FoodPants
Iteration III

GITHUB
https://github.com/boothverse/food-pants

WEBSITE (with .jar of FINAL APPLICATION)
https://sites.google.com/view/foodpants/home

PRODUCT BACKLOG
https://trello.com/b/NX1l7UQG

REQUIREMENTS
https://trello.com/b/7aSPYcU2

ISSUE TRACKING
https://github.com/boothverse/food-pants/issues

GIT CONTRIBUTIONS
https://github.com/boothverse/food-pants/graphs/contributors?from=2022-02-06&to=2022-05-07&type=c

JAVADOC
https://github.com/boothverse/food-pants/tree/main/Iteration3/JavaDoc

Project Vision

For people who want an easy way to track their foods and recipes, “FoodPants” is a local executable grocery app that allows users to track their physical pantry in a digital form, as well as their recipes and foods consumed. Unlike existing solutions that commonly segment these ideals into separate apps, our product will allow the user to find out what items they need to cook a recipe and help create a shopping list, as well as track foods consumed and calories in a single location.

Team Members

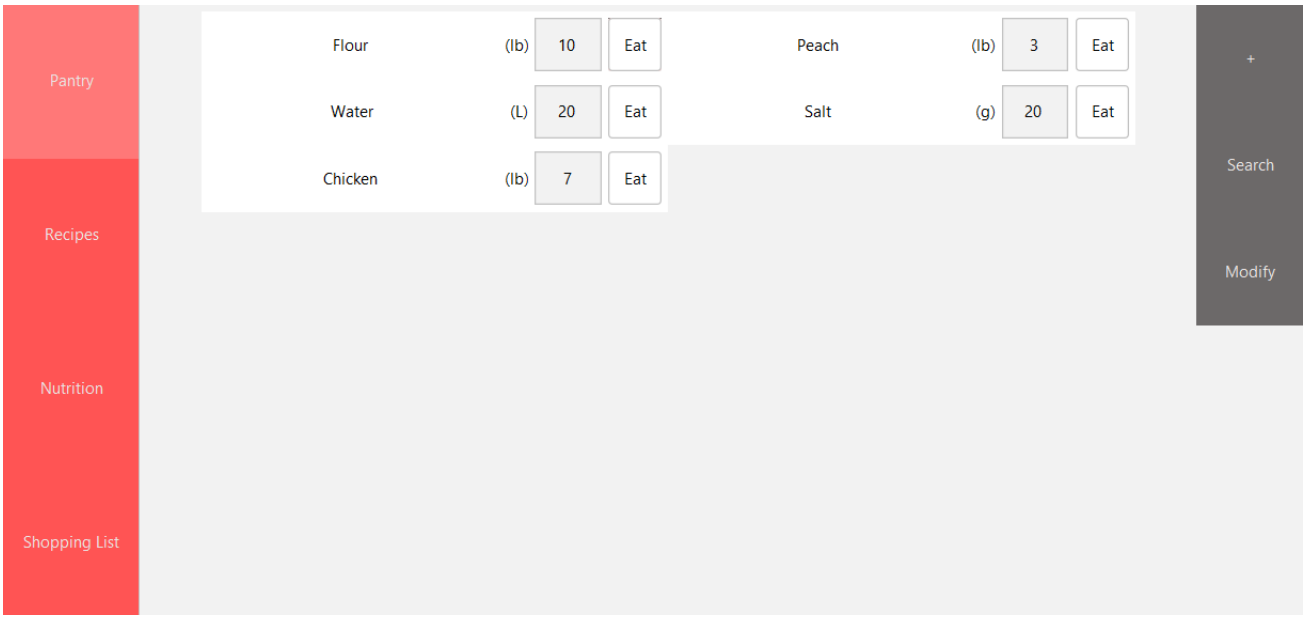
Austin Huizinga (Leader)
Daniel Luper
PJ Wallace
Kurt Wokoek
Patrick Harris
Luka Lelovic

FoodPants User Manual

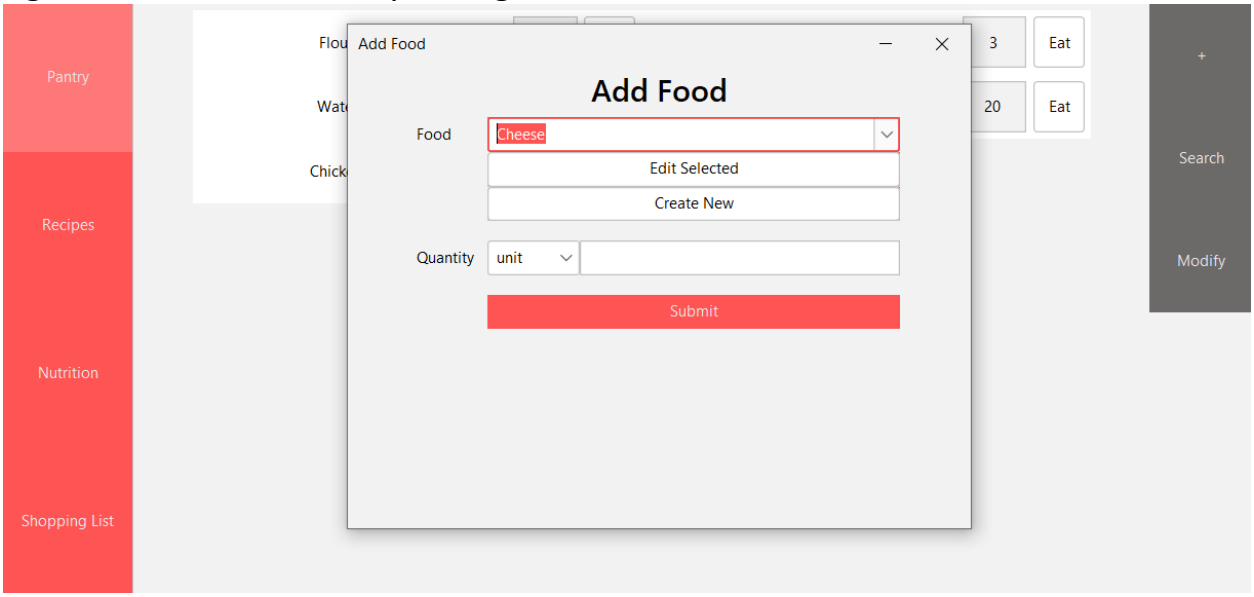
Table of Contents

1.1 – Manage Pantry

Your digital pantry keeps track of all your real-world pantry items you currently have.



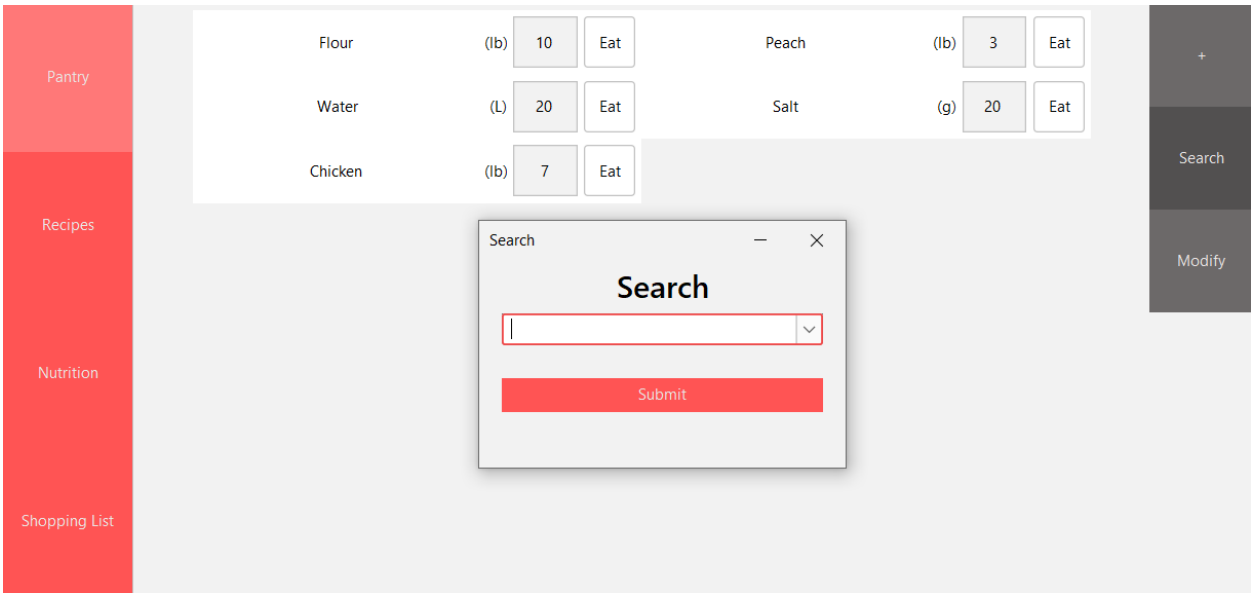
When a new item gets added to your physical pantry, click the “+” button on the right-hand side to add to your digital one!

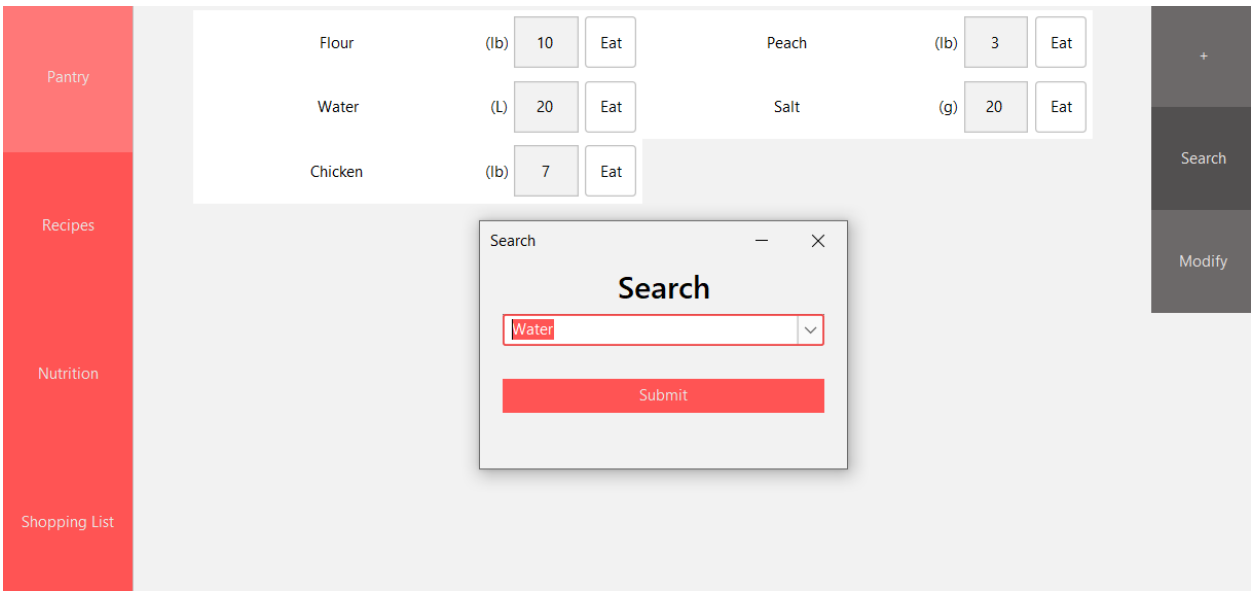


Enter the appropriate quantity, and press submit!

1.2 – Search Pantry

If you would like to find a specific item in your pantry, you can do this easily by searching for it. Simple click the “Search” button on the right-hand side.

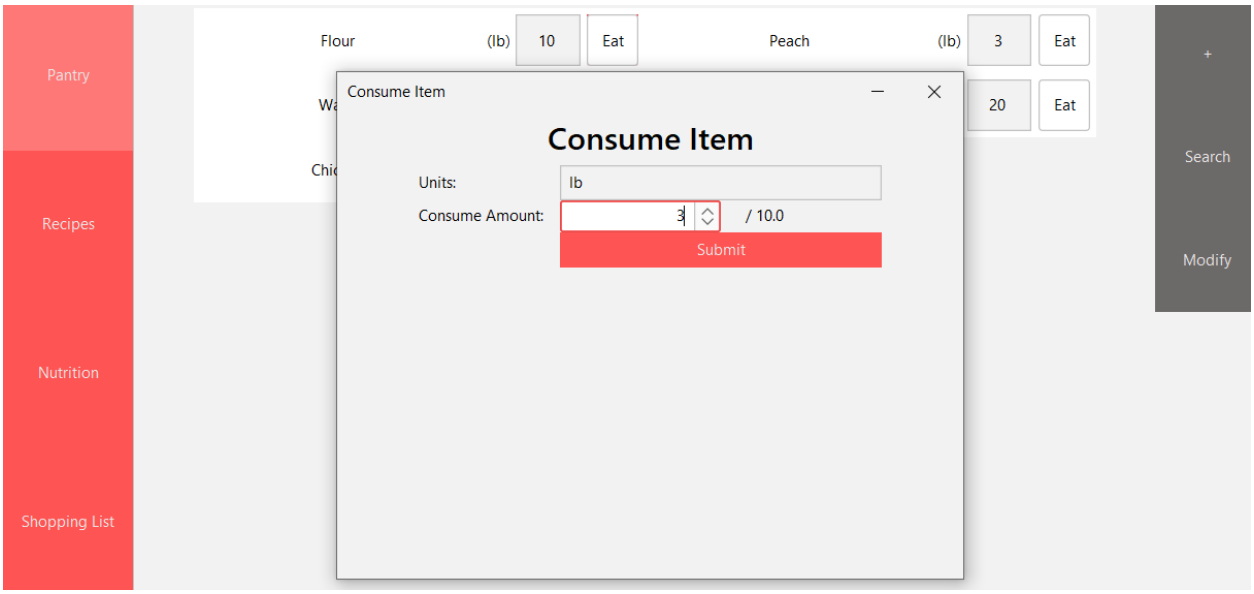




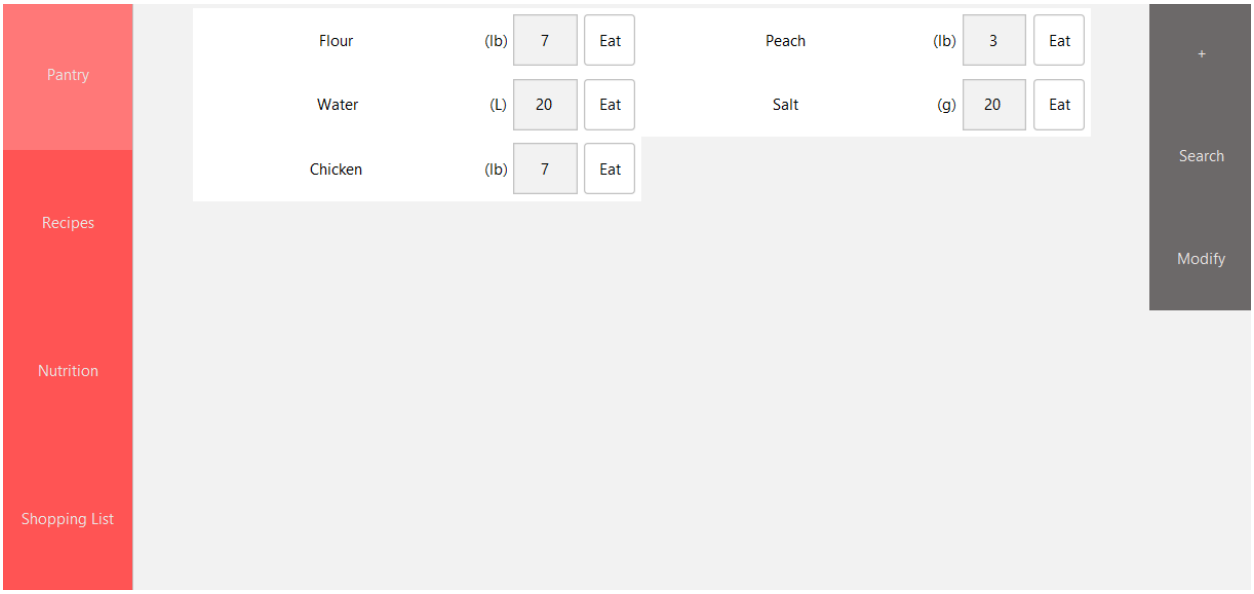
After you type in the item you want to find, click submit! If the item is in your pantry, it will be displayed!

1.3 – Consume Pantry Item

An item in your pantry will not be there forever! Whenever you consume an item, you can track this in your digital pantry. Simple press the “Eat” button the item you consumed.

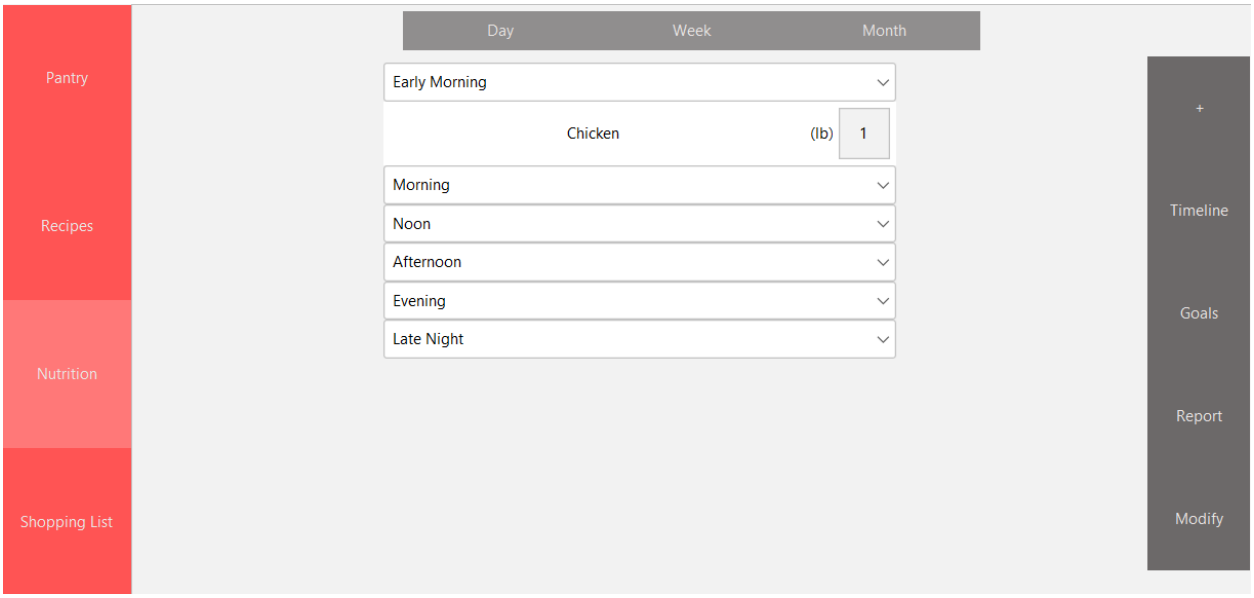


Specify how much out of the total amount you have consumed. For example, in the image above I removed 3 pounds of flour out of my pantry.



2.1 – Manage Nutritional Log

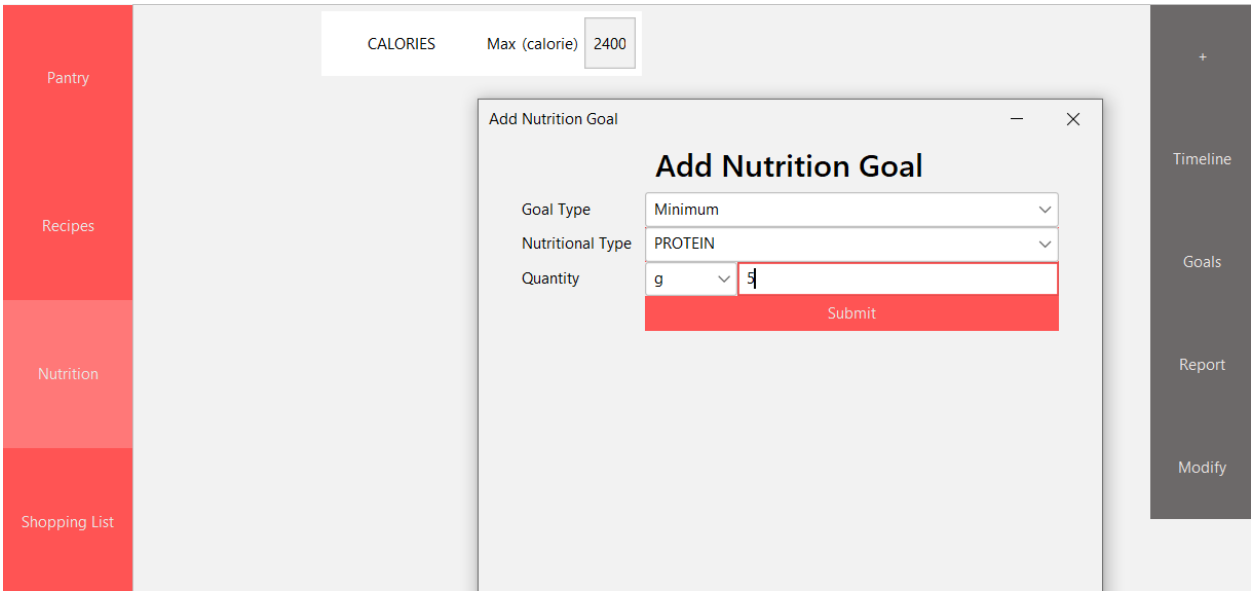
Nutrition page has a timeline of foods consumed, tracked by day, week, and month intervals.



