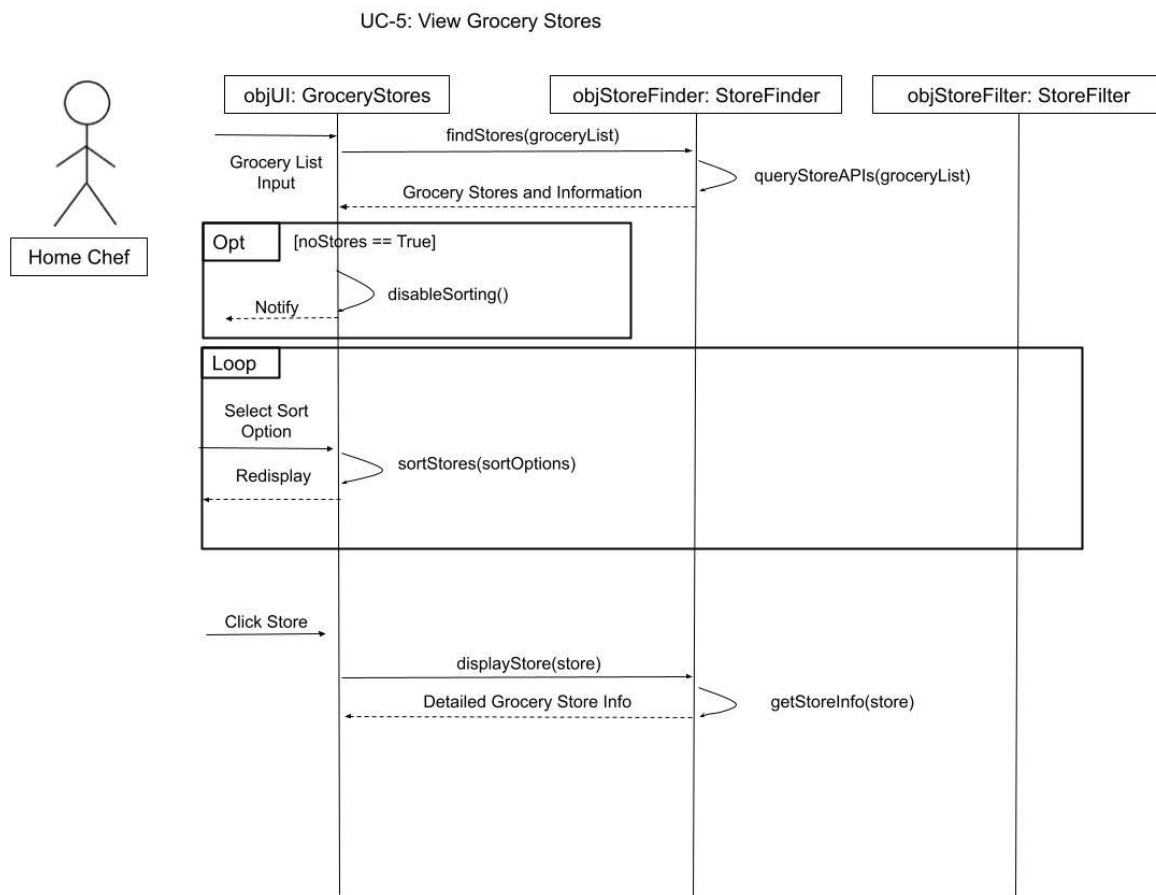


## View Grocery Stores

**Description:** In the UC-5: View Grocery Stores diagram below, it shows how the user can generate grocery stores. Once the user is in the Grocery List section, the user can generate a list of grocery stores by using the grocery list. When generating the stores, we will be using the Store API to check and find the appropriate stores. The user can choose to sort the stores based off of any set of user-specified preferences such as distance, price. The user can also choose to filter the stores based off of user preference. If the user wants to know more details about a specific store on the list, the user can click on the store and the information relating to the store will be displayed.

### Design Principles:

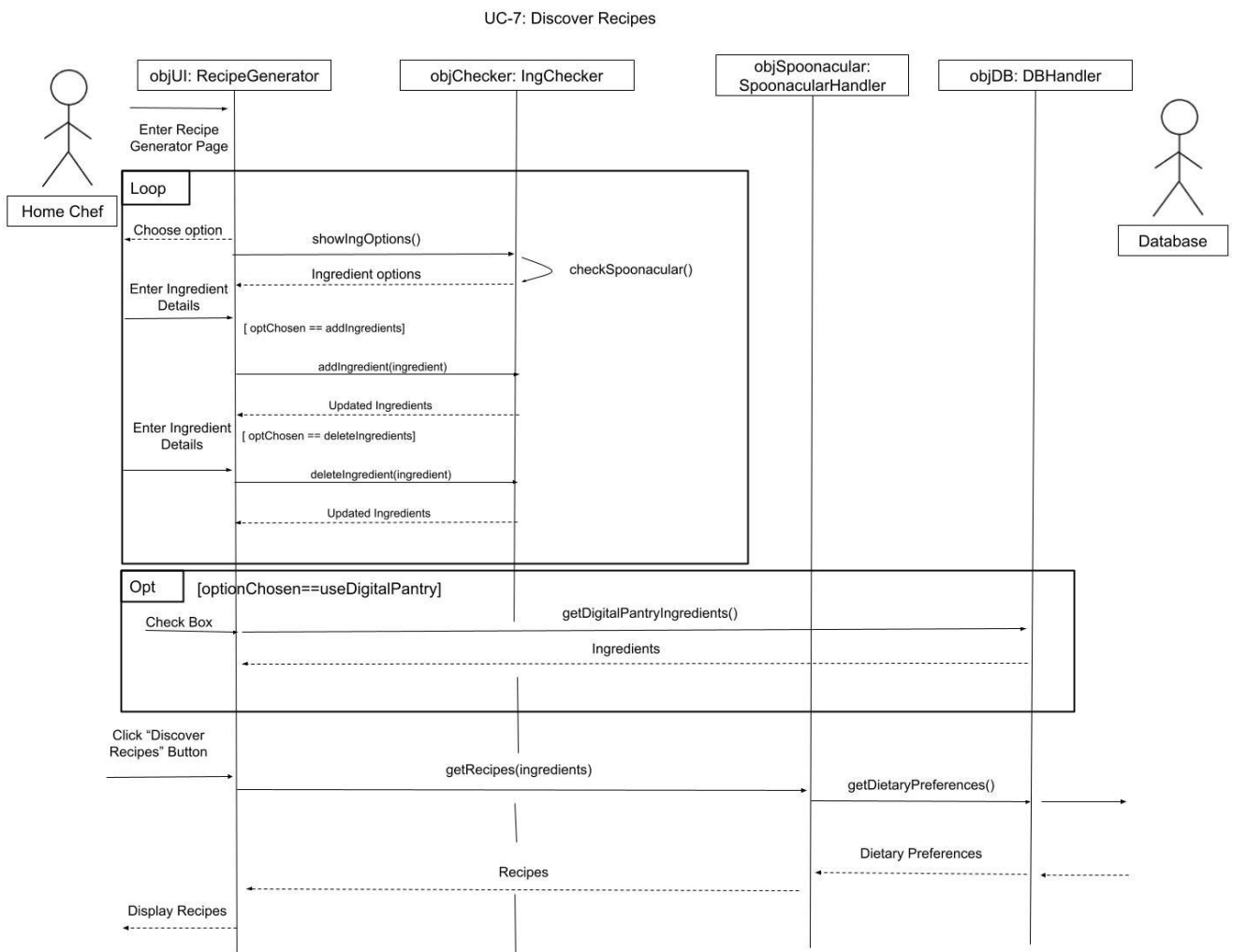
For this UC our most important principle was the ‘expert doer’ or information expert principle. There are several pieces of information floating in this use case. We need the user’s location, the location of grocery stores, the user’s store preferences and filters, and other info about each specific 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.