Items can be added to the timeline with the “+” button and modified with the “Modify” button!

2.2 – Manage Nutritional Goals

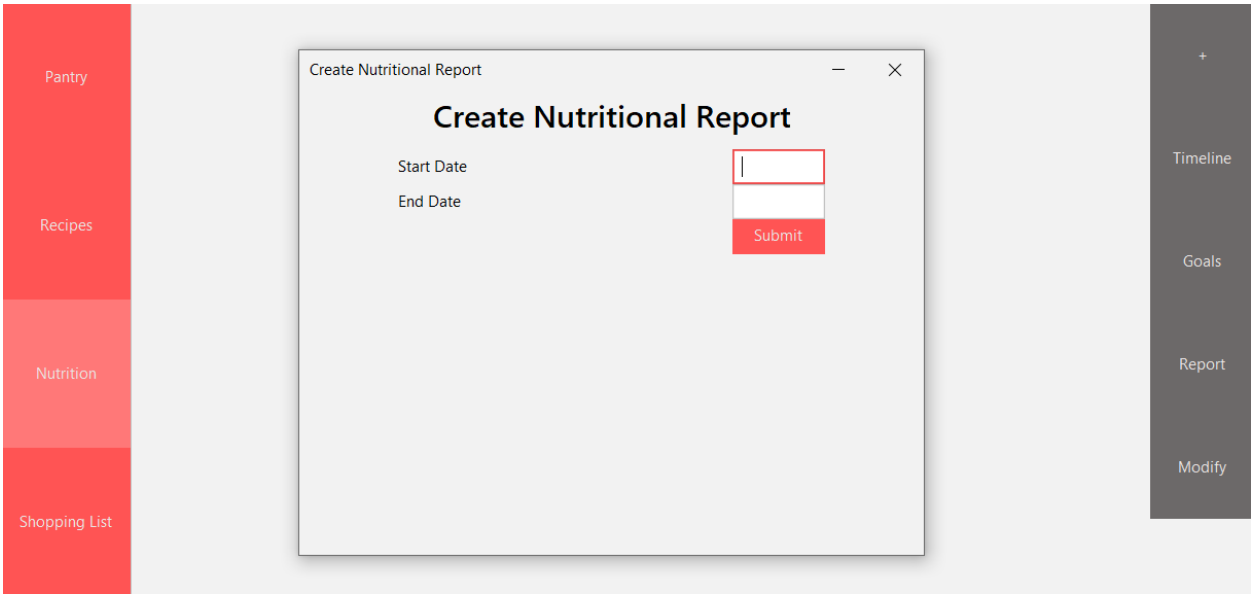
Nutritional goals track how much you want to consume. These are personal goals you set for yourself!



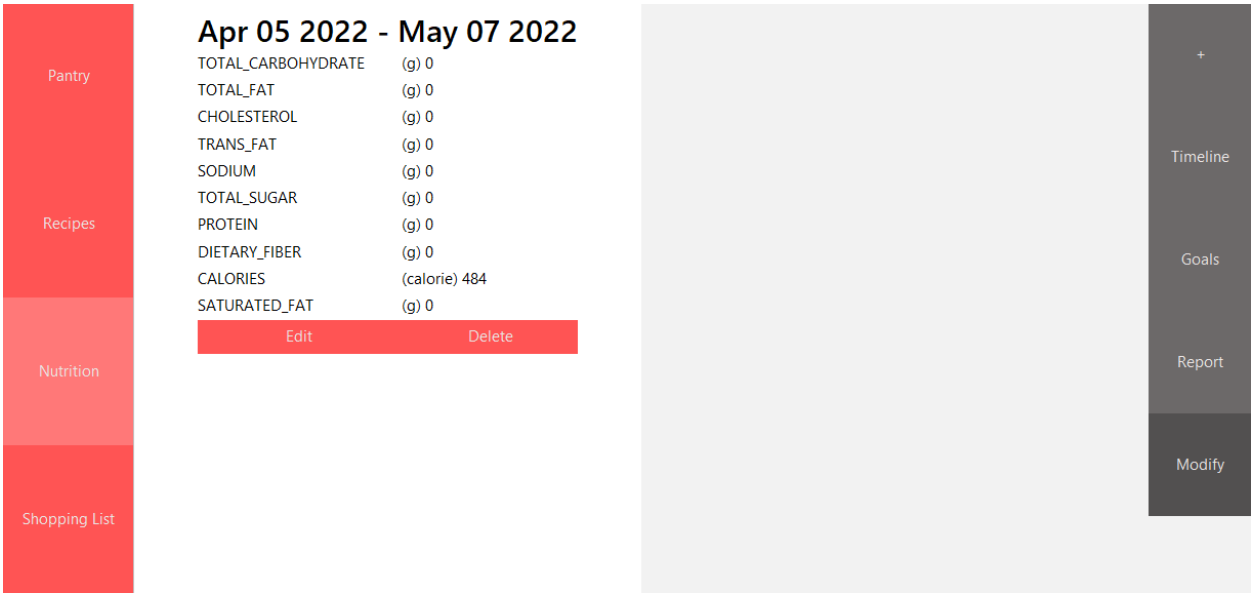
Add to it with the “+” button, and edit or delete a goal with the “Modify” button on the right-hand side!

2.3 – Manage Nutrition Report

A nutritional report can give you the total nutritional information within a certain time frame.



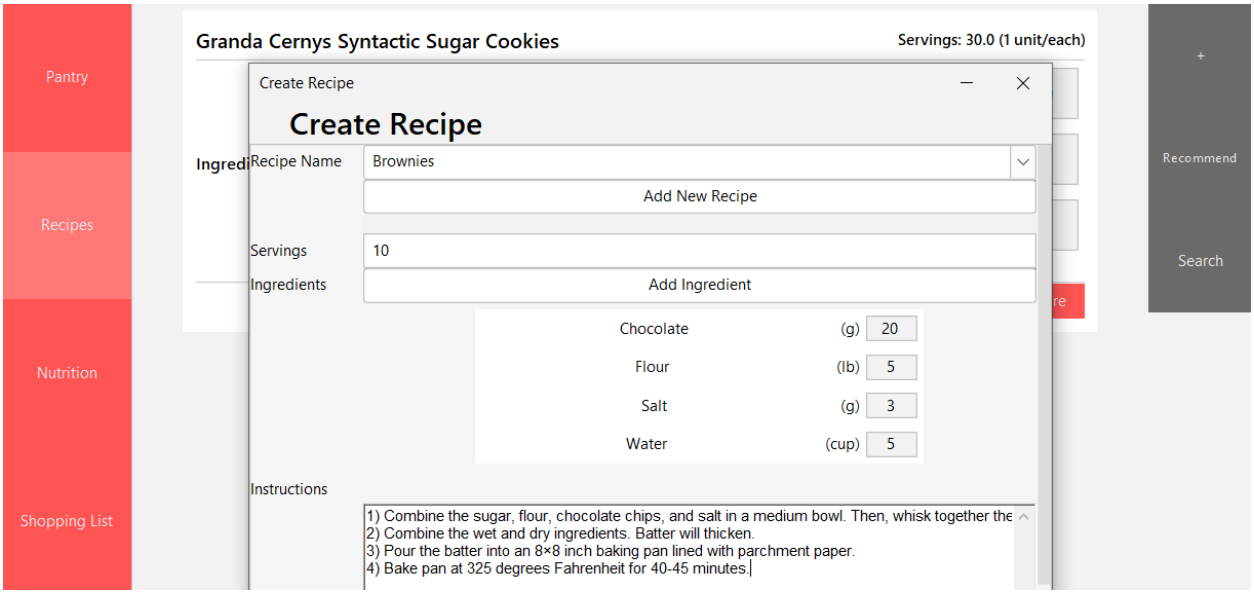
Click the “+” button to generate a new nutritional report!



Click the “Modify” button to edit/delete a nutritional report!

3.1 – Recipe Creation

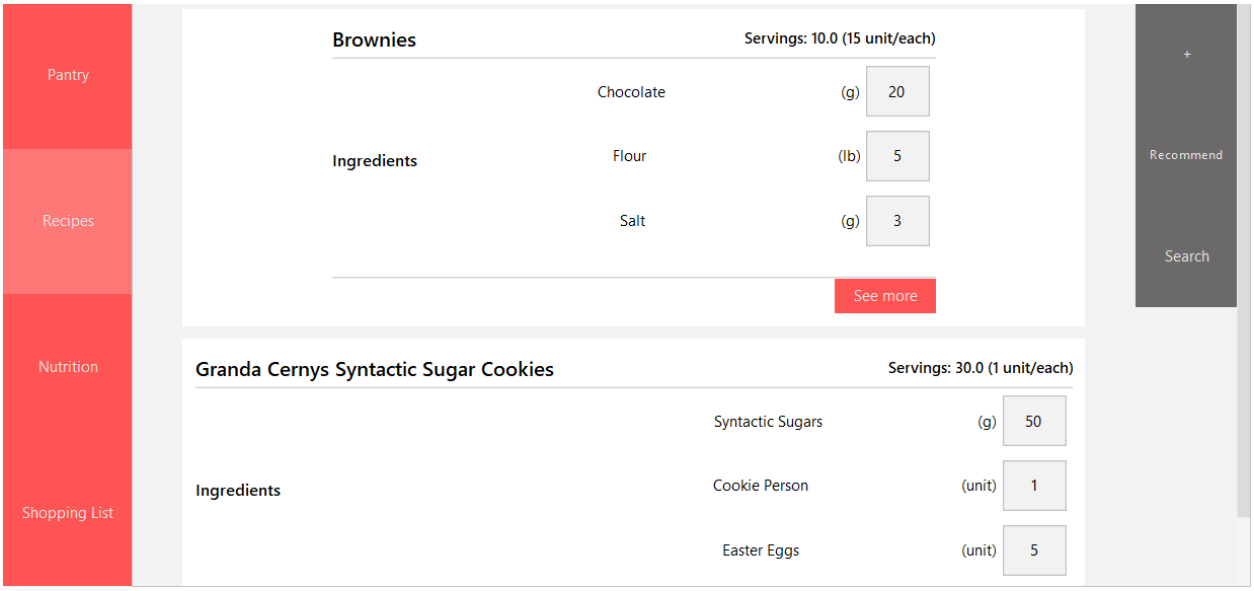
Navigate to the recipe page, and create a recipe easily! Simply click the “+” button and enter in the food name, number of servings, ingredients, and instructions!

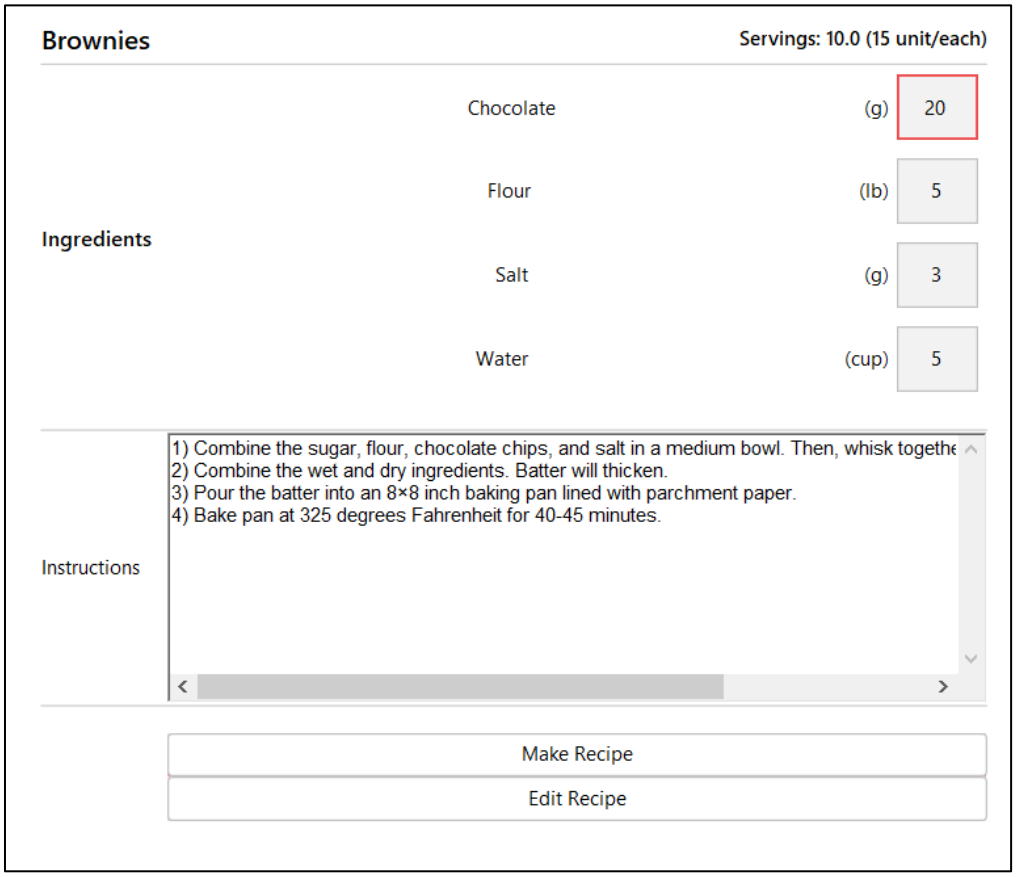


Once you are done, click submit! Your new recipe has been created!

3.2 – View Recipe

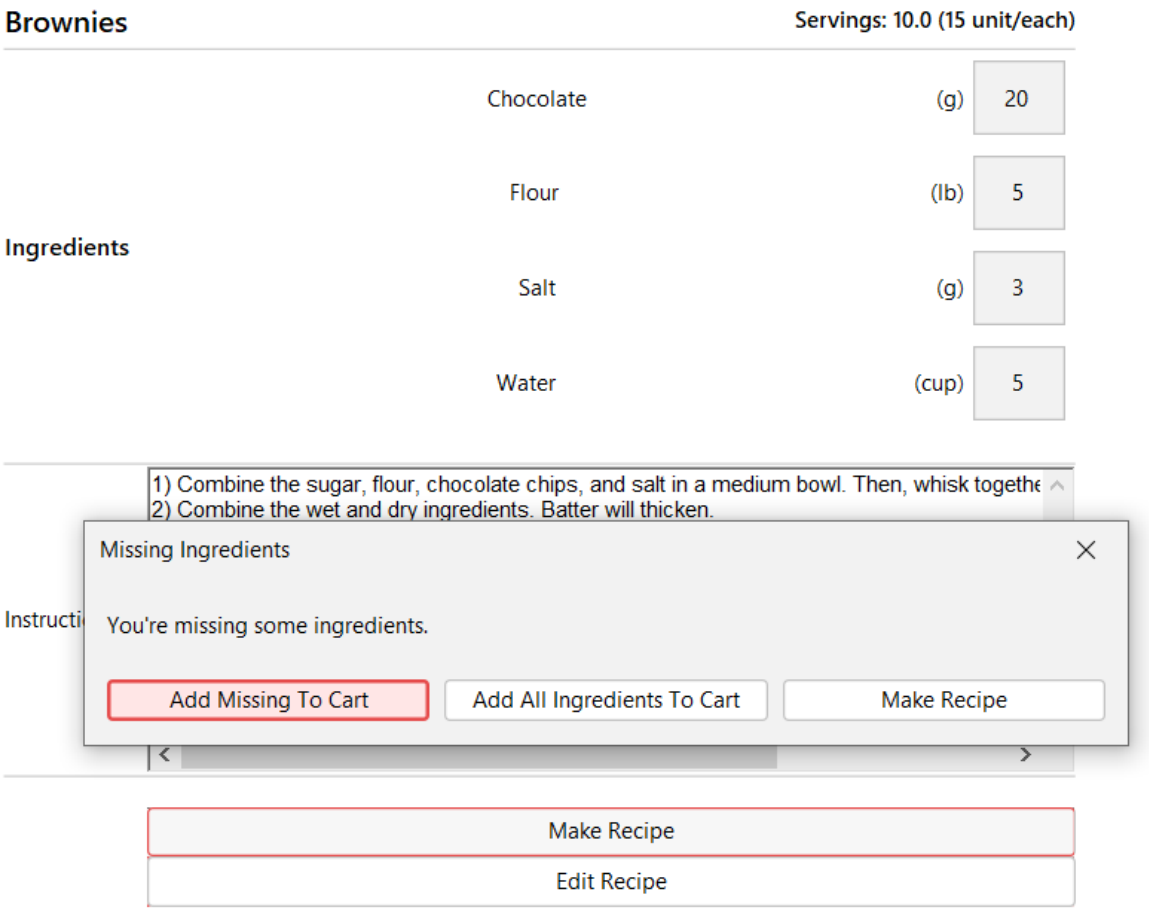
On your recipe page, scroll to view each recipe, and click “See More” to see all the information about each recipe!





3.3 – Recipe to Shopping List

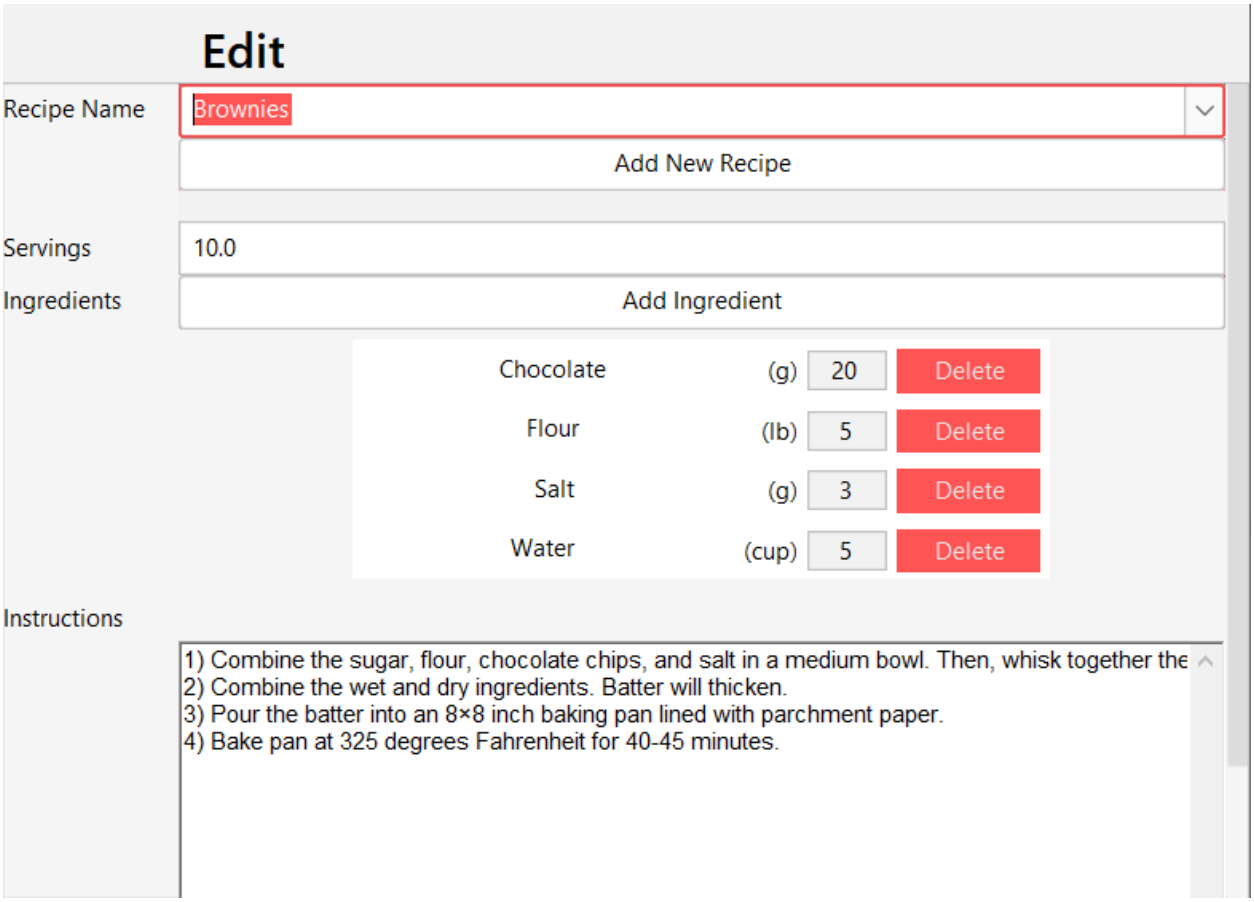
When you want to make your recipe, you can add all recipe ingredients to your shopping list for purchase! Click “See More” on the recipe you want to add, and click “Make Recipe.”



A message will pop up telling you that you are missing ingredients. Simply click “Add All Ingredients To Cart!”

3.4 – Modify Recipe

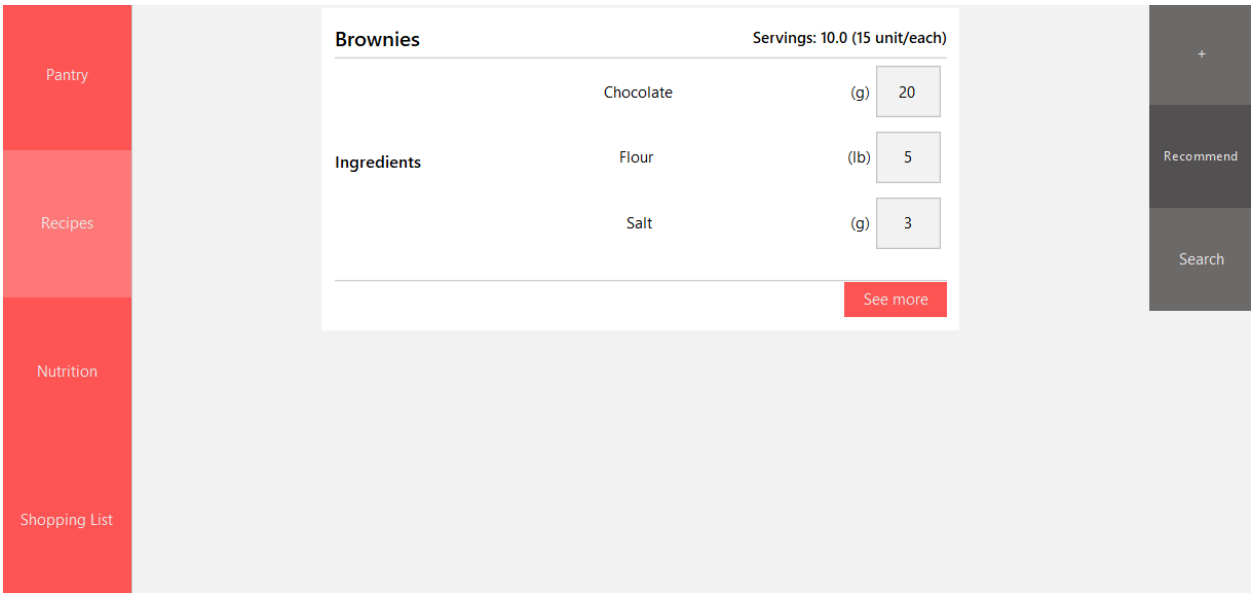
To modify a recipe, click “See More” for the recipe you want to modify, and press the “Edit Recipe” button.



Edit your recipe with the new information you would like to provide, and press submit!

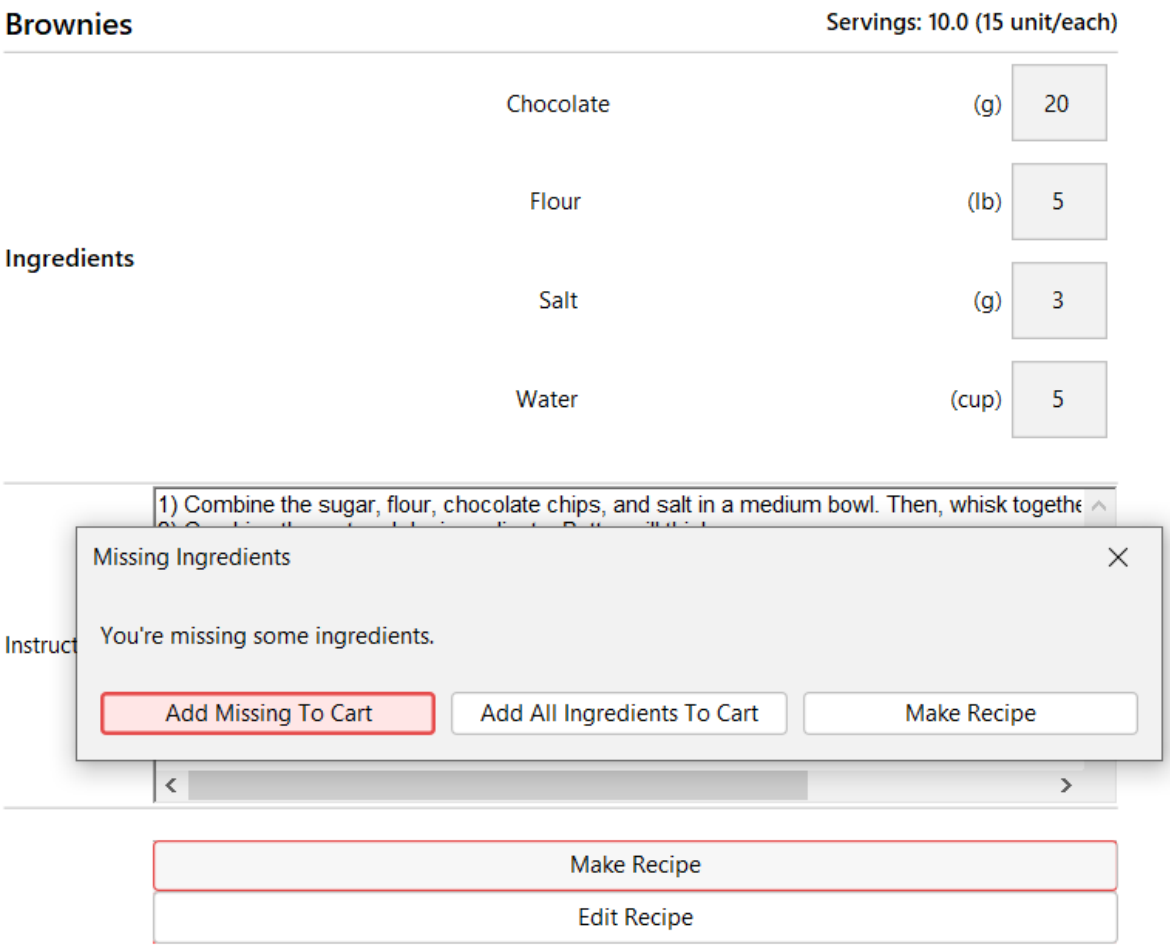
3.5 – Recommend Recipes

If you have created no recipes, one will already be created and recommended to you (“Syntactic Sugar Cookies”). If you have more recipes already created, press the “Recommend” button to get the most recommended recipe to make in your recipe list!



3.6 – Make Recipe

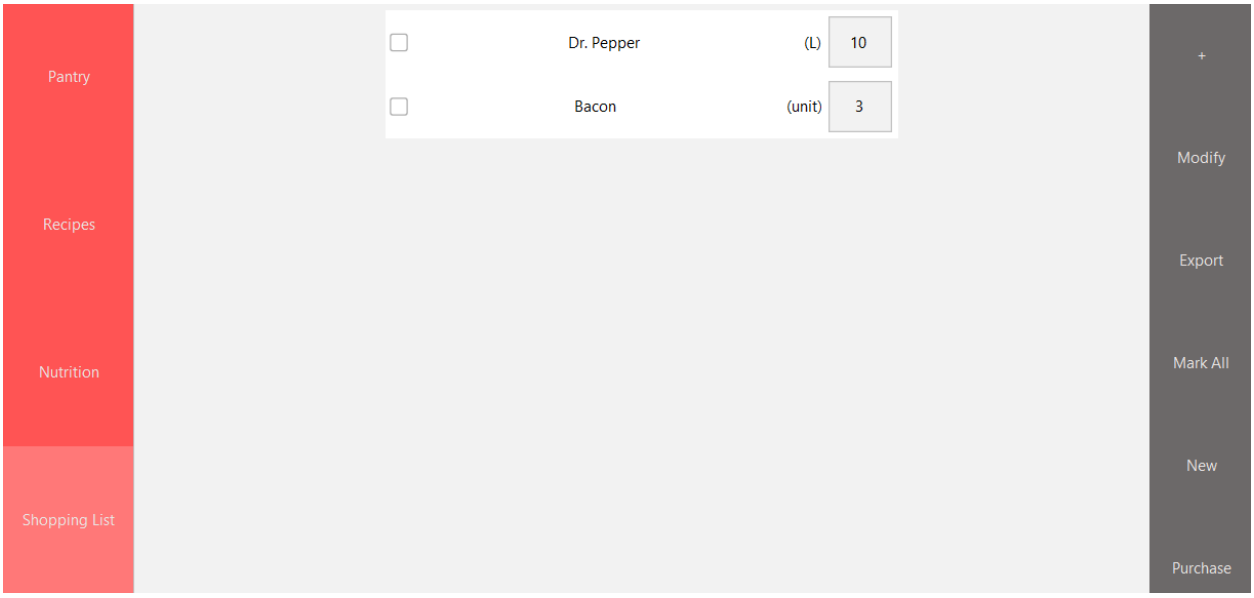
When you want to make a recipe, click “See More” on the recipe, and press “Make Recipe.”



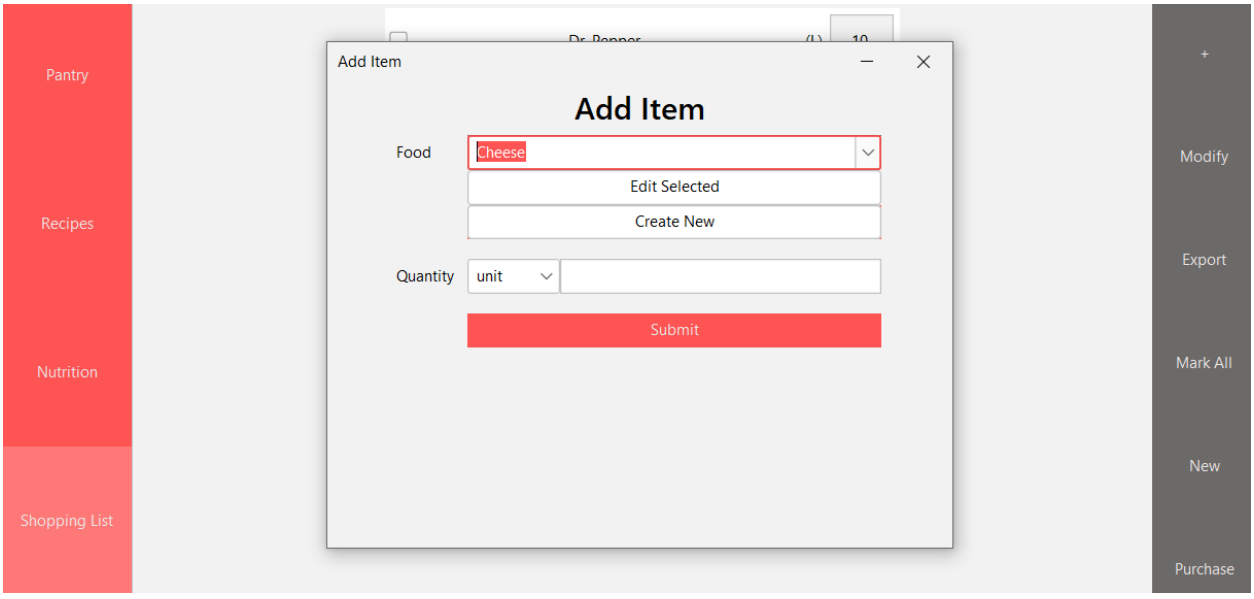
When prompted, click the “Make Recipe” button again! Your recipe has been created and put in your pantry! Congrats!

4.1 – Manage Shopping List

Navigate to the shopping list page. Your shopping list keeps track of all the items you plan on buying!



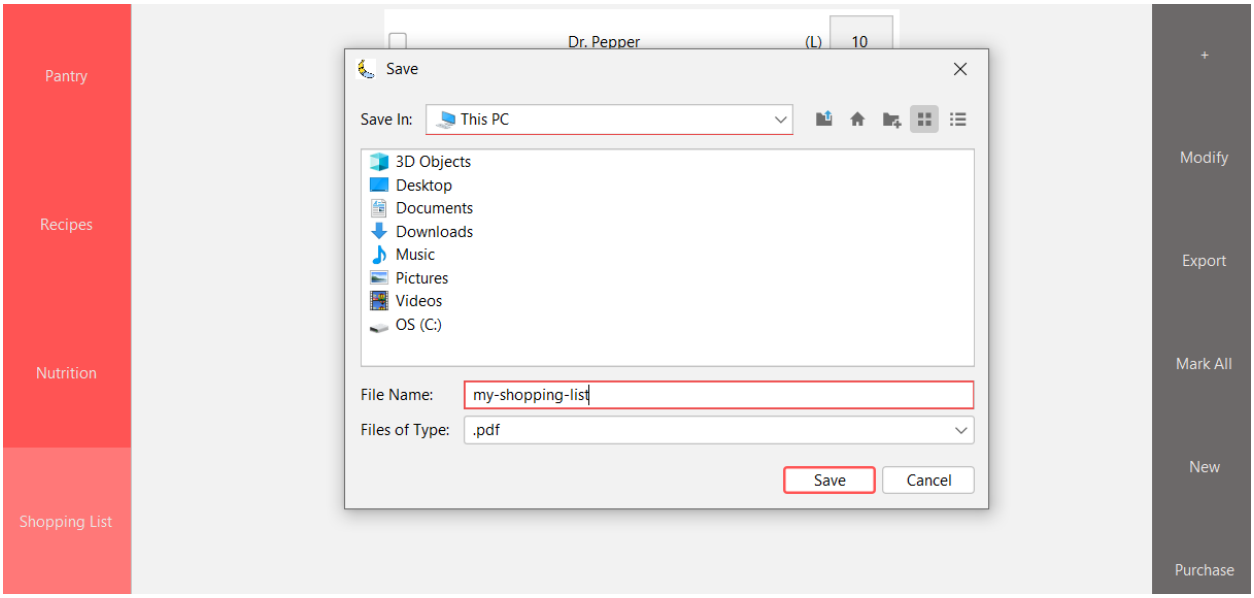
When you want to add to your shopping list a new item, press the “+” button on the right-hand side. A form to add an item will be displayed!



Enter the desired food and quantity, and press submit! If you would like to change your shopping list, press the modify button on the right to edit and delete items!

4.2 – Export Shopping List

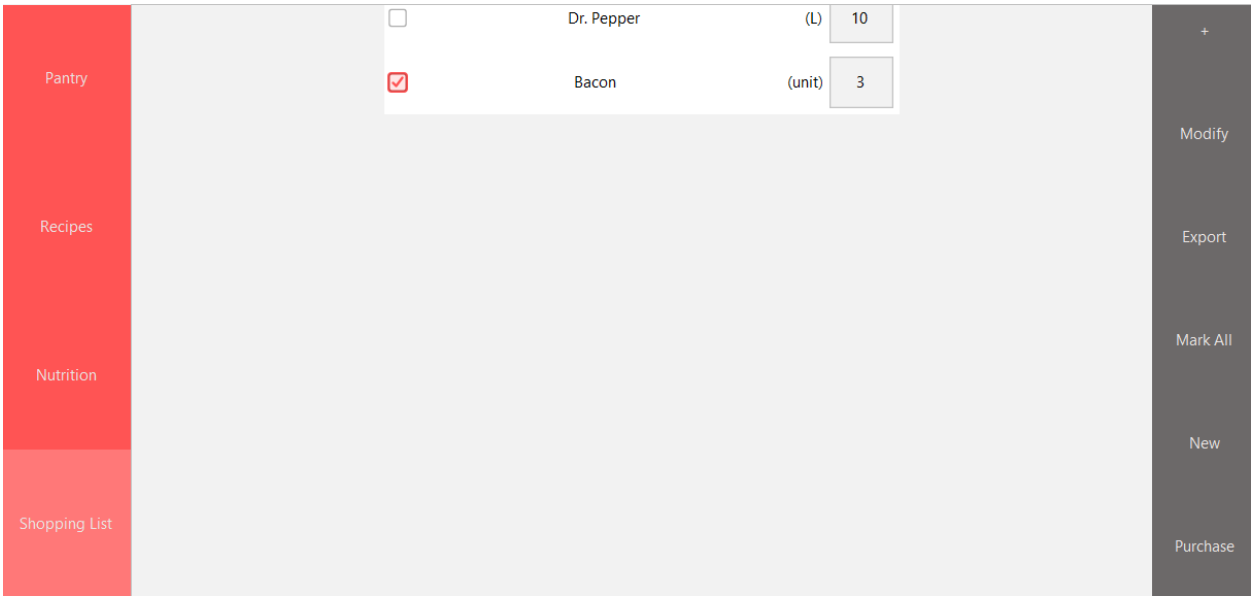
If you would like to export your shopping list to a document in PDF format, to use when you are at the store, press the “Export” button on the right-hand side!



Navigate to where you would like to save your shopping list, then press “Save”

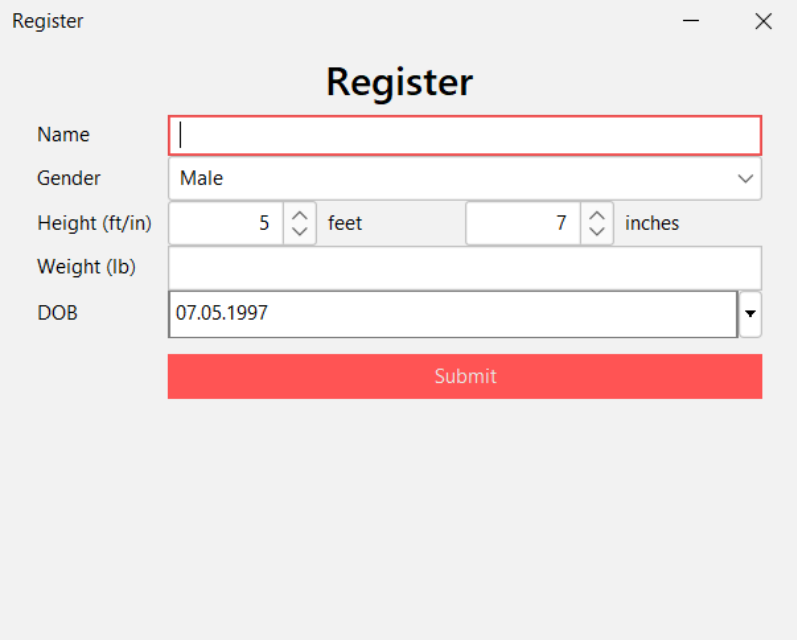
4.3 – Mark Purchased Items

Have you already purchased an item in your shopping list? Simply mark it with the checkbox next to the associated item, and press the “Purchase” button on the right-hand side!



5.1 – Startup

When opening FoodPants for the first time, you will be prompted to enter in your information!