## Discover Recipes

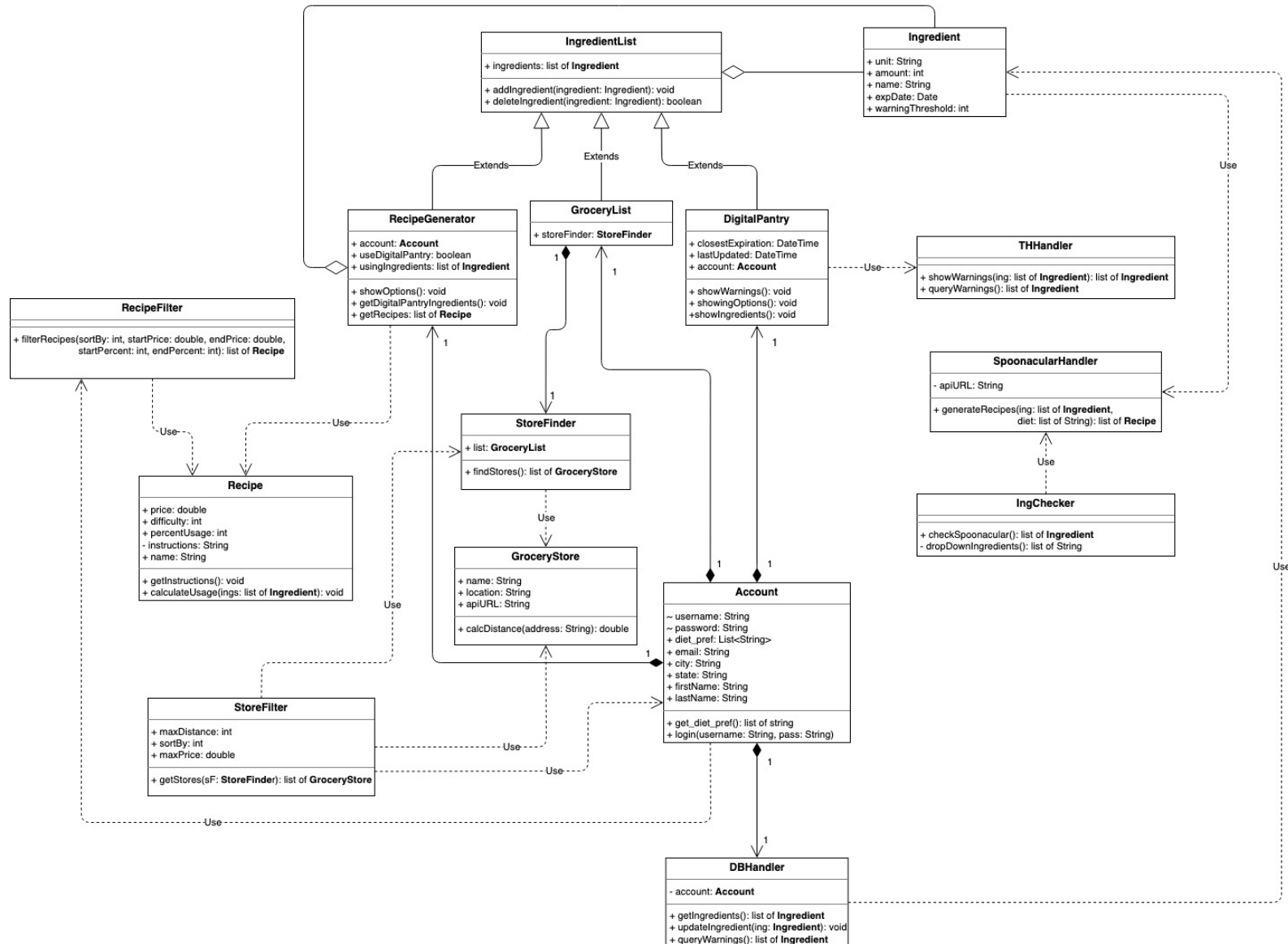
**Description:** The diagram below shows how the user can generate recipes. Once the user is in the Recipe Generator, they can generate a list of recipes by using the digital pantry or manual inputs. There is a check box for the user to click if they want to use the digital pantry to create the list of ingredients. 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. We removed the option to edit ingredients because it can be simplified to adding or deleting ingredients by amount instead. After all the ingredients are finalized and the user clicks on the Discover Recipe button, a list of recipes will be generated. When generating the recipes, we will also use the database to get the user's dietary restriction and preferences to find recipes that are better suited for the user.