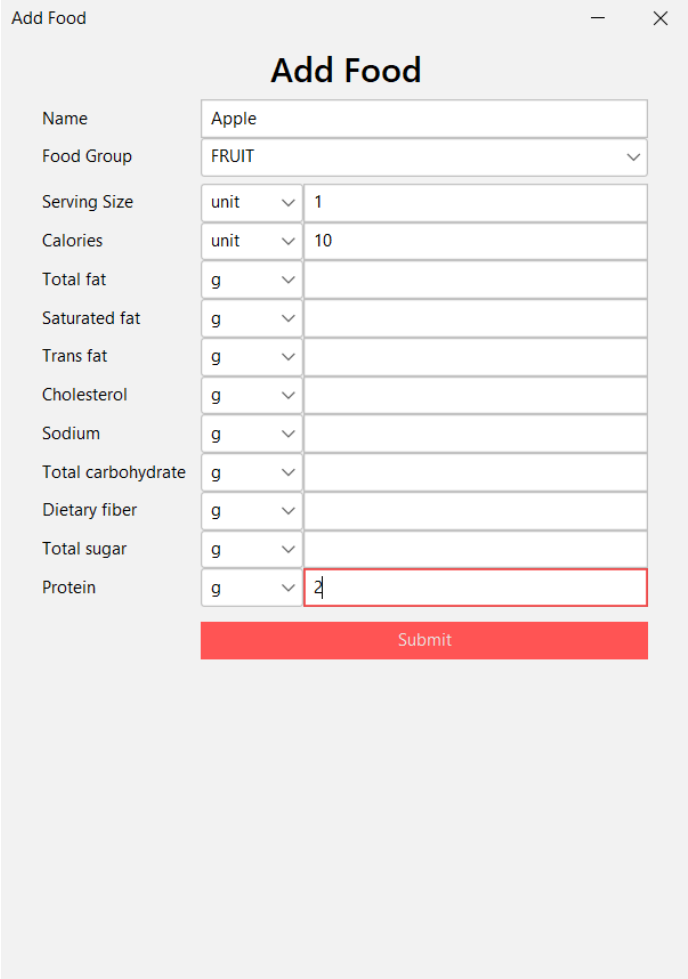


Simply provide your name, gender, height, weight, and date of birth in the respective fields, and press submit! Welcome to FoodPants!

6.1 – Manage Food Types

Foods are an essential part of FoodPants (hence the name). If you would like to add new ones (spoiler alert) this is quite easy!

Creating new food types can be done on multiple pages. For this guide, the pantry page will be shown. On the pantry page, click the “+” button to add a new pantry item, and within that popup, click “Create New”



Hope this user manual has been helpful in guiding you through how to use FoodPants!

Use Cases

ID UC 1.1 Manage Pantry
Scope Pantry system
Level User goal
Stakeholders and interests

- **User:** Person interested in managing a digital pantry. Wants the ability to view food items currently in the pantry. Wants the ability to add food items to their pantry. Wants the ability to modify food items in their pantry. Wants the ability to remove food items from their pantry.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: User has gone through setup process. User wants to manage pantry.
Postcondition: Pantry is updated with any user modifications.

Main success scenario:

1. User navigates to the "Pantry" page.
2. System displays a list of all food items in the pantry.
3. User presses the "Add Item" button.
4. User enters the name of the food type to add to pantry in a search box.
5. System displays food type search results in a drop-down menu.
6. User selects the food type to add.
7. User enters the quantity/amount of the food item to add to pantry.
8. System adds food item to pantry.

User repeats steps 3-8 until satisfied with the pantry

Extensions:

a.* Anytime the user decides to stop managing the pantry

1. *User presses the back button*
   a. *User has unsaved changes*
      i. *System prompts user to save their changes*
      ii. *User presses "OK"*
      iii. *System redirects user to the a pantry overview page where all food items in the pantry are displayed*

4.a User wants to completely remove a food item from the pantry

1. *User selects trash icon next to food item*
2. *System prompts the user for confirmation*
3. *a. User presses the "Yes" button*
   i. *System deletes the food item from the pantry*

b. User presses the "No" button

4.b User wants to change the quantity/amount of an existing food item

1. *User selects the pencil button next to a food item to edit it*
2. *User selects the quantity/amount text box*
3. *User enters a new quantity/amount*

4.c User wants to consume an item in the pantry

1. *User selects the "Consume" option next to a food item*
2. *extend <consumePantryItem>*

4.d User wants to search for an item

1. *extend <searchPantry>*

6.a No results meaning food type is not registered in the food type database

1. *User selects the "Add food type" option from the end of the drop-down menu*
2. *extend <addFoodType>*

9.a Food item already exists in pantry

1. *System adds quantity of food item user entered to quantity of food item already in the pantry*

ID UC 1.2 Search Pantry
Scope Pantry system
Level User goal
Stakeholders and interests

- **User:** Person interested in searching the pantry. Wants the ability to find a food item in the pantry.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: User has gone through setup process. User wants to search the pantry.
Postcondition: Food items in the pantry are displayed based on user search criteria.

Main success scenario:

1. User navigates to the "Pantry" page to view the pantry.
2. System displays pantry
3. User navigates to the search bar along the top.
4. System permits input into search bar
5. User types in the search bar.
6. System displays food items in the pantry in a list with names that contain the user's entered search keyword.

User repeats steps 4-5 until done searching the pantry

Extensions:

a.* Anytime the user decides to stop searching the pantry
    1. *User presses the back button*
    2. *System returns full list of food items in the pantry*

5.a Pantry is empty
    1. *System tells users that there are no food items in the pantry*
    2. *System prompts user if they would like to add a food item to the pantry*
    3. *User clicks "Add Item" button*
    4. *extend <managePantry>*

5.b No food items in pantry contain user search criteria in name
    1. *System tells the user no results were found*
    2. *System prompts user if they would like to add a food item to the pantry*
    3. *User clicks "Add Item" button*
    4. *extend <managePantry>*

ID UC 1.3 Consume Pantry Item
Scope Pantry system
Level User goal
Stakeholders and interests

- **User:** Person interested in consuming an item from the pantry. Wants the ability to decrement the quantity of a food item in the pantry.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: Food types to purchase are registered in the food type database. User wants to consume a pantry item.
Postcondition: Shopping list modifications are saved.

Main success scenario:

1. User navigates to the “Pantry” page to view the pantry.
2. System displays the pantry to the user
3. User navigates and selects the food item they want to consume.
4. System displays the particular food item information
5. User clicks “Consume” option.
6. System decrements quantity of food item on pantry.

User repeats steps 3-5 until consumed all food items they wish to

Extensions:

a.* Anytime the user decides to stop consuming items from the pantry
    1. *User presses the back button*
    2. *System returns full list of food items in the pantry*

5.a Food item is decremented to zero
    1. *System removes food item from the pantry*

ID UC 2.1 Manage Nutritional log
Scope Nutrition Log system
Level User goal
Stakeholders and Interests

- **User:** A person utilizing the Nutrition Log system within the FoodPants application to view, add, remove, or edit an item.
- **System maintainer:** Person responsible for application execution.

Precondition: User has undergone setup process. User wants to manage their nutritional log.
Postcondition: Nutritional log updated with modifications.

Main success scenario:

1. User clicks navigates to "Nutrition" page
2. System displays nutrition page and nutrition timeline to the user
3. User views their day, week, and month timeline
4. System displays each corresponding nutritional timeline depending on the day, week, month time frame
5. User selects specific nutritional item in timeline
6. System displays nutritional information about specific item
7. User clicks "Edit" button on specific item
8. System provides a form with the current item values which can be changed
9. User alters the desired values within the form which include the timestamp for consumption and nutritional information (calories, carbs, protein, fat, cholesterol, sodium)
10. System updates the form fields
11. User submits the form
12. System updates specific nutritional item

User repeats steps 3-9 until satisfied with the nutritional log.

Extensions:

a.a User exits page abruptly
    1. *Nutrition page and information is no longer displayed to the user*

a.b System encounters an unexpected error
    1. *System will exit nutritional page*
    2. *System will log the error internally*

2.a If the user wants to manually add an item to the nutrition log
    1. *User presses the "+" Add Item button*
    2. *User enters the name of the Item to add in a search box.*
    3. *System displays search results in a drop-down menu.*
    4. *a. User selects the item they want to add.*

b. If the food type is not registered in the food type database

i. User selects the "Add food type" option from the end of the drop-down menu

ii. extend <addFoodType>

5. *User enters the quantity they consumed.*

   a. The user can change the timestamp for consumption if it is not the current time.

6. *User selects create item*

7. a If the User wants to remove the existing item from the nutrition log

1. *User clicks on the 'Delete Item' button within the item details interface*
2. *System asks the user to confirm that they want to delete the item*
3. *User clicks 'Confirm'*
4. *System removes item from the nutrition log*

11. a User hits 'X' to exit the nutritional modification form

1. *System exits the form without saving the modifications*

ID UC 2.2 Manage Nutrition Goals
Scope Nutrition Log system
Level User goal
Stakeholders and Interests

- **User:** A person utilizing the Nutrition Log system within the FoodPants application to set and manage nutritional goals.
- **System maintainer:** Person responsible for application execution.

Precondition: User has undergone setup process. User is viewing the Nutrition page. User wants to manage their nutrition goals.
Postcondition: A nutrition goal has been added, edited, or removed.

Main success scenario:

1. User selects the goals button to retrieve nutritional goals menu
2. System displays nutritional goals menu
3. User selects the "+" button
4. System displays nutrition goal creation form
5. User picks the nutrient type, goal value, and goal type (minimum or maximum)
6. System updates form with modifications
7. User submits the new nutritional goal
8. System updates nutritional goals menu with the new goal

User repeats steps 3-8 until satisfied with the management of their nutrition goals

Extensions:

a.a User exits page abruptly

2. *Nutrition page and information is no longer displayed to the user*

a.b System encounters an unexpected error

3. *System will exit nutritional page*
4. *System will log the error internally*

User repeats this step until they have gotten all the way out of the specified process

3. a If the User wants to modify a pre-existing nutrition goal

1. *User clicks on the edit goal button within the nutritional goals menu*
2. *System displays form where current goal values can be changed*
3. *User alters the desired values within the form*
   a. User hits the 'Confirm' button
   b. User hits an 'X' which exits without saving changes
4. *System updates goal information*

3. b If the User wants to remove a pre-existing goal from the nutrition log

1. *User clicks on the edit goal button within the nutritional goals interface*
2. *System displays form with the current goal values*

3. *User clicks on the delete goal button within the edit goal interface*
4. *System asks the user to confirm that they want to delete the goal*
5. *User clicks 'Confirm'*
6. *System removes goal from the nutrition goals menu*

ID UC 2.3 Manage Nutrition Report
Scope Nutrition Log system
Level User goal
Stakeholders and Interests

- **User:** A person utilizing the Nutrition Log system within the FoodPants application to view, create, modify, or remove a nutrition report.
- **System maintainer:** Person responsible for application execution.

Precondition: User has undergone setup process. User is viewing the Nutrition page. User wants to manage nutrition reports.
Postcondition: Nutrition reports are updated with any user modifications.

Main success scenario:

1. User selects the nutrition reports button to navigate to the nutrition reports menu
2. System displays nutrition reports menu
3. User views nutritional reports via scroll along the page
4. System displays a list of all previous nutritional reports
5. User selects the "+" button
6. System displays nutrition report creation form
7. User enters the desired time range in which they would like the report
8. System updates form with desired time range
9. User submits the creation form
10. System updates nutritional reports menu with new nutritional report

User repeats steps 5-8 until satisfied with the management of their nutrition goals

Extensions:

a.a User exits page abruptly
   3. *Nutrition page and information is no longer displayed to the user*

a.b System encounters an unexpected error
   5. *System will exit nutritional page*
   6. *System will log the error internally*

4. a Nutritional reports list is empty
   1. *System displays a message saying no nutritional reports exist yet*

5. a If the User wants to modify a pre-existing nutrition report
   1. *User clicks on the edit report button within a particular report*
   2. *System displays form with the current report specifications which can be changed*
   3. *User alters the desired values within the form*
      a. User hits the 'Confirm' button
      b. User hits an 'X' which exits without saving changes
   4. *System updates report information*

5. b If the User wants to remove a pre-existing report from the nutrition report menu

1. *User clicks on the edit report button within the specified report*
2. *System displays form with current report specifications*
3. *User clicks on the delete report button within the edit report interface*
4. *System asks the user to confirm that they want to delete the report*
5. *User clicks 'Confirm'*
6. *System removes report from the nutrition reports menu*

ID UC 3.1
Scope Recipe creation system.
Level User goal
Stakeholders and interests

- **Project Leader:** Person responsible for overseeing system goals.
- **Project Member:** Person responsible for implementing and maintaining the system.
- **System:** Person responsible for recipe creation execution.

Precondition: The user can successfully execute the application.
Postcondition: Recipe successfully created.

Main success scenario:

1. User clicks on “Recipes” from sidebar menu
2. System displays recipe page with stored recipes
3. User creates a recipe manually
4. System provides a recipe creation form
5. User enters recipe information into form
6. System updates form fields with user input
7. User submits recipe form
8. System verifies form has required information
9. New recipe is added into recipes list

Extensions:

a.a User exits form abruptly
    1. *recipe creation will be canceled.*

a.b User exits page abruptly
    1. *Recipe creation will be canceled*

a.c User exits application abruptly
    1. *Recipe creation will be canceled*

b.* System encounters an unexpected error
    1. *Recipe creation will be canceled*
    2. *Page will be restarted*

2.a If there are no recipes yet created
    1. *System will only displayed recommended recipes*

5.a User does not enter a recipe name into form
    1. *System will ask user for recipe name*

5.b If user does not enter at least one ingredient into form
    1. *System will ask user for at least one ingredient*

5.c If user does not enter number of servings into form
    1. *System will ask user for number of servings*

ID UC 3.2
Scope View recipe system
Level User goal
Stakeholders and interests

- **Project Leader:** Person responsible for overseeing system goals.
- **Project Member:** Person responsible for implementing and maintaining the system.
- **System:** Person responsible for recipe viewing execution.

Precondition: User can successfully execute the application. User has created at least one recipe.
Postcondition: Recipes successfully viewed.

Main success scenario:

1. User selects “Recipes” from sidebar menu
2. System displays list of recipes to user
3. User searches for specific recipe by name
4. System displays recipe search results
5. User clicks on a specific recipe
6. System displays information from recipe

Extensions:

a.a User exits page abruptly
    1. *Recipes are no longer displayed to the user*

a.b System encounters an unexpected error
    1. *System will exit recipe page*
    2. *System will store the error internally*

2.a User has no recipes
    1. *System will only display recommended recipes in list*

3.a User searches for recipe based on ingredients
    1. *System displays recipes based on matching ingredients*

4.a If no matching recipes from search
    1. *System will display a blank page to the user*

ID UC 3.3
Scope Recipe to shopping list system
Level User goal
Stakeholders and interests

- **Project Leader:** Person responsible for overseeing system goals.
- **Project Member:** Person responsible for implementing and maintaining the system.
- **System:** Person responsible for recipe viewing execution.

Precondition: A user can successfully view recipes list.
Postcondition: Recipe successfully added to shopping list

Main success scenario:

1. User clicks on “Recipes” from sidebar menu
2. System displays stored recipes to user
3. User views recipes
4. System displays list of recipes
5. User selects specific recipe
6. System displays specific recipe information
7. User adds recipe to shopping list
8. System adds recipe ingredients to shopping list

Extensions:

a.a User exits page abruptly
    4. *Recipes are no longer displayed to the user*

a.b System encounters an unexpected error
    7. *System will exit recipe page*
    8. *System will store the error internally*

4.a If user has no recipes
    1. *System will only display recommended recipes in recipe list*
    2. *System will ask user to create a recipe or choose from recommended*

7.a Recipe ingredients already exist in shopping list
    1. *System will add the ingredients again*

ID UC 3.4 Modify Recipe
Scope Recipe System
Level User goal
Stakeholders and Interests

- **User:** A person utilizing the Recipe System within the FoodPants application to modify existing recipes.
- **System maintainer:** Person responsible for application execution.

Precondition: Recipes exist within the Recipe System so there is something to modify. User wants to modify a recipe.
Postcondition: A recipe has been edited.

Main success scenario:

1. User navigates to the recipe page.
2. System displays all recipes.
3. User selects the "Edit Recipe" button.
4. System displays the recipe specifications.
5. User edits food type and amount fields and/or the recipe instruction field, and selects the "Save Changes" button to save edited fields.
6. System confirms that changes were saved, and replaces the old recipe with the user-edited recipe.
7. User is returned to the recipe view page with the newly modified recipe listed.

Extensions:

3.a User has no recipes
    1. *User selected the "+" button.*
    2. *Recipe creation page is displayed to the user.*

5.a The user wants to add a new food type
    1. *User selects the "+" button in the food type field.*
    2. *New food type page is displayed*
    3. *User fills required fields.*
    4. *User selects the "Create Item" button.*
    5. *Recipe modification page is displayed with newly created food type added to the food type field.*

5.a The user wants to exit the recipe modification page without saving modifications
    1. *The user selects the "Discard Changes" button.*
    2. *The user is returned to the recipe view page.*

5.b The user wants to delete the recipe.
    1. *The user selects the "Delete Recipe" button.*

2. *The user is returned to the recipe view page and the recipe is no longer listed.*

ID UC 3.5 Recommend Recipes
Scope Recipe System and Digital Pantry
Level User goal
Stakeholders and Interests

- **User:** A person utilizing the Recipe System within the FoodPants application to view recommended recipes.
- **System maintainer:** Person responsible for application execution.

Precondition: Recipes exist within the Recipe System and the user has items in their pantry. User wants to view recommended recipes based on what they already have in their pantry.
Postcondition: Recommended recipes are displayed to the user for further action to be taken.

Main success scenario:

1. User navigates to the recipe page.
2. System displays all recipes.
3. User selects the "View Recommendations" button.
4. System displays recommended recipes.
5. User selects a recipe to view

Extensions:

3.a User exits the recipe viewing page
    1. *User selects the "Back" button.*
    2. *Home page is displayed to the user.*

5.a The user wants to stop viewing recommended recipes
    1. *User selects the "View Recommendations" button, removing its highlight*
    2. *Default recipe page is displayed*

ID UC 3.6 Make Recipe
Scope Recipe system
Level User goal
Stakeholders and Interests

- **User:** Person who has stored recipes and items within the FoodPants digital pantry, wants to use stored items to make a recipe and log the results to their nutrition log.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: The user has created the recipe within the recipe system, and has decided how much of the recipe they will eat (they can perform this step after cooking if necessary)
Postcondition: Used items will have been consumed from the user's pantry, and the recipe and nutritional info will be stored in the user's nutrition log.

Main success scenario:

1. User selects to make the recipe.
2. System removes items from the digital pantry that were used to make the recipe.
3. User is prompted for how much of the recipe they consumed.
4. System adds the amount of the recipe consumed is added to the nutrition log.
5. include (manageNutritionLog)
6. User is prompted to enter how much of the recipe is leftover.
7. System stores leftovers as part of the User's digital pantry.
8. include (managePantry)
9. User is returned to the recipe view menu.

Extensions:

a. * The desired recipe does not exist
    1. *The System will not allow the user to cook the recipe.*
    2. *The user can add the recipe to the system.*

4. a The user does not have all the required items to cook the recipe.
    1. *The system prompts the user if they want to cook the recipe anyways.*
    2. *a. User confirms to cook the recipe anyways.*
        i. *User selects to consume only existing items from the pantry.*
        ii. *User selects to consume no items from the pantry.*
    b. User requests desired items to be added to the shopping list.
        i. *System adds items to the shopping list (include addRecipeItemsToShoppingList).*
        ii. *User is redirected to the shopping list menu, preparing the recipe is aborted.*

6. a The user has not set up serving sizes for this recipe

    1. *The system prompts the user to dictate how many servings the recipe makes*
    2. *The user enters how many servings were consumed.*

9. a The user has no food remaining or does not wish to record leftovers
    1. *The user can press an option to skip this step.*

Special Requirements

- Users can utilize this function even if they are missing ingredients.
- This will interface nicely with the shopping list feature so that users can automatically fill in missing items.

ID UC 4.1 Manage Shopping List
Scope Shopping List system
Level User goal
Stakeholders and interests

- **User:** Person interested in managing a shopping list. Wants the ability to modify their shopping list. Wants the ability to add items to their shopping list based on a recipe. Wants an easily visible display of their shopping list.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: Food types to purchase are registered in the food database. User wants to manage a shopping list.
Postcondition: Shopping list modifications are saved.

Main success scenario:

1. User navigates to the "Shopping List" page to view the list.
2. System displays the list of shopping items.
3. User opens the "Add Item" form and enters the name of the food to purchase.
4. System displays food search results.

User repeats steps 3-4 until the food with the given name is found

5. User submits the form with the selected food to purchase.
6. System adds the item to the shopping list and displays the updated list.

User repeats steps 3-6 until satisfied with the shopping list

Extensions:

a.* Anytime the user decides to stop managing the shopping list
    1. *User exits the active window*

3.a User wants to add items to their list from a recipe instead of manually
    1. *User navigates to the "Add Items From Recipe" option*
    2. *extend <addIngredientsToCart>*
    3. *System adds the recipe items to the shopping list*

3.b User presses the "Delete Item" button to delete an existing item
    1. *System prompts the user for confirmation*
    2. *a. User presses the "Yes" button*
        i. *System deletes the item*
    b. User presses the "No" button

3.c User wants to change the quantity/amount to purchase for an existing item
    1. *User selects the quantity/amount text box*
    2. *User enters a new quantity/amount*

4.a Food is not registered in the food database

1. *User selects the "Add food" option from the end of the drop-down menu*
2. *extend <addFood>*

6.a User wants to export the shopping list

1. *extend <exportShoppingList>*

ID UC 4.2 Export Shopping List
Scope Shopping List system
Level User goal
Stakeholders and Interests

- **User:** Person that is interested in creating a grocery list of their items and exporting it from the application.
- **System maintainer:** Person responsible for application execution.

Precondition: Shopping list is created, desired items have been added
Postcondition: A shopping list is added to shopping lists and can be viewed from that menu.

Main success scenario:

1. User is viewing their shopping list and selects to export the shopping list.
2. System prompts the user on how they would like to export the list.
3. User selects that they would like to print the list.
4. System displays list as a print preview.
5. User prints the list.
6. System saves the list for view in the list menu.

Extensions:

a. * If the list is empty
    1. *The system will deny the user request, telling them to add items*

3. a If the user changes their mind,
    1. *The system will return the user to the main shopping list window.*

5. a The user wants to save the list as a PDF.
    1. *The system will export the list to a PDF.*
    2. *The user can download the file.*

5. b The user wants to save the list as a text file.
    1. *The system will export the list to a text file.*
    2. *The user can download the file.*

5. c The user wants to save the list as a csv file.
    3. *The system will export the list to a csv file.*
    4. *The user can download the file.*

5. d User does not want to export the list
    1. *The system will proceed to save the list for view in the list section.*

ID UC 4.3 Mark Purchased Items
Scope Shopping List system
Level User goal
Stakeholders and interests

- **User:** Person interested in shopping list. Wants the ability to cross out purchased items from their shopping list (one by one or all items at once).
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: User had previously added items to the shopping list. User purchased all items from their shopping list or a subset of the items from their shopping list. User wants to mark items from their shopping list as purchased.
Postcondition: Items marked as purchased are removed from the shopping list. Items marked as purchased are added to the pantry.

Main success scenario:

1. User navigates to the “Shopping List” page to view the list.
2. System displays all items in the shopping list.
3. User marks a subset of the items as purchased.
4. System moves the selected items from the shopping list to the pantry.

User repeats steps 3-4 until the shopping list and pantry accurately reflect the user’s purchases

Extensions:

a.* Anytime the user decides to stop managing the shopping list
    1. *User exits the active window*

3.a. The list is empty
    1. *The system will deny the user request, telling them to add items*

ID UC 5.1 Startup
Scope Startup
Level Summary
Stakeholders and Interests

- **New User:** New person utilizing the FoodPants application for the first time, wants to become familiarized with the application.
- **System maintainer:** Person responsible for application execution.

Precondition: FoodPants is installed as an executable on the user's machine and running.
Postcondition: New User has a profile created, and has been familiarized with the system.

Main success scenario:

1. The system greets the new user
2. The system prompts the new user for their details.
3. The user enters details such as name, gender, height, and weight for calorie info.
4. The system saves the user data and configures the new profile.
5. The user is shown the digital pantry.
6. The user is prompted to enter items they may already have in their pantry.
7. include (addFoodToPantry)

Continue step 7 until the user decides they are satisfied with their pantry.

8. User is shown the recipe menu and suggested a recipe to add
9. include (getNewReccomendedRecipies)
10. User is shown the shopping list and prompted to add an item of their choice
11. include (manageShoppingList)
12. System thanks the user for using the walkthrough, displays documentation for further questions
13. User finishes tutorial and redirects to the home window.

Extensions:

a. * The user does not wish to continue the tutorial
    1. *The user can click a button to exit the tutorial*
    2. *The system will resume processes from the home menu*

3. a If the user does not wish to enter gender, height, or weight.
    1. *These fields are optional, some nutrition features will be disabled.*

6. b User does not wish to set up the digital pantry
    1. *The user can press an option to skip this step for now.*

8. a The user does not want to add the suggested recipe
    1. *The user can press an option to skip this step.*

10. a The user does not want to add an item to their list
    1. *The user can press an option to skip this step.*
13. a The user does not want to view the documentation
    1. *The user can deny this option to finish the tutorial.*

Special Requirements

- Provide startup as an embedded walkthrough within the application.

ID UC 6.1 Manage Food Types
Scope Food system
Level User goal
Stakeholders and interests

- **User:** Person interested in managing the food types. Wants to customize their available food types for tracking within their pantry, shopping list, recipes, and nutrition log.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: Basic foods are predefined (but still modifiable). User wants to manage their foods.
Postcondition: Food modifications are saved.

Main success scenario:

1. User navigates to the "Foods" page to view the foods.
2. System displays the defined foods.
3. User opens, fills out, and submits the "Add Food" form.
4. extend <addFood>
5. System adds the food to the list of foods and displays the updated list.

User repeats steps 3-5 until satisfied with the list of foods

Extensions:

a.* Anytime the user decides to stop managing the list of foods
    1. *User closes the active window*

3.a User presses the "Delete Food" button to delete an existing food type
    1. *System prompts the user for confirmation*
    2. *a. User presses the "Yes" button*
        i. *System deletes items in pantry of the given food type*
        ii. *System deletes items in shopping list of the given food type*
        iii. *System removes the given food type from the list of food types displayed*

    b. User presses the "No" button

3.b User wants to edit an existing food details
    1. *User opens, fills out, and submits the "Edit Food" form*
    2. *System updates the food in the list of foods and displays the updated list*

ID UC 6.2 Create New Food Type
Scope Food system
Level User goal
Stakeholders and Interests

- **User:** Person interested in managing the foods. Wants to customize their available foods for tracking within their pantry, shopping list, recipes, and nutrition log.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: User wishes to create a new food or manage existing foods.
Postcondition: New food is created and is selectable by the user.

Main success scenario:

1. User navigates to the "Food Types" page to view the food types.
2. System displays food types.
3. User navigates to the "Modify Food Types" option and presses the "Add Food Type" button.
4. System displays new food type specifications.
5. User enters required information to complete a new food type, and selects the "Create Food Type" button.
6. System confirms new food type was successfully created.
7. The Food Types page is displayed to the user.

User repeats steps 4-5 until satisfied with the list of food types

8. User presses the "Save" button.

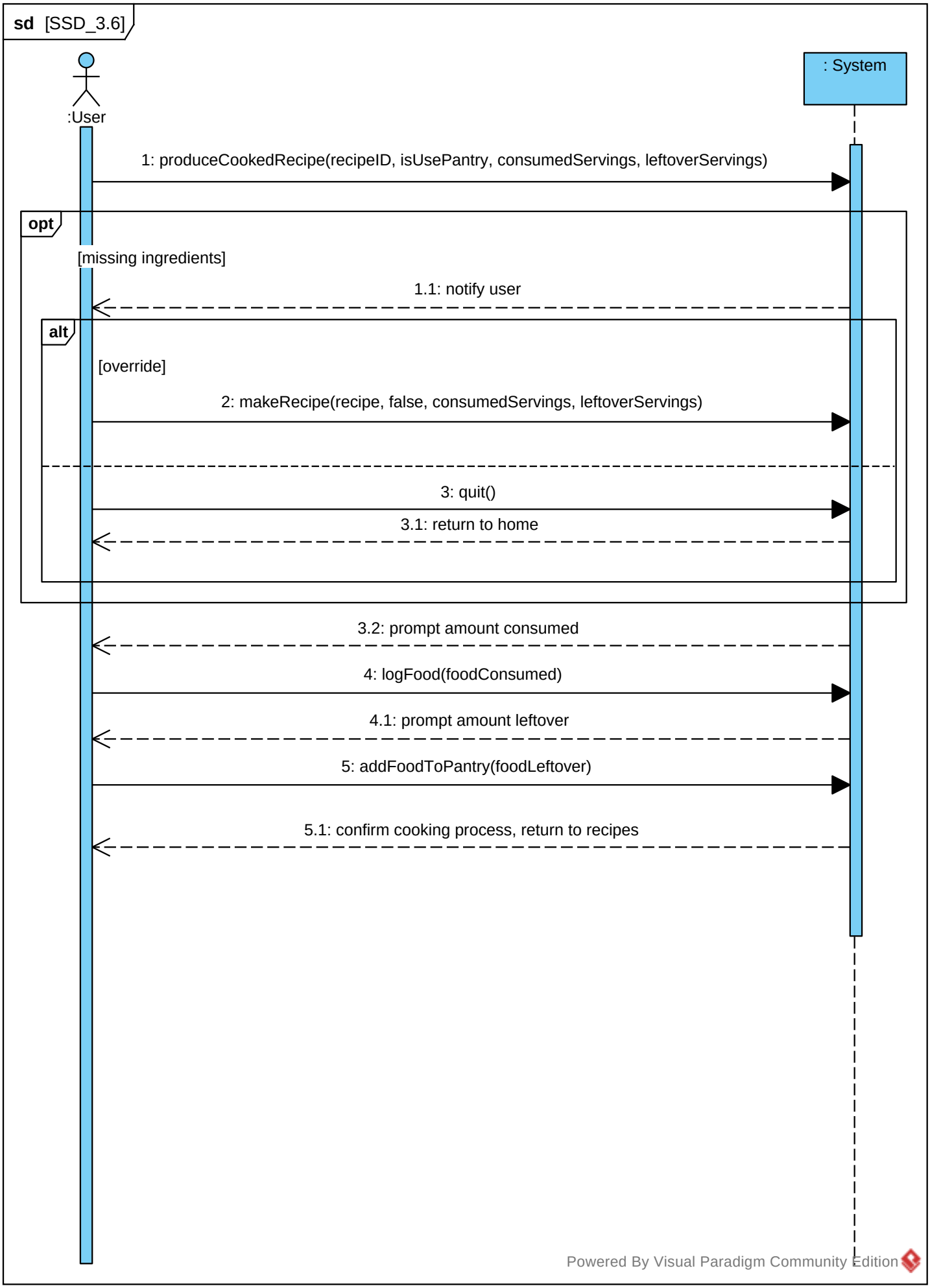
Extensions:

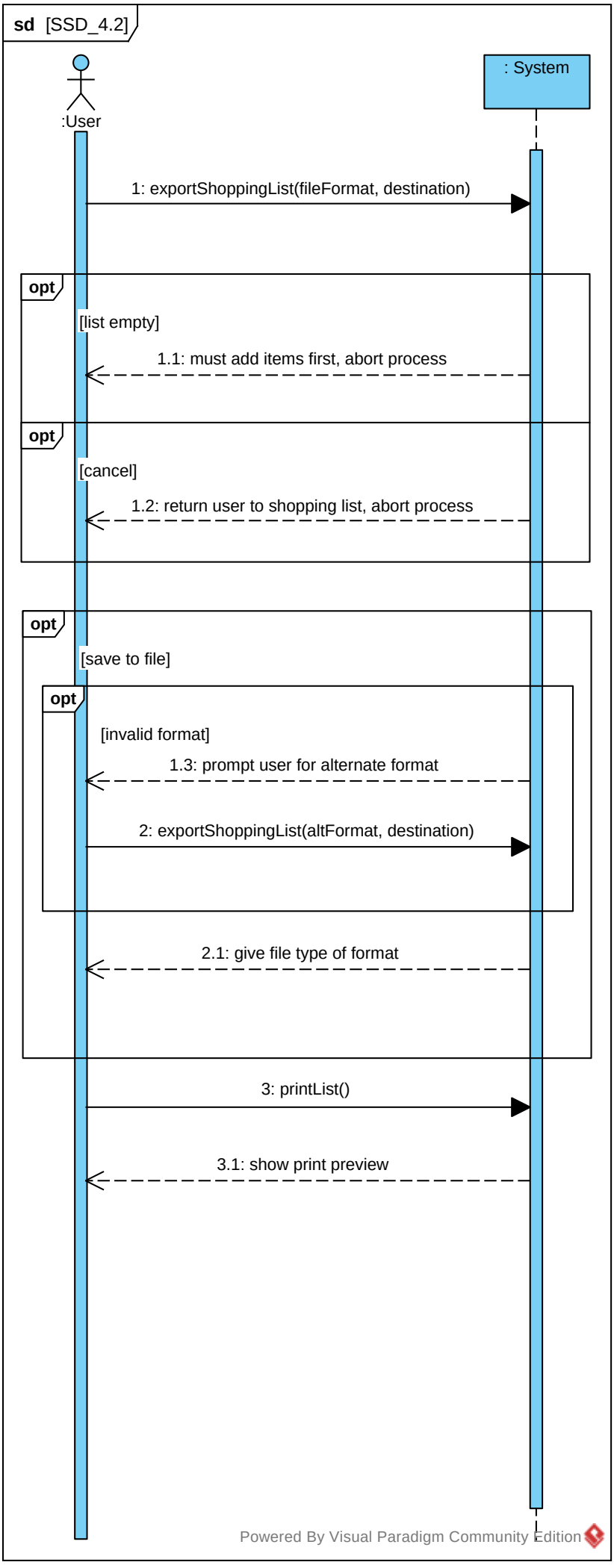
3a.* The user no longer wants to manage food types.
    1. *User presses the "Back" button*
        a. *User has unsaved changes*
            i. *System prompts user to save their changes*
            ii. *User presses "OK"*
            iii. *System redirects user to the "Modify Shopping List" page*

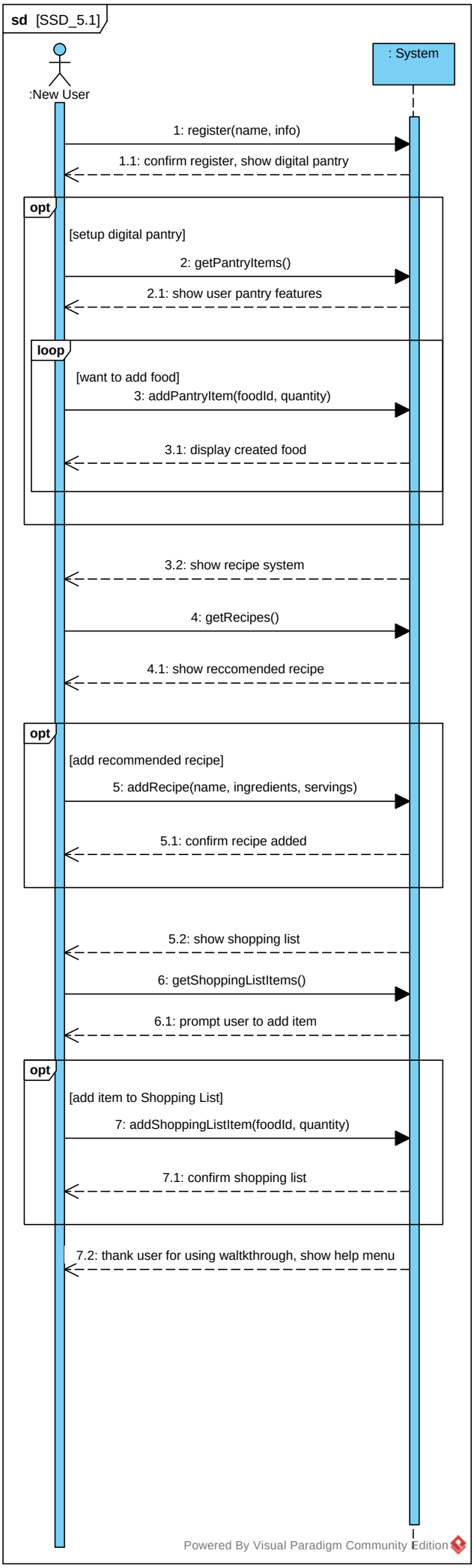
5.a User enters an existing name in the "Food Name" text box
    1. *The system prompts the user to change the name and prevents the user from exiting the text box.*
    2. *Return to step 3 of the main success scenario.*

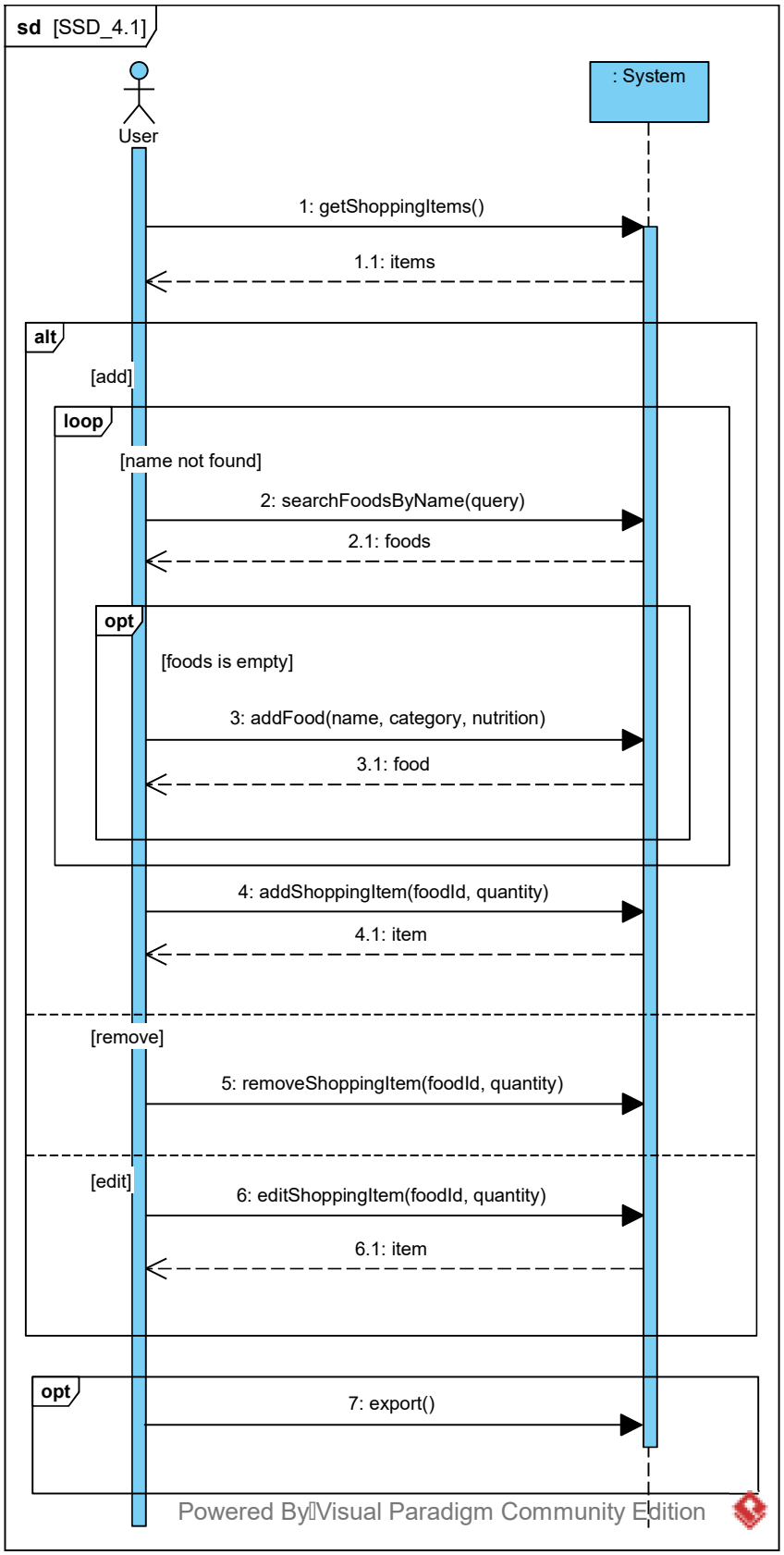
5.b User leaves a required text box empty
    1. *User prompts user to fill out required text fields.*
    2. *Return to step 7 of main success scenario.*