**Design Principles:** 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.



## 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.



## 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
~ username: String ~ password: String + diet_pref: List<String> + email: String + city: String + state: String + firstName: String + lastName: String
+ get_diet_pref(): list of string + login(username: String, pass: String)

### Attributes

- username: String - the username of the user
- 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
- city: String - the user's city of residence
- state: String - the state where the user resides
- firstName: String - the first name of the user
- lastName: String - the last name of the user

### 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
  - setUsername(username: String): void
  - setPassword(password: String): void
  - setDietPref(preferences: List of String): void
  - setEmail(email: String): void
  - setCity(city: String): void
  - setState(state: String): void
  - setFirstName(firstName: String): void
  - setLastName(lastName: String): void
- Getters - Gets the value of the corresponding attribute
  - getUsername(): String
  - getDietPref(): List of String
  - getEmail(): String
  - getCity(): String
  - getState(): String
  - getFirstName(): String
  - getLastName(): String

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

<b>IngredientList</b>
+ ingredients: list of <b>Ingredient</b>
+ addIngredient(ingredient: Ingredient): void + deleteIngredient(ingredient: Ingredient): boolean

Attributes

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

Method

- addIngredient(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.

<b>DigitalPantry</b>
+ closestExpiration: DateTime + lastUpdated: DateTime + account: <b>Account</b>
+ showWarnings(): void + showingOptions(): void + showIngredients(): void

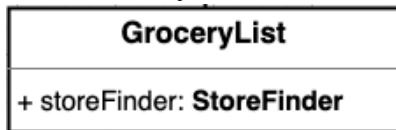
Attributes

- 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.



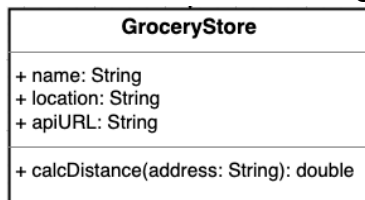
Attributes

- 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.

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

#### 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 Ingredient - 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
- addIngredient(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.

<b>Ingredient</b>
+ unit: String + amount: int + name: String + expDate: Date + warningThreshold: int

### 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 <b>Ingredient</b> - 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: StoreFilter - This class sorts stores to display to the user.

StoreFilter
+ maxDistance: int + sortBy: int + maxPrice: double + getStores(sF: <b>StoreFinder</b> ): list of <b>GroceryStore</b>

### Attributes

- maxDistance: int - an integer mile distance from the user's position
- sortBy: int - an int representing how to sort Stores (0 = name, 1 = distance, 2 = location, else = price);
- maxPrice: double - maximum price one is willing to pay at any given store



## Methods

- `getStores()`: returns a list of stores fitting the attribute constraints specified above.
- **Setters** - Sets the value of the corresponding attribute
  - `setMaxDistance(distance: int): void`
  - `setSortBy(sort: int): void`
  - `setMaxPrice(price: double): void`
- **Getters** - Gets the value of the corresponding attribute
  - `getMaxDistance(): int`
  - `getSortBy(): int`
  - `getMaxPrice(): double`

Class: `StoreFinder` - This class finds the stores that a user can shop at.

<b>StoreFinder</b>
+ list: <b>GroceryList</b>
+ findStores(): list of <b>GroceryStore</b>

## Attributes

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

## Methods

- `findStores()`: returns a list of stores that have the items in the grocery list
- **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.

<b>THHandler</b>
+ showWarnings(ing: list of <b>Ingredient</b> ): list of <b>Ingredient</b>
+ queryWarnings(): list of <b>Ingredient</b>

## 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: <b>Account</b>
+ getIngredients(): list of <b>Ingredient</b> + updateIngredient(ing: <b>Ingredient</b> ): void + queryWarnings(): list of <b>Ingredient</b>

#### 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
+ filterRecipes(sortBy: int, startPrice: double, endPrice: double, startPercent: int, endPercent: int): list of <b>Recipe</b>

#### Attributes

- N/A

#### Methods

- filterRecipes(startPrice: double, endPrice: double, startPercent: int, endPercent: int): List of Recipe - returns the filtered list of recipes based on a price and percentage usage range of ingredients.
- 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.

Recipe
+ price: double + difficulty: int + percentUsage: int - instructions: String + name: String
+ getInstructions(): void + calculateUsage(ings: list of <b>Ingredient</b> ): void

#### Attributes

- price: double - the estimated price for making the recipe
- difficulty: int - the estimated difficulty of cooking the recipe provided by Spoonacular
- percentUsage: int - the percentage of ingredients used that the user desired to use
- instructions: String - the instructions for the particular recipe
- name: String - the name of the recipe

#### Methods

- calculateUsage(ings: List of Ingredient): void - calculates and sets the percentUsage variable for the percentage of ingredients used
- Setters - Sets the value of the corresponding attribute
  - setPrice(price: double): void
  - setDifficulty(difficulty: int): void
  - setPercentUsage(percent: int): void
  - setInstructions(instructions: String): void
  - setName(name: String): void
- Getters - Gets the value of the corresponding attribute
  - getPrice(): double
  - getDifficulty(): int
  - getPercentUsage(): int
  - getInstructions(): String
  - getName(): String

#### Traceability Matrix: Line by line concepts → classes

Concept Name	Responsibility	Derived Classes
Interface	Display options and data for the user to interact with ChefCart.	Interface
DBConnection	Manages connections with	DBHandler - All database actions are

	the database	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.	DigitalPantry - Class must intake the Ingredient concept GroceryList - Class must intake the ingredient concept
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.
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

StoreFilter	Tracks the grocery store filters.	StoreFinder - Contain the user updated parameters to find stores
StoreFilter	Filters shown grocery stores based on user preferences.	StoreFinder - contained parameters are used to filter stores GroceryStores - 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.
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.

# Algorithms and Data Structures

## Algorithms

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

### Digital Pantry Page

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, the system will use the Spoonacular API to check to see 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 of the ingredient. 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.

### Recipe Generator Page

When the user wants to find a recipe to use, the user can use the recipe generator. This algorithm will generate a list of recipes and display them to the user. The user interface will ask the user what ingredients that he or she 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). Once the user clicks the 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 system can also filter the recipes based on the percentage of ingredients available, price, and difficulty if the user chooses to.

### Grocery List Page

The user would want to buy more ingredients to use when low on certain ingredients. The algorithm creates the grocery list and generates a list of stores 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. ChefCart will comb store APIs to check if it has each and every ingredient in stock from the grocery list. If so, it will include it in the list of stores for the user to purchase from. Upon display, it will also calculate the total price of the user's grocery shopping trip at that store. The store information will be displayed when the user clicks on the store. The user can also filter the store. For example, the user can filter it by distance and find the closest store.