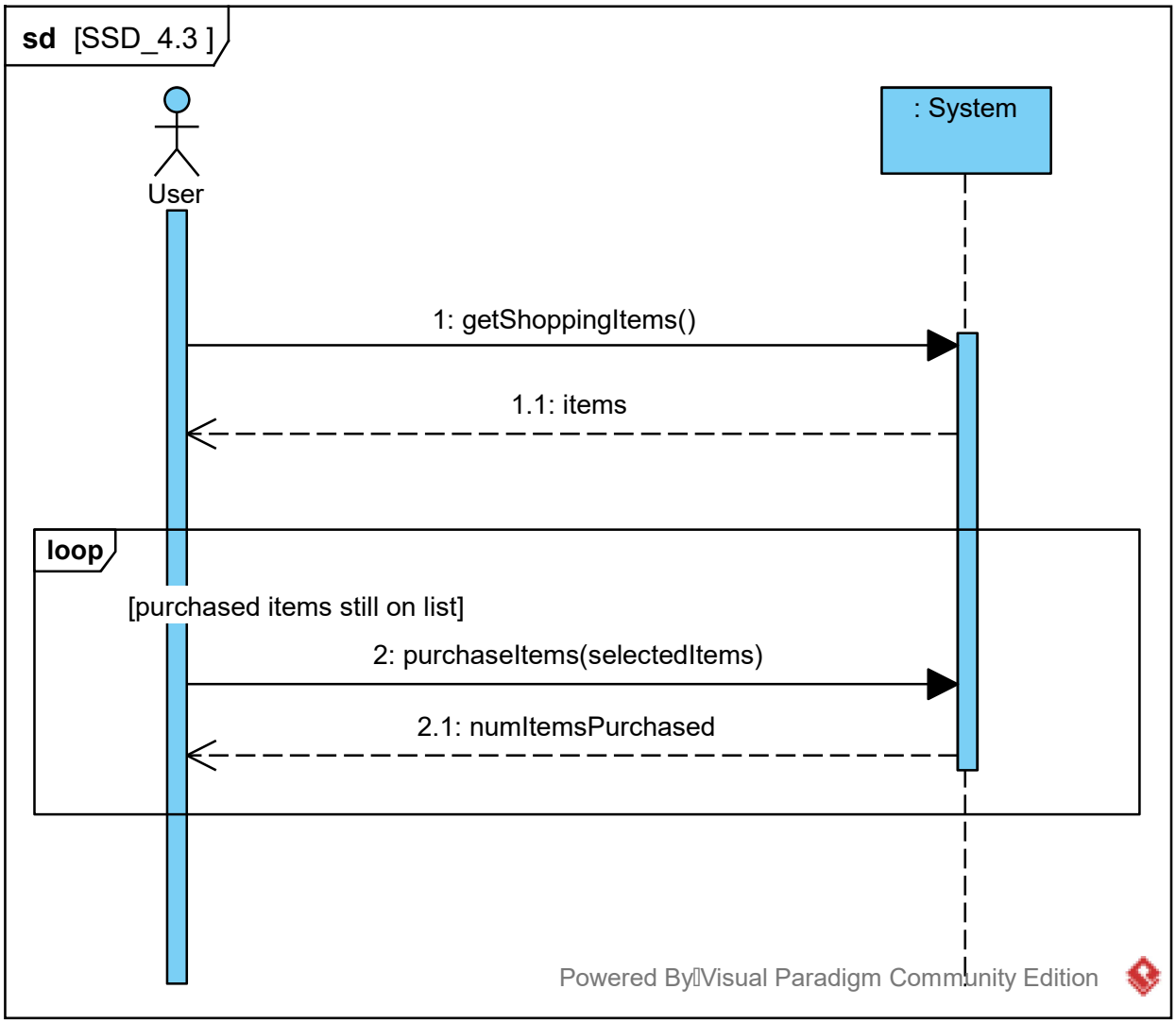
Use Case Name	Use Case ID	REQ 1	REQ 2	REQ 3	REQ 4	REQ 5	REQ 6	REQ 7	REQ 8	REQ 9	REQ 10
Manage Pantry	1.1	X						X	X		X
Search Pantry	1.2	X							X		X
Consume Pantry Item	1.3	X							X		X
Manage Nutrition Log	2.1		X					X	X		X
Nutrition Goals	2.2		X						X	X	X
Nutritional Report	2.3		X						X		X
Create Recipe	3.1			X			X	X	X	X	
View Recipes	3.2			X				X	X		
Add Recipe Items to Shopping List	3.3			X			X		X		X
Modify Recipe	3.4			X					X	X	
Recommend Recipe	3.5			X				X	X		X
Cook Recipe	3.6	X		X					X		X
Manage Shopping List	4.1				X		X	X	X		X
Export Shopping List	4.2				X				X		X
Mark Purchased Items	4.3				X				X		X
Startup Tutorial	5.1	X	X	X	X	X	X	X	X	X	X
Manage Food Types	6.1						X	X	X	X	
Create New Food Type	6.2						X			X	

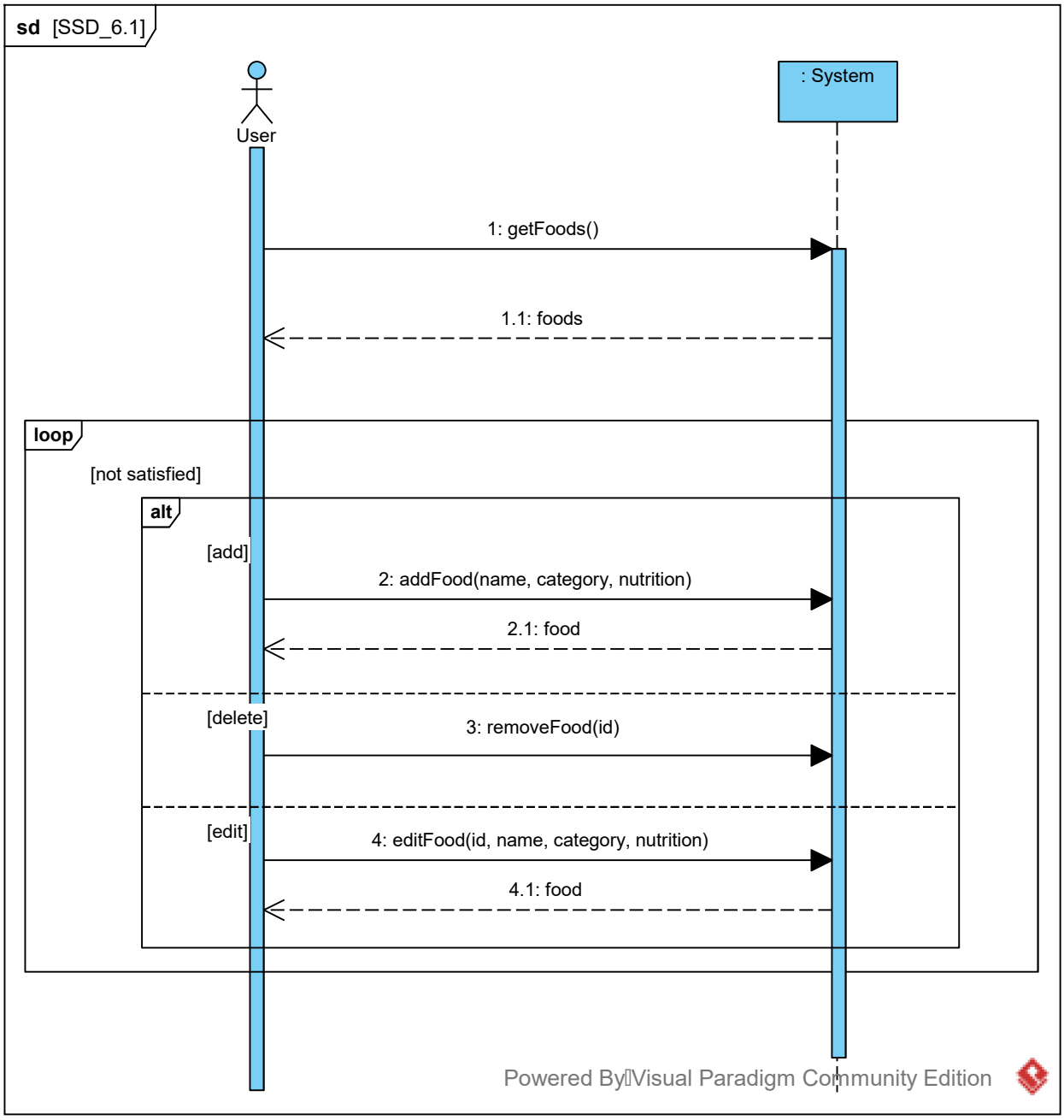

sd [SSD_3.6]
:User
: System
1: produceCookedRecipe(recipeID, isUsePantry, consumedServings, leftoverServings)
opt
[missing ingredients]
1.1: notify user
alt
[override]
2: makeRecipe(recipe, false, consumedServings, leftoverServings)
3: quit()
3.1: return to home
3.2: prompt amount consumed
4: logFood(foodConsumed)
4.1: prompt amount leftover
5: addFoodToPantry(foodLeftover)
5.1: confirm cooking process, return to recipes
Powered By Visual Paradigm Community Edition

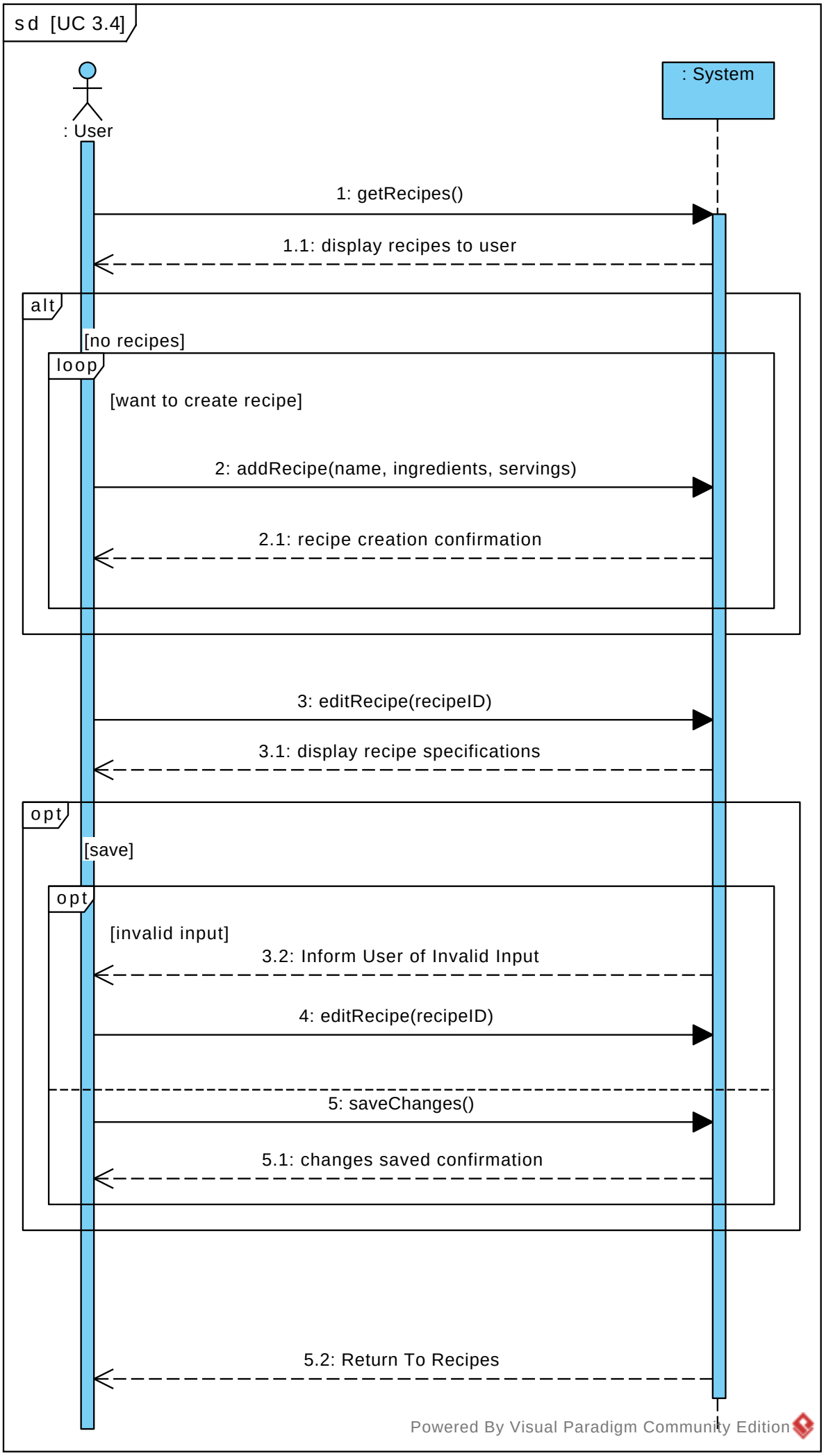

sd [SSD_4.2]
:User
: System
1: exportShoppingList(fileFormat, destination)
opt
[list empty]
1.1: must add items first, abort process
opt
[cancel]
1.2: return user to shopping list, abort process
opt
[save to file]
opt
[invalid format]
1.3: prompt user for alternate format
2: exportShoppingList(altFormat, destination)
2.1: give file type of format
3: printList()
3.1: show print preview
Powered By Visual Paradigm Community Edition

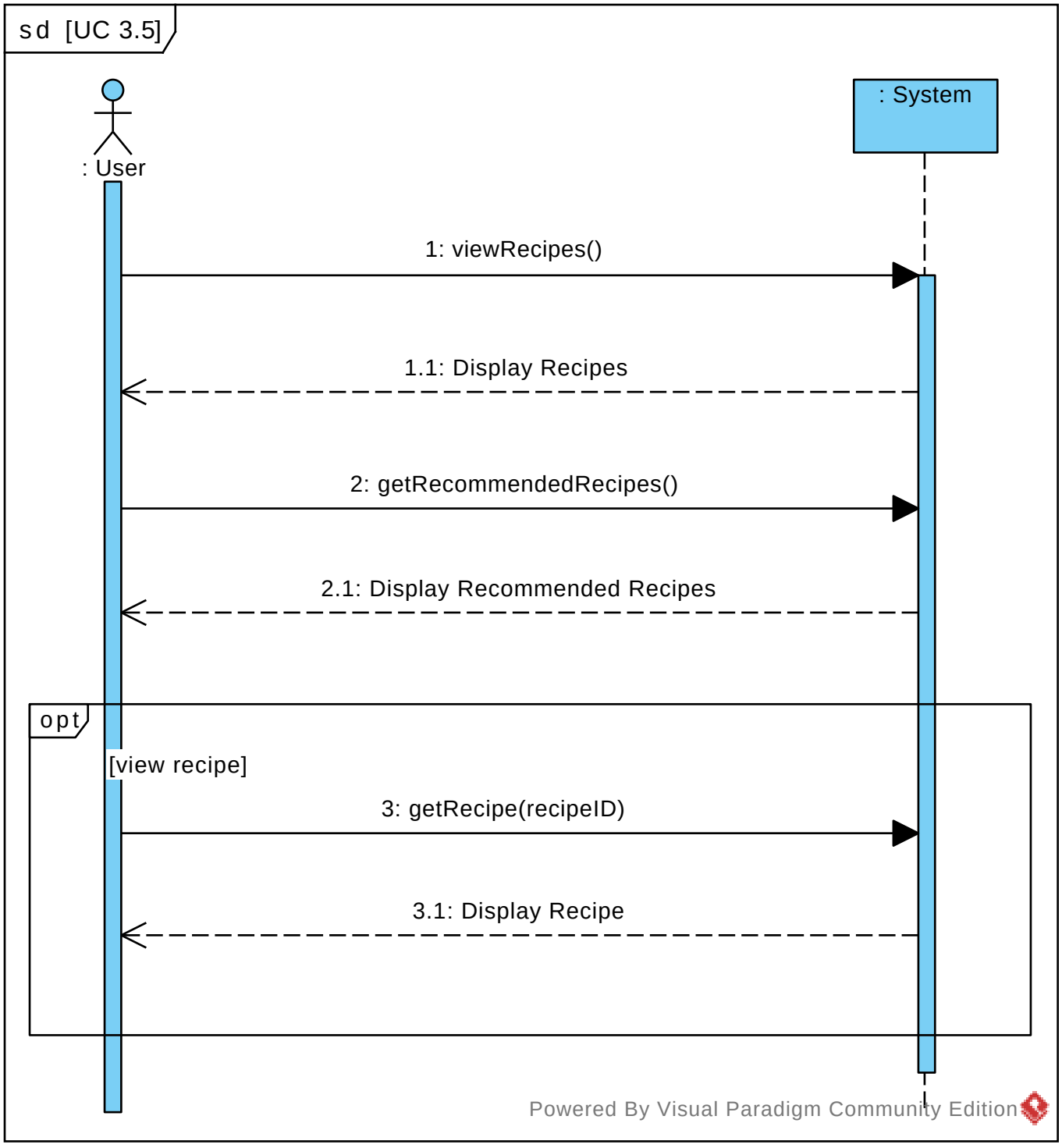

sd [SSD_5.1]
:New User
: System
1: register(name, info)
1.1: confirm register, show digital pantry
opt
[setup digital pantry]
2: getPantryItems()
2.1: show user pantry features
loop
[want to add food]
3: addPantryItem(foodId, quantity)
3.1: display created food
3.2: show recipe system
4: getRecipes()
4.1: show reccomended recipe
opt
[add recommended recipe]
5: addRecipe(name, ingredients, servings)
5.1: confirm recipe added
5.2: show shopping list
6: getShoppingListItems()
6.1: prompt user to add item
opt
[add item to Shopping List]
7: addShoppingListItem(foodId, quantity)
7.1: confirm shopping list
7.2: thank user for using waltkthrough, show help menu
Powered By Visual Paradigm Community Edition

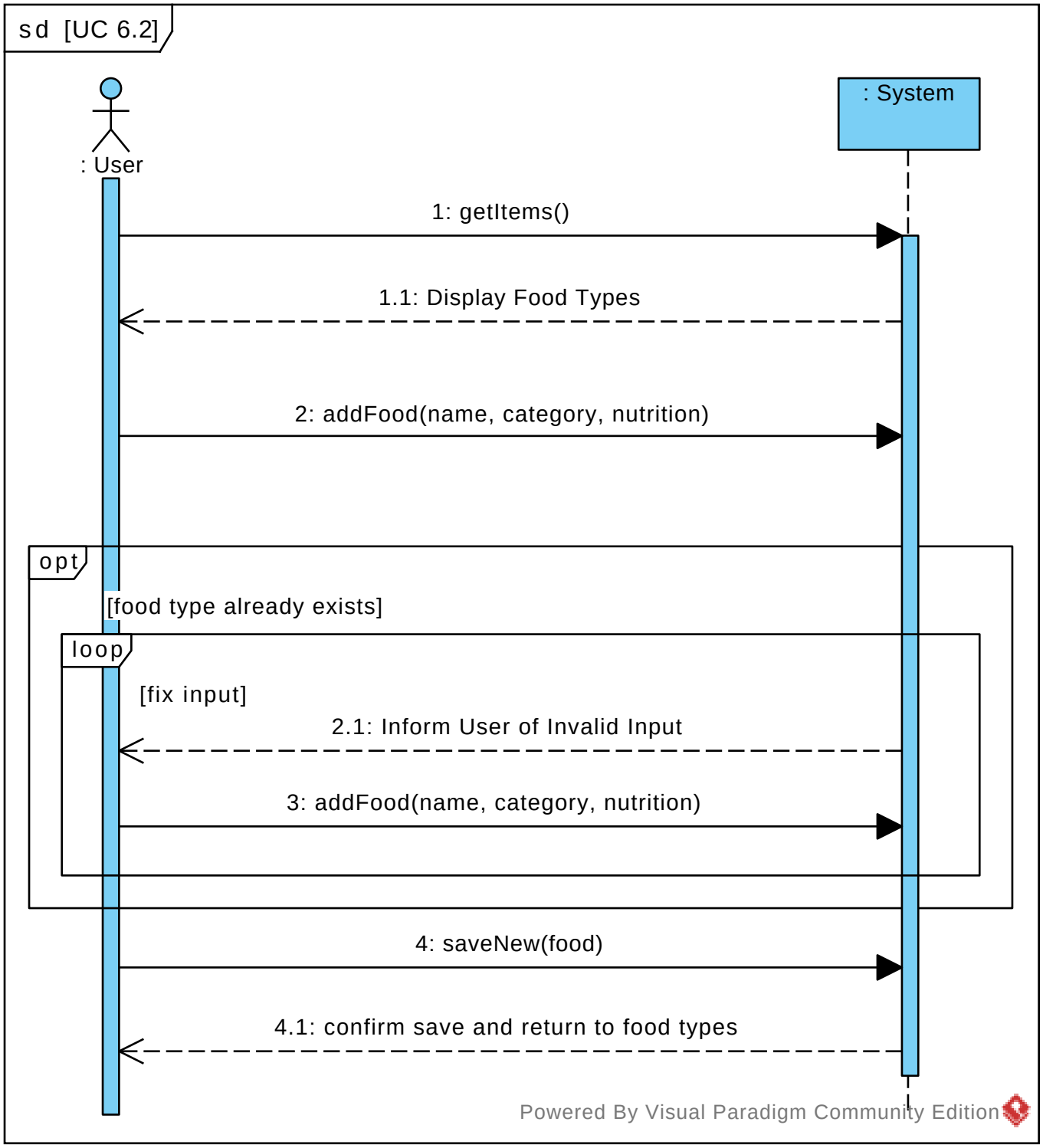

sd [SSD_4.1]
User
: System
1: getShoppingItems()
1.1: items
alt
[add]
loop
[name not found]
2: searchFoodsByName(query)
2.1: foods
opt
[foods is empty]
3: addFood(name, category, nutrition)
3.1: food
4: addShoppingItem(foodId, quantity)
4.1: item
[remove]
5: removeShoppingItem(foodId, quantity)
[edit]
6: editShoppingItem(foodId, quantity)
6.1: item
opt
7: export()
Powered By Visual Paradigm Community Edition

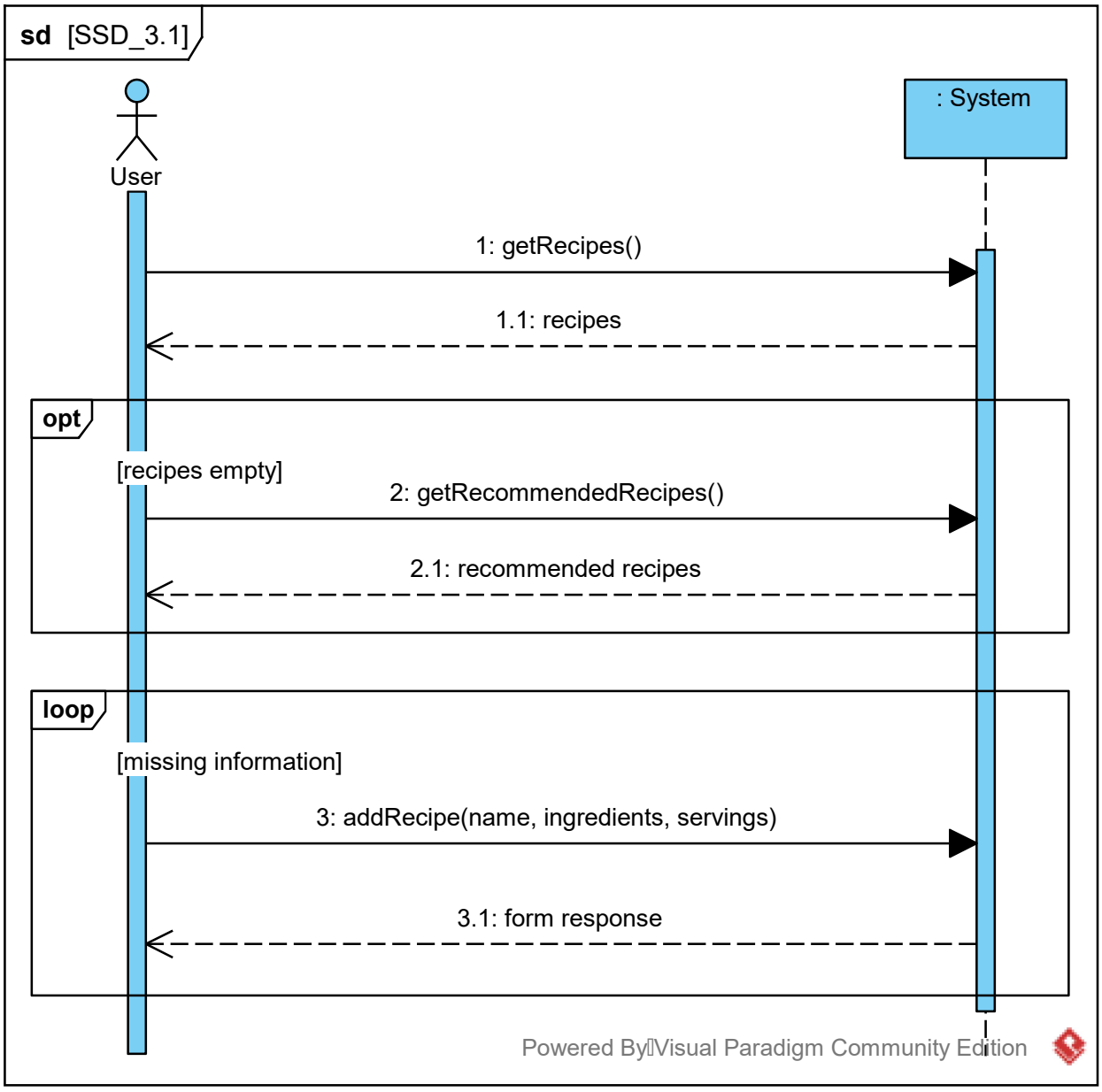

sd [SSD_4.3]
User
: System
1: getShoppingItems()
1.1: items
loop
[purchased items still on list]
2: purchaseItems(selectedItems)
2.1: numItemsPurchased
Powered By Visual Paradigm Community Edition

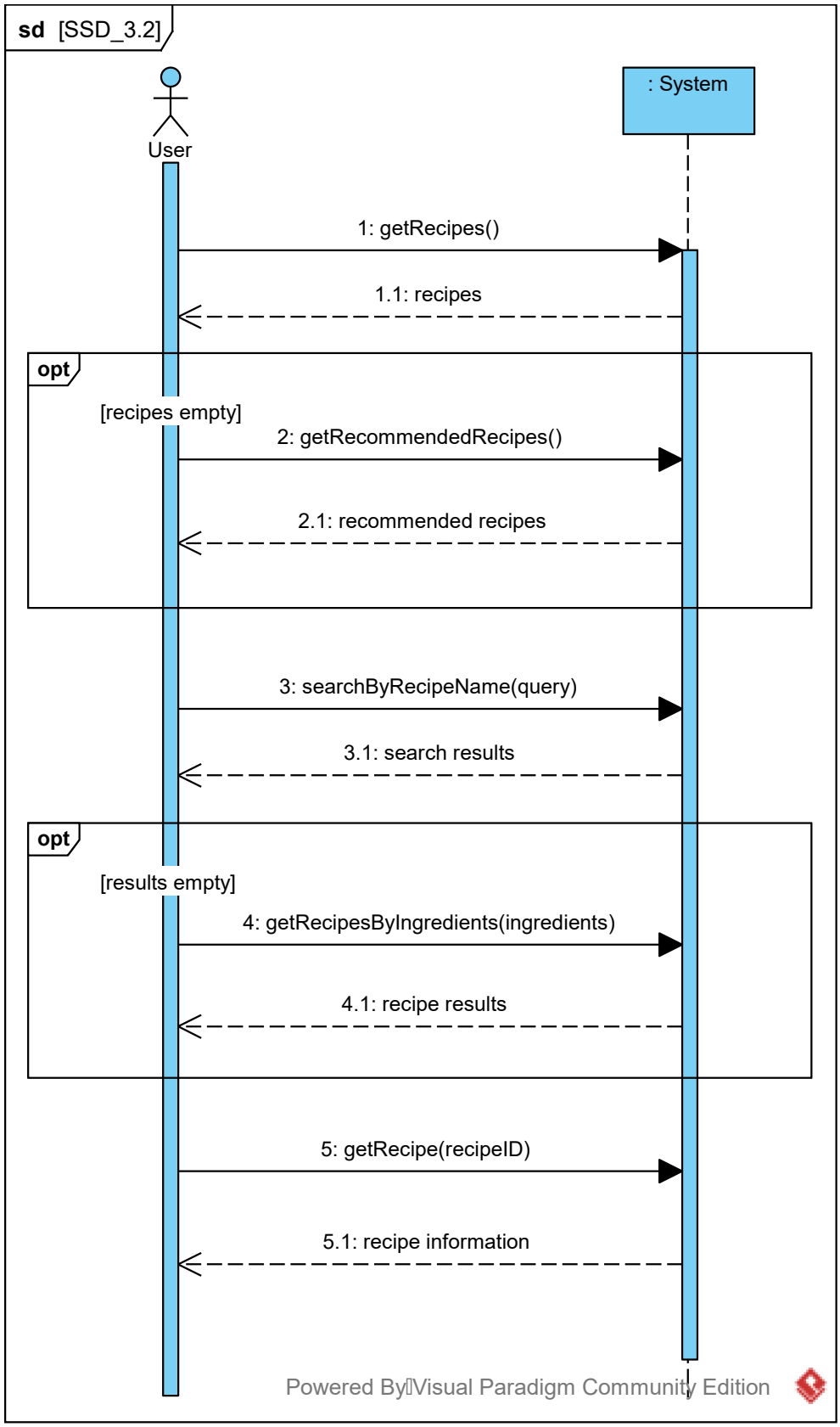

sd [SSD_6.1]
User
: System
1: getFoods()
1.1: foods
loop
[not satisfied]
alt
[add]
2: addFood(name, category, nutrition)
2.1: food
[delete]
3: removeFood(id)
[edit]
4: editFood(id, name, category, nutrition)
4.1: food
Powered By Visual Paradigm Community Edition

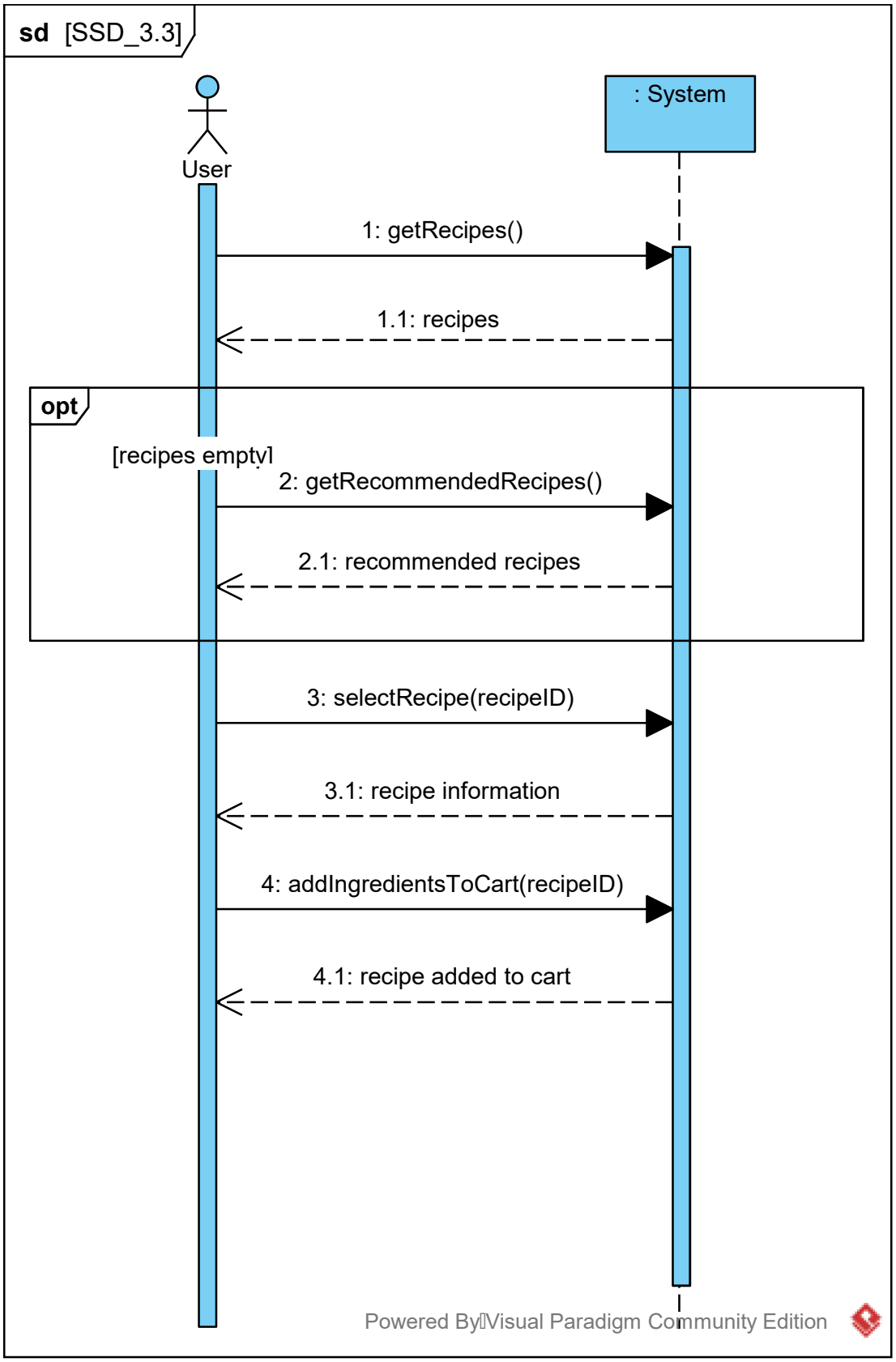

sd [UC 3.4]
: User
: System
1: getRecipes()
1.1: display recipes to user
alt
[no recipes]
loop
[want to create recipe]
2: addRecipe(name, ingredients, servings)
2.1: recipe creation confirmation
3: editRecipe(recipeID)
3.1: display recipe specifications
opt
[save]
opt
[invalid input]
3.2: Inform User of Invalid Input
4: editRecipe(recipeID)
5: saveChanges()
5.1: changes saved confirmation
5.2: Return To Recipes
Powered By Visual Paradigm Community Edition

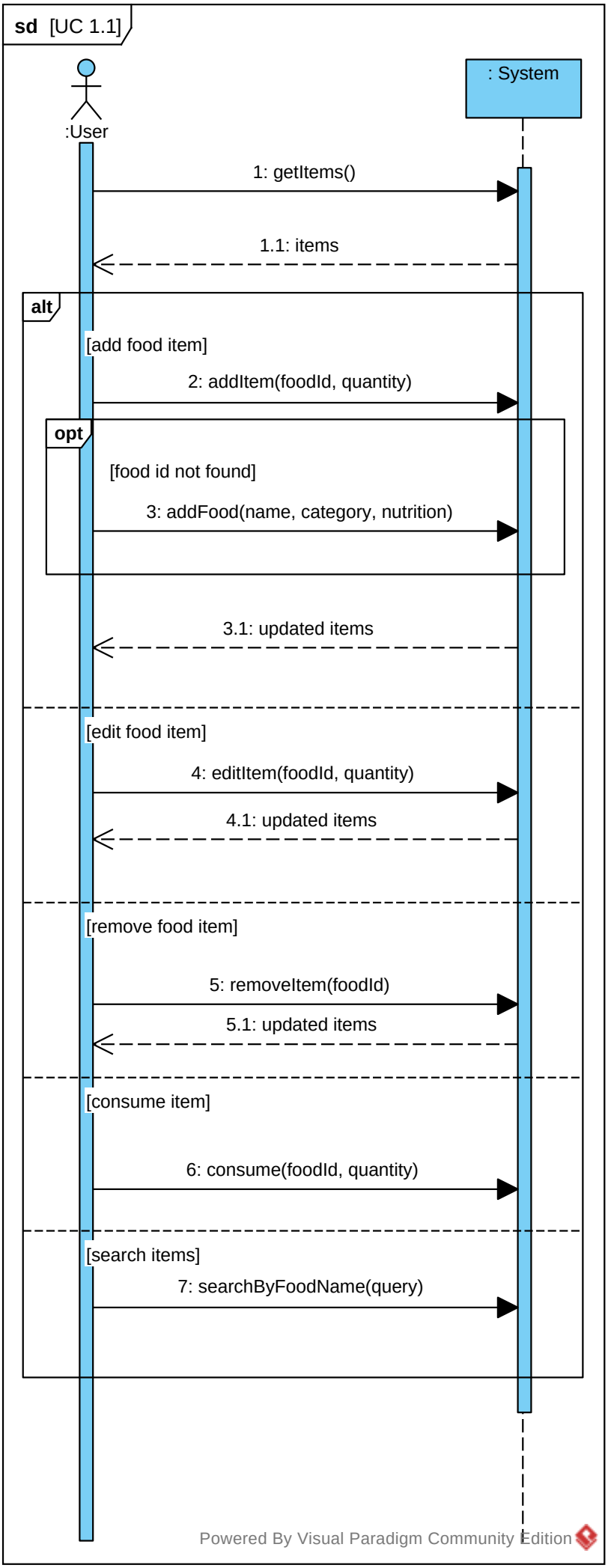

sd [UC 3.5]
: User
: System
1: viewRecipes()
1.1: Display Recipes
2: getRecommendedRecipes()
2.1: Display Recommended Recipes
opt
[view recipe]
3: getRecipe(recipeID)
3.1: Display Recipe
Powered By Visual Paradigm Community Edition

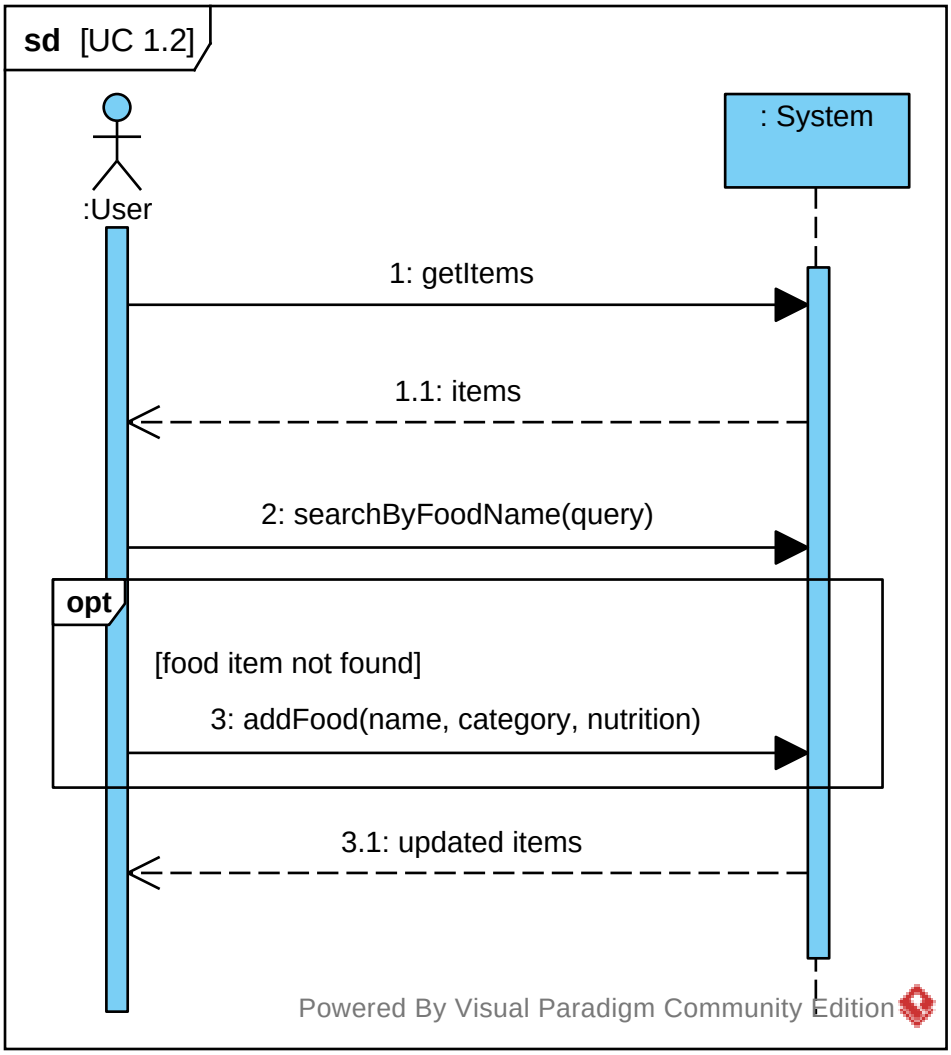

sd [UC 6.2]
: User
: System
1: getItems()
1.1: Display Food Types
2: addFood(name, category, nutrition)
opt
[food type already exists]
loop
[fix input]
2.1: Inform User of Invalid Input
3: addFood(name, category, nutrition)
4: saveNew(food)
4.1: confirm save and return to food types
Powered By Visual Paradigm Community Edition