## Data Structures

The data structures that we decide to use are fairly simple. The only complex data structure that we will need are arrays. In particular, we only need these arrays 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 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.

These data structures for ingredients will be packaged nicely in JSON to send parameters and receive results. This is useful when pingging grocery store APIs for particular ingredients as well as receiving recipes that can be generated using particular ingredients by sending it to the Spoonacular API. JSON data sent back to us will be processed to fill in the necessary UI for the user to see.

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.

**Concurrency**

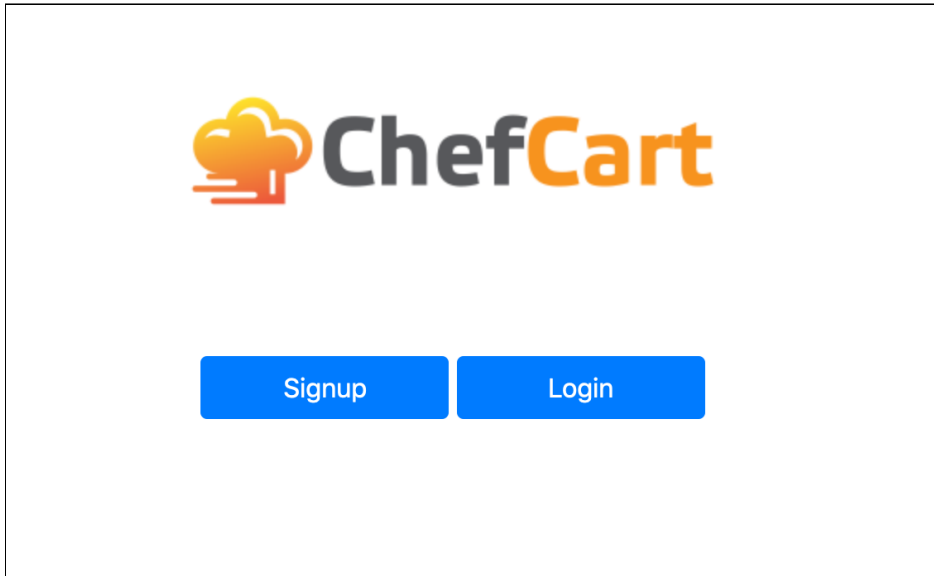
We do not use any concurrent threads or programming.



# User Interface Design and Implementation

## Account Handling subproject:

### Start Up Page



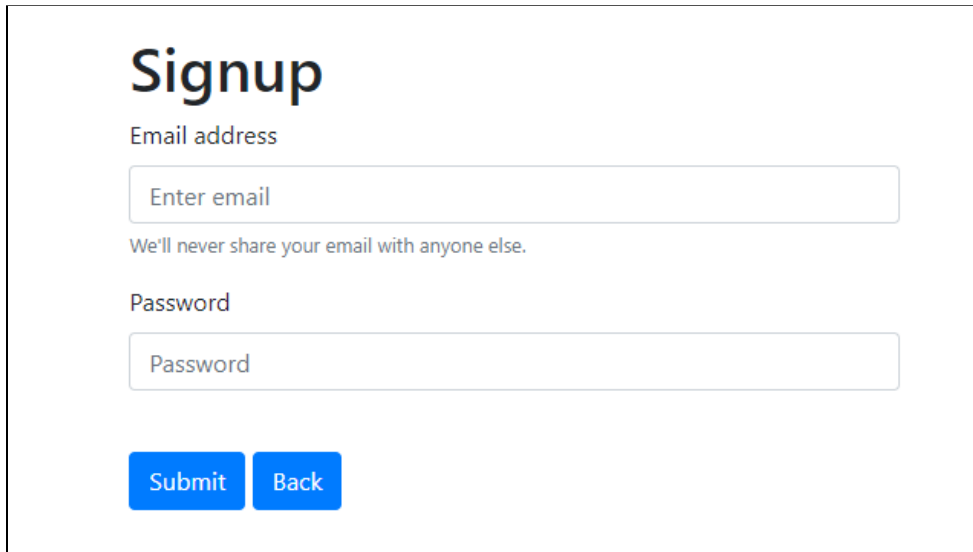
#### Description:

This is the start up page where 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 Signup 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.

One change we made is adding the start up page. Originally, after inputting the url and click enter, it should bring you to the login page. Now, it brings the user to an “welcome” page and from there, the user can choose to login or signup. The user would have one additional task to perform before logging in, but with this new start up page, signup and login are more organized.

## Sign-Up Page



**Signup**

Email address

Enter email

We'll never share your email with anyone else.

Password

Password

Submit Back

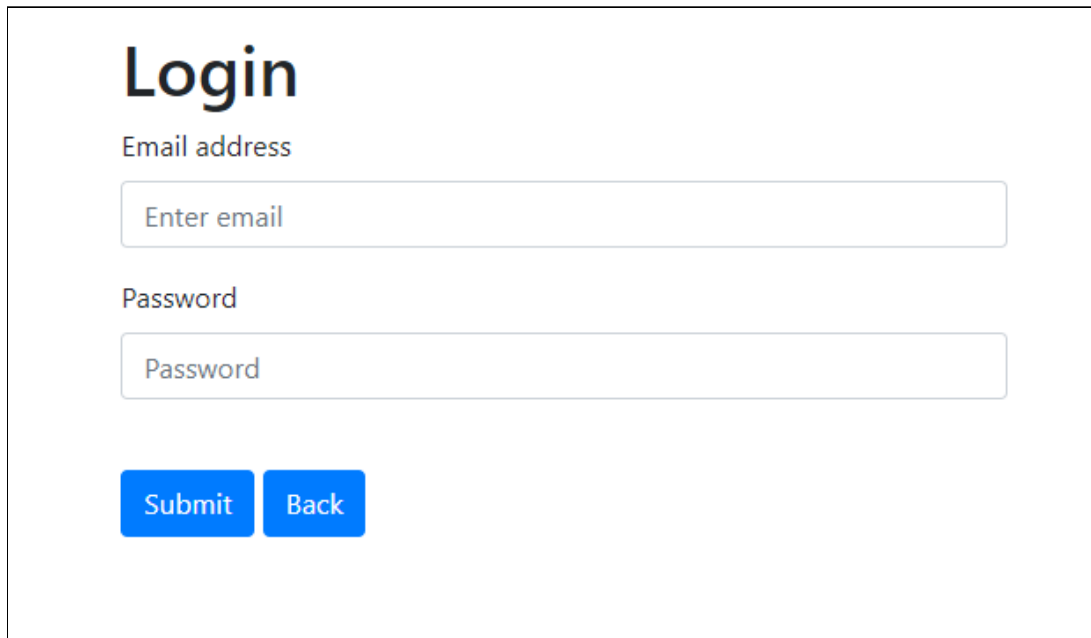
### Description:

This is the signup 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.

One change we made is that the user doesn't have to enter city and state when creating a new account. Those fields can be created/changed in the Edit Account Page. This change reduces the amount of information that the user needs to input.

## Login Page



The login form is titled "Login" in a large, bold, black font. Below the title, there are two input fields. The first field is labeled "Email address" and contains the placeholder text "Enter email". The second field is labeled "Password" and contains the placeholder text "Password". Below the input fields, there are two buttons: a blue "Submit" button and a blue "Back" button.

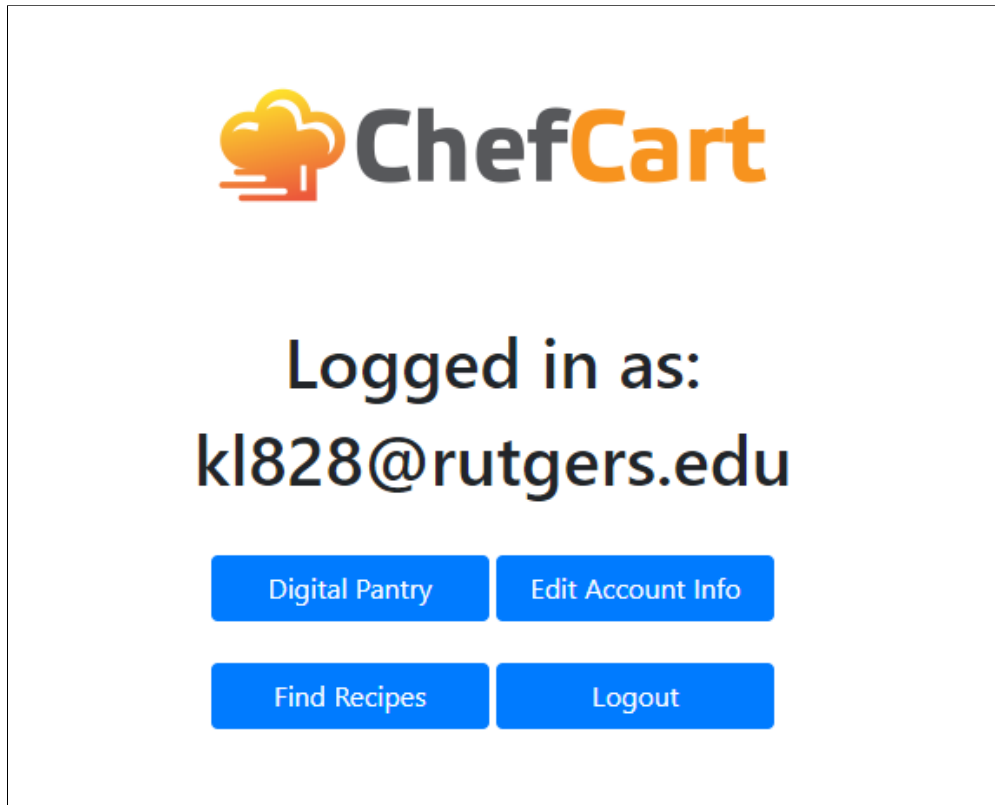
### Description:

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 Main Page.

One change that we made is that we combine the username and email. Originally, we had email and username as two completely different things. Now, the username is the email. This change reduces the amount of information that the user needs to input. It becomes easier to login and more efficient.

## Main Page



## Description:

This is the main 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
  - Logout button will lead to the Start Up Page

This change will make the user perform one additional task by clicking on the button to navigate to the pages: Digital Pantry, Edit Account Info, Find Recipes, Logout. Even though there is one additional step to take, this page makes the app more organized. Note that the Grocery List button is not implemented yet. When the Grocery List is implemented, there should be five buttons total. The Grocery List will lead the user to the Grocery List Page.

## Edit Account Page

### Edit User

Email address: kl828@rutgers.edu

Password

City

State

New Jersey ▾

Diets

Nothing selected ▲

Intolerances




Nothing selected ▲

### Description:

- When in the *Account Page*, the user can edit their accounts. The *Account Page* consists of:
  - Password - The user can change their password by inputting a new password into the field.
  - 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.

Digital Pantry subproject:

## Digital Pantry

My Digital Pantry							
	Name	Quantity	Weight	Volume	Expiration		
	Tomatoes	4	N/A	N/A	04/09/21	<a href="#">Edit</a>	<a href="#">Delete</a>
	Bread	4	1000 grams	N/A	04/10/21	<a href="#">Edit</a>	<a href="#">Delete</a>
	Oil	4	N/A	800 fl.oz	08/05/21	<a href="#">Edit</a>	<a href="#">Delete</a>
<a href="#">Add +</a> <a href="#">Find Recipes</a> <a href="#">Back</a>							

Description:

- When in the *Digital Pantry*, the user will be able to see what is in their digital pantry along with key information: ingredient name, quantity, weight in grams, volume in milliliters, and expiration date.
- The user has the two main actions in the digital pantry:
  - The *Add+* button opens up the *Add Item* page, a separate window that allows users to manually add ingredients and information. Once the ingredient is added, the pop-up will close and the *Digital Pantry* will refresh, showing the new ingredient(s) included.
  - The *Find Recipes* button opens the *Recipe Generator* page.
  - The *Edit* link for each existing ingredient allows users to edit the amount of said ingredient in the *Digital Pantry*.
  - The *Delete* link for each existing ingredient allows users to delete the said ingredient from the *Digital Pantry*.
- These are the following deviations from our initial UI design:
  - When adding an ingredient, we no longer open up a popup. Instead, we redirect the user to an entirely new window to input the ingredient.
  - We can now go to the *Recipe Generator* directly from the *Digital Pantry* page through a button click. Our initial UI design relied on switching tabs.
  - The explicit *Edit* button was included for more transparency when a user would want to simply update an ingredient instead of searching for it again.

## Add Ingredient Page

### Add item

Item Name

Quantity

Set to 0 for N/A

Volume

Set to 0 for N/A

Volume Units

Ignore if volume is N/A

Weight

Set to 0 for N/A

Weight Units

Ignore if weight is N/A

Expiration Date

Image Link

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

Add item

Back

### 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.

### Recipe Generator subproject:

#### Recipe Generator

## Recipe Finder

### Pantry Ingredients

	Name	Quantity	Add to Search
	Tomatoes	4	<input type="checkbox"/>
	Bread	4	<input type="checkbox"/>
	Oil	4	<input type="checkbox"/>

### Additional Ingredients

### Specific Cuisines

Nothing selected ▾

Leave empty for all cuisines

### Diets

Nothing selected ▴

### Intolerances

Nothing selected ▴

#### Description:

- The Recipe Generator 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 Recipe Generator 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 “Back” to go back to the previous page (this could be the Main Page or the Digital Pantry page).
- Deviations from original design:
  - The new design allows for input for cuisine and diet preferences, as well as food intolerance. This allows for more tailored recipe searches, which would provide a better user experience, especially in the case that the user may be cooking for multiple people with various conditions and preferences.
  - We also changed checking the entire Digital Pantry to just checking each individual ingredient from it. This also allows for more tailored recipe searches.
  - We made adding an ingredient to the recipe generator simpler as well for better user experience, allowing a user to just have to enter text for the ingredient they want instead of going through a cumbersome pop-up.
  - The “Back” button is also a new edition, a byproduct of the new design not using tabs.

### Recipes Page

Recipes			
	Name	Used Ingredients	Missing Ingredients
	Roasted Fish Bruschetta	3	6
			<a href="#">View Recipe</a>
<div>Back</div>			

### 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. We noticed that we were limited in the number of Spoonacular API calls that we could use, so we decided to eliminate the Meal Page by navigating the user to the source of the recipe itself.
- Due to such limitations, 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 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.

### Grocery List subproject:

Note: The Grocery List subproject is not yet implemented, but will be implemented for Demo 2. The following are the mockups that we designed previously.

#### Grocery List

Ingredients You Need		Add Ingredient			
	Ingredient	Quantity	Weight (g)	Volume (ml)	Expiry
	Tomatoes	2	N/A	N/A	2/18/21
	Oil	N/A	N/A	1000	1/4/22

[Find Store](#)

#### Description:

- When in the Grocery List, the user will be able to see what is in their grocery list along with key information: ingredient name, quantity, weight in grams, volume in milliliters, and expiration date.
- The user has three main actions in the Grocery List:
  - By clicking on the Add Ingredient button, it opens a pop-up window that allows users to manually add ingredients and information. Once the

ingredient is added, the pop-up will close and the grocery list will refresh, showing the new ingredient(s) included.

- The Trash Can button allows the user to manually delete an ingredient from their grocery list. Upon clicking the Trash Can button, the ingredient will be deleted and the grocery list will refresh.
- By clicking on the Find Store button, it will lead to the Choose Store Page. (Shown below)




### Choose Store Page

[Username]			Logout
Digital Pantry	Grocery List	Recipe Generator	Account
Back	Choose Store		
Name	Distance (mi) ^	Total Price (\$) ^	
Wegmans	4	27.32	
Walmart	10	39.18	
Whole Foods	12	41.47	

### Description:

- The user can view the list of stores available to them. The list contains the store name, distance, and total price.
- By clicking on the arrow icon next to the “Distance” or “Total Price” headers, 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 when they click it.

Store Page (*Back* → Choose Store Page)

[Username]		Logout
Digital Pantry	Grocery List	Recipe Generator Account
Back	Wegmans	
  	Ingredient	Price (\$)
	Tomatoes	9.23
	Bread	10.36
	Oil	7.73
Total Price:		\$27.32

Description:

- In the Store Page, the user can view the different ingredients, the price for each ingredient and the total price.
- The user can click on the Back button to return back to the Choose Store Page.

## Design of Tests

### Test Cases

<p>Test Case Identifier: TC-1</p> <p>Use Case Tested: UC-1: Signup</p> <p>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.</p> <p>Input Data: username (email) and password</p>	
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. Each credential is valid and the username/email does not already have an account associated with it.	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.

<p>Test Case Identifier: TC-2</p> <p>Use Case Tested: UC-2: Track Pantry, UC-12: View Pantry Ingredients</p> <p>Pass/Fail Criteria: The test will pass if the user is able to see a list of ingredients in their digital pantry.</p> <p>Input Data: session data</p>	
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.

<p>Test Case Identifier: TC-3</p> <p>Use Case Tested: UC-3: Login</p> <p>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.</p> <p>Input Data: username (email) and password</p>	
Test Procedure:	Expected Result:
Test 1: The user enters an invalid username.	ChefCart fails to let the user login and shows an “Invalid Credentials” message.
Test 2: The user inputs a valid username and 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 valid username, but the wrong password.	ChefCart fails to let the user login and shows an “Invalid Credentials” message.

<p>Test Case Identifier: TC-4</p> <p>Use Case Tested: UC-4: Create Grocery List</p> <p>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 and will be able to see their grocery list update when they add and remove desired ingredients. The test will fail if the automatically generated grocery list does not restock every ingredient or if the user is unable to make changes to the list.</p> <p>Input Data: User inputted ingredients, edits and Digital Pantry data.</p>	
Test Procedure:	Expected Result:
Test 1: The user enters the Grocery List 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 2: The user adds, removes and edits the grocery list ingredient.	The displayed grocery list is updated to reflect the user’s desired changes.

<p>Test Case Identifier: TC-5</p> <p>Use Case Tested: UC-5: View Grocery Stores</p> <p>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.</p> <p>Input Data: session data, user specific grocery list</p>	
Test Procedure:	Expected Result:
Test 1: The user clicks on the "View Stores" button on the Grocery List page.	A list of stores with the desired items on the grocery list is displayed, which each store showing its distance from the user, and the total cost of the grocery list items.

<p>Test Case Identifier: TC-6</p> <p>Use Case Tested: UC-6: Sort Grocery Stores</p> <p>Pass/Fail Criteria: The test will pass if the stores are sorted according to the distance and price ascending or descending as indicated by the user. The test will fail if the sort does not work.</p> <p>Input Data: Sorting options for sorting ascending or descending for store distance and grocery list price indicated by the state of the arrow buttons.</p>	
Test Procedure:	Expected Result:
Test 1: Sort stores by distance ascending by toggling the arrow button.	All the stores are displayed by increasing distance.
Test 2: Sort stores by distance descending by toggling the arrow button.	All the stores are displayed by decreasing distance.
Test 3: Sort stores by price ascending by toggling the arrow button.	All the stores 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 stores by price ascending and then by distance ascending by toggling the arrow buttons.	All the stores are displayed by increasing price and stores with the same price will be sorted by distance ascending.
Test 6: Sort stores by price descending and	All the stores are displayed by decreasing

then by distance ascending by toggling the arrow buttons.	price and stores with the same price will be sorted by distance ascending.
Test 7: Sort stores by price ascending and then by distance descending by toggling the arrow buttons.	All the stores are displayed by increasing price and stores with the same price will be sorted by distance descending.
Test 8: Sort stores by price ascending and then by distance descending by toggling the arrow buttons.	All the stores are displayed by decreasing price and stores with the same price will be sorted by distance descending.
Test 9: Sort stores by distance ascending and then by price ascending.	All the stores are displayed by distance ascending and stores that are the same distance will be sorted by price ascending.
Test 10: Sort stores by distance ascending and then by price descending .	All the stores are displayed by distance ascending and stores that are the same distance will be sorted by price descending .
Test 11: Sort stores by distance descending and then by price ascending.	All the stores are displayed by distance descending and stores that are the same distance will be sorted by price ascending.
Test 12: Sort stores by distance descending and then by price descending .	All the stores are displayed by distance descending and stores that are the same distance will be sorted by price descending .

<p>Test Case Identifier: TC-7</p> <p>Use Case Tested: UC-7: Discover Recipes</p> <p>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.</p> <p>Input Data: digital pantry ingredients, cuisine input, intolerance input, additional ingredients</p>	
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	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.



valid ingredients.	
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 valid and invalid ingredients.	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.

<p>Test Case Identifier: TC-8</p> <p>Use Case Tested: UC-8: Sort/Filter Recipes</p> <p>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.</p> <p>Input Data: none</p>	
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.

<p>Test Case Identifier: TC-9</p> <p>Use Case Tested: UC-9: Viewing Recipe, UC-21: Show Missing Recipe Ingredients</p> <p>Pass/Fail Criteria: The test will pass if the user is able to view the recipes and all the information relating to it.</p> <p>Input Data: recipe link</p>	
Test Procedure:	Expected Result:
Test 1: The user can click on the recipe and then the user will be able to view the information relating to the recipe.	ChefCart 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.

<p>Test Case Identifier: TC-10</p> <p>Use Case Tested: UC-10: Account Editing</p> <p>Pass/Fail Criteria: The test will pass if the user is able to edit their account in the Account Page.</p> <p>Input Data: session data</p>	
Test Procedure:	Expected Result:
Test 1: The user can edit the account. The user can change the password, city, state, diets, and tolerances. After clicking submit, the information will be saved.	ChefCart will save/update all the new information.

<p>Test Case Identifier: TC-11</p> <p>Use Case Tested: UC-11: Logout</p> <p>Pass/Fail Criteria: The test will pass if the user is able to successfully log out of the web app.</p> <p>Input Data: none</p>	
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.
---	---

<p>Test Case Identifier: TC-12</p> <p>Use Case Tested: UC-13: Add Pantry Ingredients</p> <p>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.</p> <p>Input Data: New ingredient name, quantity, weight, volume, expiration, and warning threshold or new existing ingredient quantity, weight and volume.</p>	
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 new quantity, weight, volume.	The digital pantry is redisplayed to show the edited ingredient and the changes are saved in the database.

<p>Test Case Identifier: TC-13</p> <p>Use Case Tested: UC-14: Remove Pantry Ingredients</p> <p>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.</p> <p>Input Data: session data, updated existing ingredient quantity, weight and volume.</p>	
Test Procedure:	Expected Result:
Test 1: The user presses the delete icon right	ChefCart will delete that ingredient from the

next to an ingredient row.	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.

<p>Test Case Identifier: TC-14</p> <p>Use Case Tested: UC-15: View List Ingredients</p> <p>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.</p> <p>Input Data: Session data, Digital Pantry data</p>	
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.

<p>Test Case Identifier: TC-15</p> <p>Use Case Tested: UC-16: Add List Ingredients</p> <p>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.</p> <p>Input Data: New ingredient quantity, volume and weight or existing ingredient quantity, volume and weight.</p>	
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.
---	--

<p>Test Case Identifier: TC-16</p> <p>Use Case Tested: UC-17: Remove List Ingredients</p> <p>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.</p> <p>Input Data: May use existing ingredients new quantity, volume, and weight.</p>	
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.

<p>Test Case Identifier: TC-17</p> <p>Use Case Tested: UC-18: Search Ingredients</p> <p>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.</p> <p>Input Data: ingredient name</p>	
Test Procedure:	Expected Result:
Test 1: The user inputs a valid ingredient.	The ingredient will be found and displayed to the user.
Test 2: The user inputs an invalid ingredient.	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-20: Shop Missing Recipe Ingredients

Pass/Fail Criteria: The test will pass if the user can add missing recipe ingredients to the grocery list.

Input Data: session data

Test Procedure:	Expected Result:
Test 1: The user clicks on a button to add the ingredients.	ChefCart will add the missing recipe ingredients to the grocery list.

## Test Coverage

Our test cases cover all the most essential use cases. The only functionality we are currently missing is the Grocery List feature. In particular, we included the basic functionality for tracking, adding and removing ingredients in the Digital Pantry in our testing. We have the basic functionality for finding recipes based on our ingredients in our tests. 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. An example of a use case that we have not yet implemented is the warning for threshold in the Digital Pantry. In the future, we will test to make sure that if an ingredient amount is lower than the user set threshold, then a warning will be displayed in the Digital Pantry page. If the ingredient amount is higher, nothing will be displayed.

## Integration Testing Strategy

We chose to use bottom-up integration as our strategy for integration testing. This strategy uses 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 and Plan of Work

### 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. 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.

### [Report 3 Update]

As per the last report, we continue to work with a branch and pull request model. We are still using Discord as a discussion platform in order to coordinate work amongst sub-teams and the overall team. Our largest issue is minor refactoring of code for server hosting, as there are some changes that need to be made in order to deploy the application on the heroku servers.

The present functionality includes: user signup, user login, account management, digital pantry management, digital pantry additions, as well as recipe generator for items in the pantry. Currently we are working to recreate all internal api HTTP requests as gRPC and protobuf as these will hopefully run quicker and provide easier debugging since the message types and formats are known. Afterwards we will work on development of the grocery list integration.



## Use Case Status Updates

The ChefCart website is now set up and is being freely hosted at <https://chefcart.herokuapp.com>.

Use Case	Current Status
UC-1	Users are able to signup for ChefCart
UC-2	Users are able to add, edit, and remove items from the digital pantry
UC-3	Users are able to login with their credentials
UC-4	Individual items can be searched on the Wegman's API for price and stock
UC-7	Spoonacular API can be polled for recipes using ingredients inputted by the user. Recipes are generated to include said ingredients.
UC-8	Recipes can be sorted by number of used ingredients and number of missing ingredients ascending or descending.
UC-9	Users are given a link to the Spoonacular website to show the recipe ingredients, prep time, price and instructions for each recipe generated.
UC-10	Users can view and edit their account information. These edits can include their password, city, state, dietary preferences, intolerances and threshold for alerts.
UC-11	Users are able to logout of ChefCart
UC-12	Users can view Digital Pantry ingredients populated by the database.
UC-13	Users can add ingredients and save them to the Digital Pantry.
UC-14	Users can remove ingredients and the Digital Pantry will update.
UC-19	Ingredients below a threshold set by the user will show a warning in the Digital Pantry page.
UC-21	The number of missing ingredients for a recipe are shown to the user.

**Projected Milestones:**

Date	Milestone
3/10	Finish User Edit Backend logic ✓
3/17	Finish & test Digital Pantry Component. Begin integration with User ✓
3/24	View / Edit / Add Digital Pantry items from web app ✓
3/24	Integrate pantry w/ recipe generator & grocery list ✓
4/7	Finish & test recipe generator
4/14	Integrate recipe generator w/ webapp
4/14	Integrate recipe generator w/ grocery list
4/21	Finish & test grocery list
5/5	Integrate w/ webapp & test completed webapp

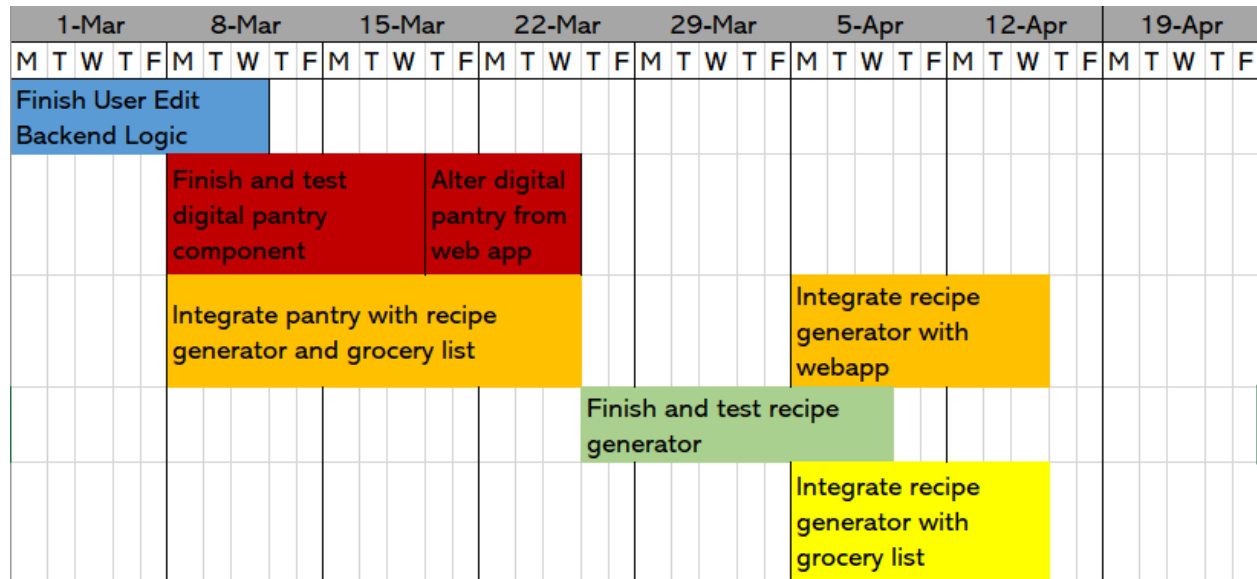
## Responsibilities Breakdown

As stated in the individual contributions section, all students are contributing equally. For the integration side, each subteam has a group member dedicated to integration with other components of the project. Max and Jonathan, in addition to their responsibilities listed below, are leading the overall integration and integration testing of the ChefCart project.

### *Individual Responsibilities:*

- Indrasish Moitra: Setting up BigQuery tables for database storage backend. Learning BigQuery APIs for Go to integrate with webapp, instead of relying on Python.
- Milos Sesar: Exploring Spoonacular API and creating local scripts to test API functions. Restructuring scripts for future Go integration.
- Shreyas Heragu: Locally testing grocery store APIs to finalize code/workflow for store integration to grocery list.
- Kevin Lin: Creating grocery list structure and formatting individual item types. Working with digital pantry to finalize “ingredient” data type.
- Maxwell Legrand: Coding login and setup pages. Developing webapp skeleton and user information input fields. Planning further webapp structures and front end features.
- Jonathan Wong: Planning design features and UI/UX changes to benefit user. Locally testing UX and webapp scripts to develop new features. Managing project reports.
- Allen Chang: Facilitating integration of components into webapp. Communicates changing requirements between teams to encourage communication.
- Elysia Heah: Coding backend edits to database from grocery list. Focusing on add/remove/edit methods from webapp.
- Mark Stanik: Working on frontend aspects of digital pantry. Working with the webapp team on creating interface buttons for user interaction.
- Brandon Luong: Integrating recipe generator with digital pantry backend. Creating functions to leverage spoonacular API with direct database access.

### Plan of Work Gantt Chart



## 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