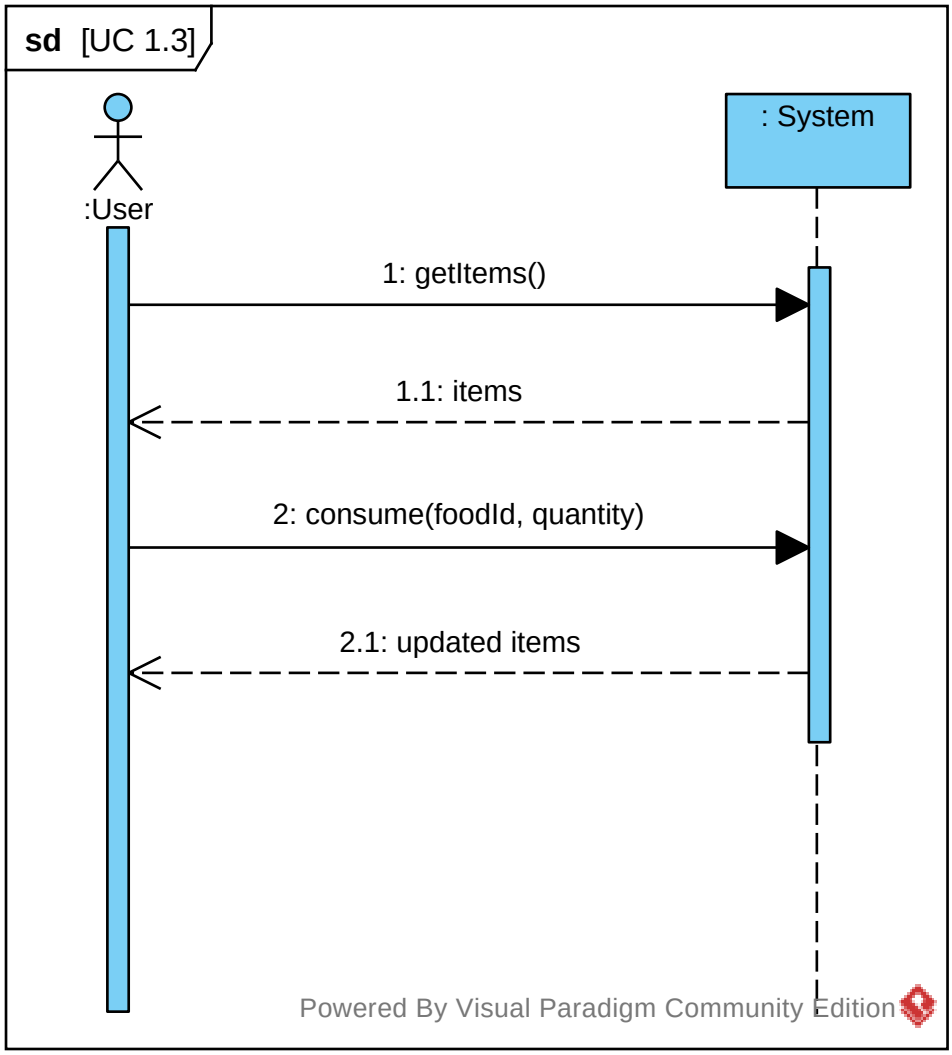

sd [SSD_3.1]
User
: System
1: getRecipes()
1.1: recipes
opt
[recipes empty]
2: getRecommendedRecipes()
2.1: recommended recipes
loop
[missing information]
3: addRecipe(name, ingredients, servings)
3.1: form response
Powered By Visual Paradigm Community Edition

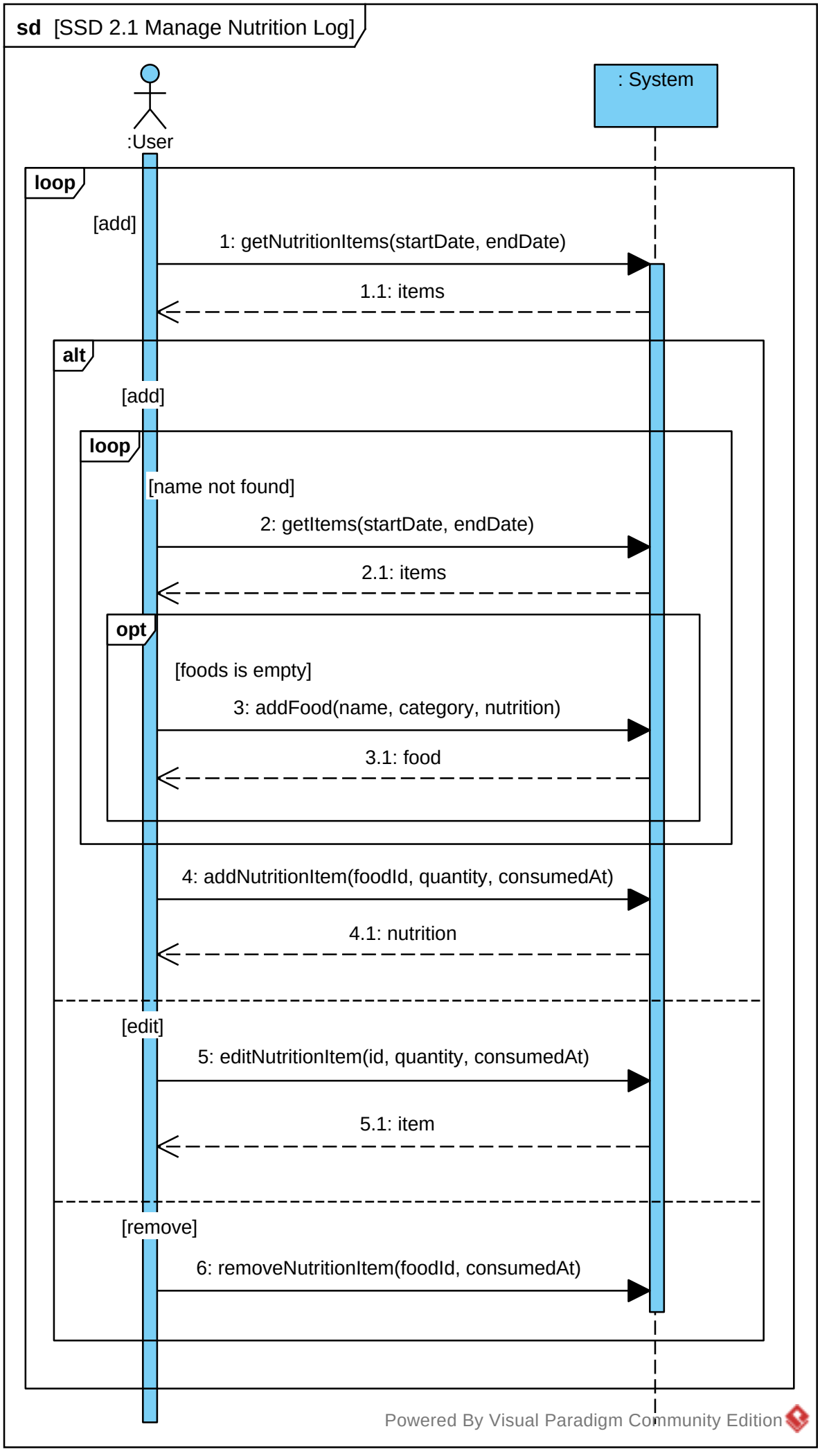

sd [SSD_3.2]
User
: System
1: getRecipes()
1.1: recipes
opt
[recipes empty]
2: getRecommendedRecipes()
2.1: recommended recipes
3: searchByRecipeName(query)
3.1: search results
opt
[results empty]
4: getRecipesByIngredients(ingredients)
4.1: recipe results
5: getRecipe(recipeID)
5.1: recipe information
Powered By Visual Paradigm Community Edition

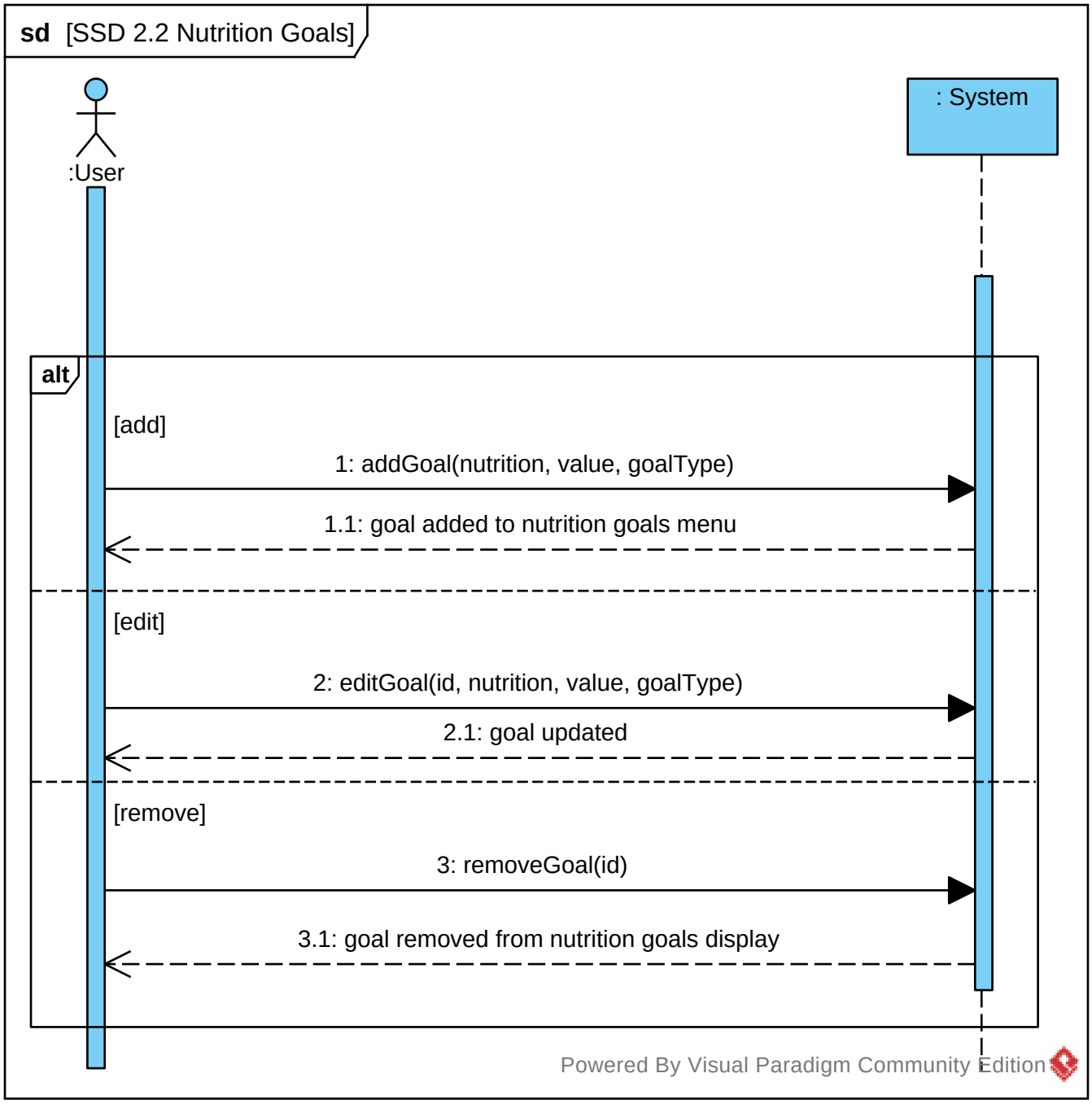

sd [SSD_3.3]
User
: System
1: getRecipes()
1.1: recipes
opt
[recipes empty]
2: getRecommendedRecipes()
2.1: recommended recipes
3: selectRecipe(recipeID)
3.1: recipe information
4: addIngredientsToCart(recipeID)
4.1: recipe added to cart
Powered By Visual Paradigm Community Edition

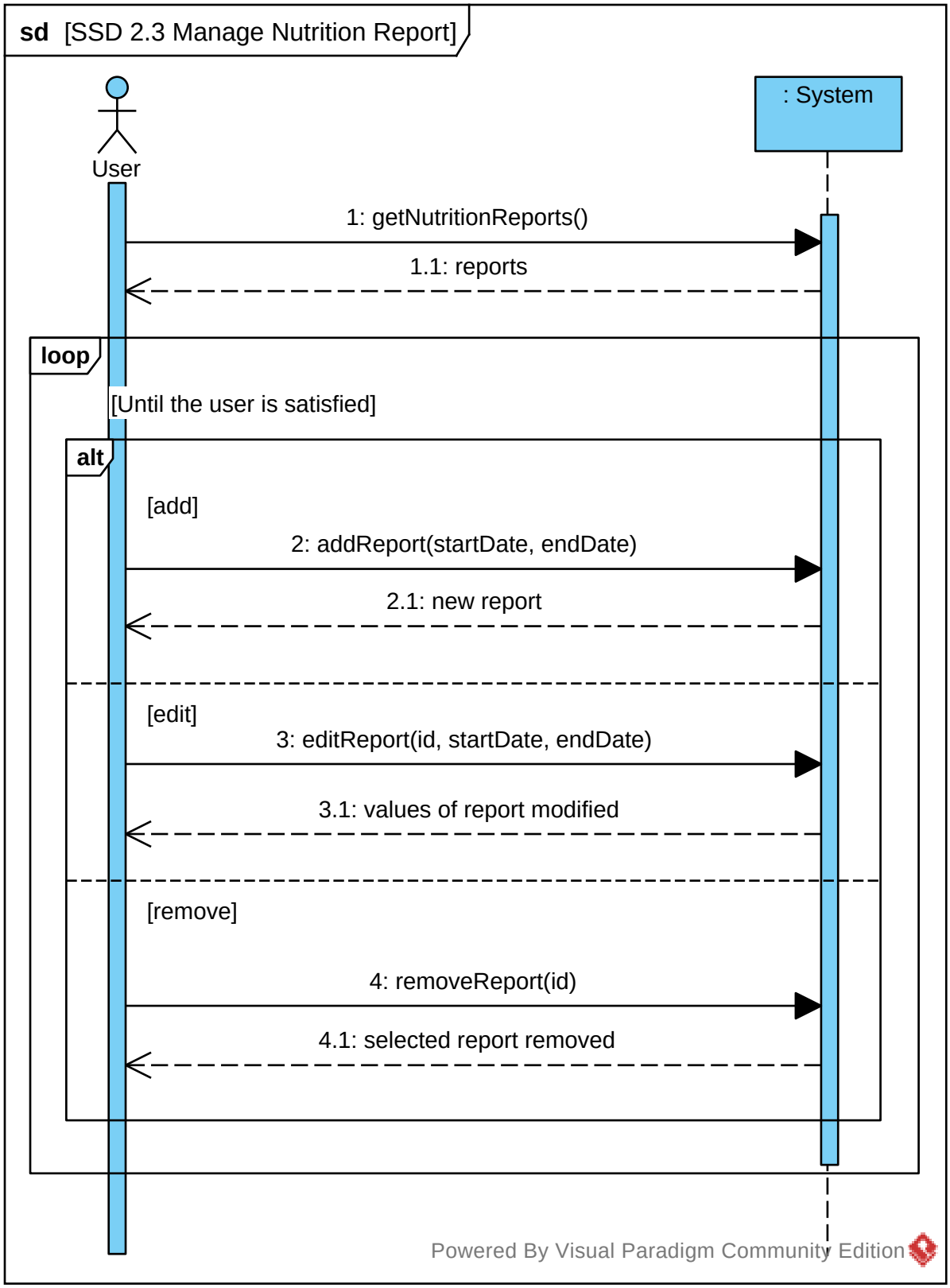

sd [UC 1.1]
:User
: System
1: getItems()
1.1: items
alt
[add food item]
2: addItem(foodId, quantity)
opt
[food id not found]
3: addFood(name, category, nutrition)
3.1: updated items
[edit food item]
4: editItem(foodId, quantity)
4.1: updated items
[remove food item]
5: removeItem(foodId)
5.1: updated items
[consume item]
6: consume(foodId, quantity)
[search items]
7: searchByFoodName(query)
Powered By Visual Paradigm Community Edition


sd [UC 1.2]
:User
: System
1: getItems
1.1: items
2: searchByFoodName(query)
opt
[food item not found]
3: addFood(name, category, nutrition)
3.1: updated items
Powered By Visual Paradigm Community Edition


sd [UC 1.3]
:User
: System
1: getItems()
1.1: items
2: consume(foodId, quantity)
2.1: updated items
Powered By Visual Paradigm Community Edition


sd [SSD 2.1 Manage Nutrition Log]
:User
: System
loop
[add]
1: getNutritionItems(startDate, endDate)
1.1: items
alt
[add]
loop
[name not found]
2: getItems(startDate, endDate)
2.1: items
opt
[foods is empty]
3: addFood(name, category, nutrition)
3.1: food
4: addNutritionItem(foodId, quantity, consumedAt)
4.1: nutrition
[edit]
5: editNutritionItem(id, quantity, consumedAt)
5.1: item
[remove]
6: removeNutritionItem(foodId, consumedAt)
Powered By Visual Paradigm Community Edition

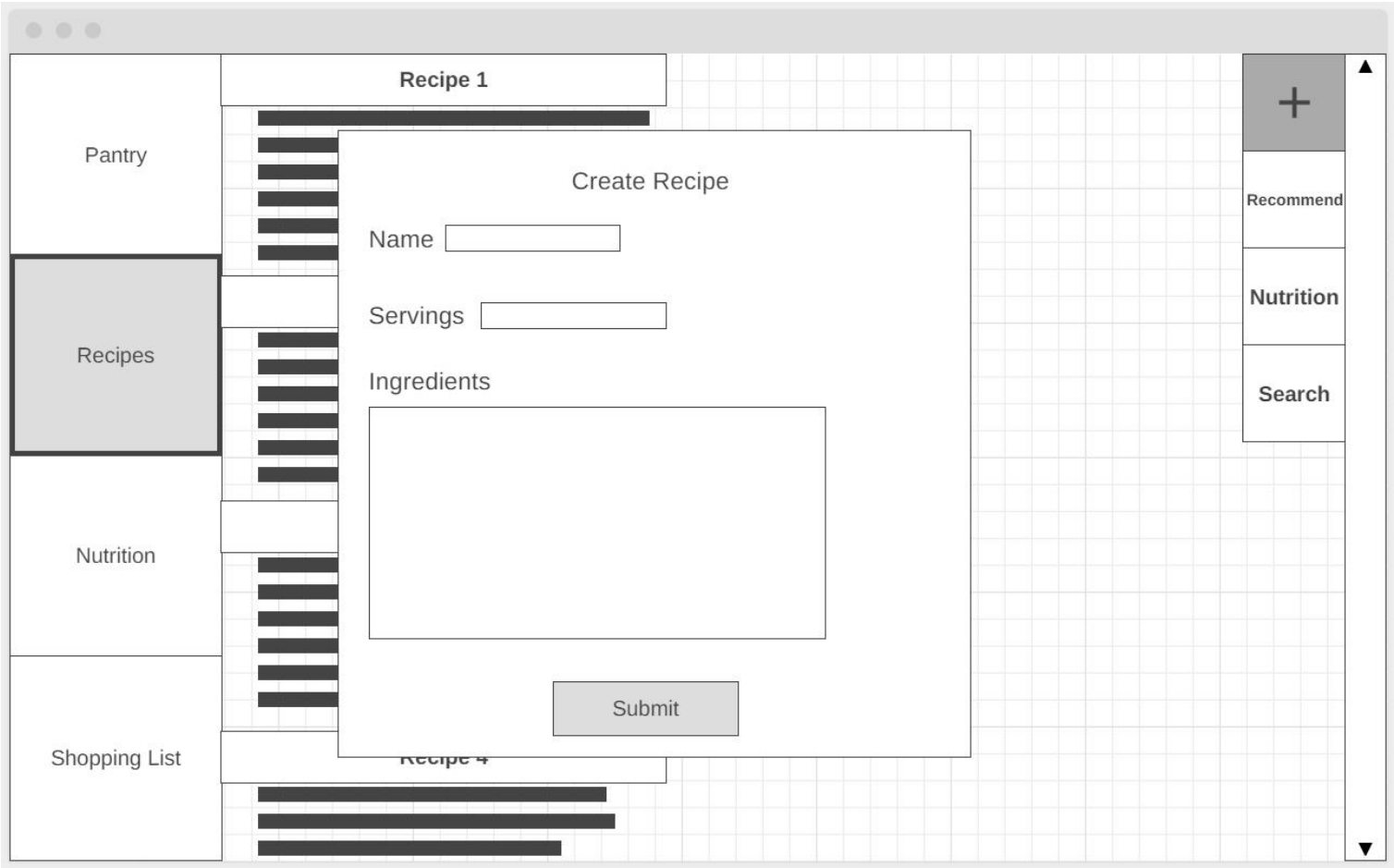

sd [SSD 2.2 Nutrition Goals]
:User
: System
alt
[add]
1: addGoal(nutrition, value, goalType)
1.1: goal added to nutrition goals menu
[edit]
2: editGoal(id, nutrition, value, goalType)
2.1: goal updated
[remove]
3: removeGoal(id)
3.1: goal removed from nutrition goals display
Powered By Visual Paradigm Community Edition

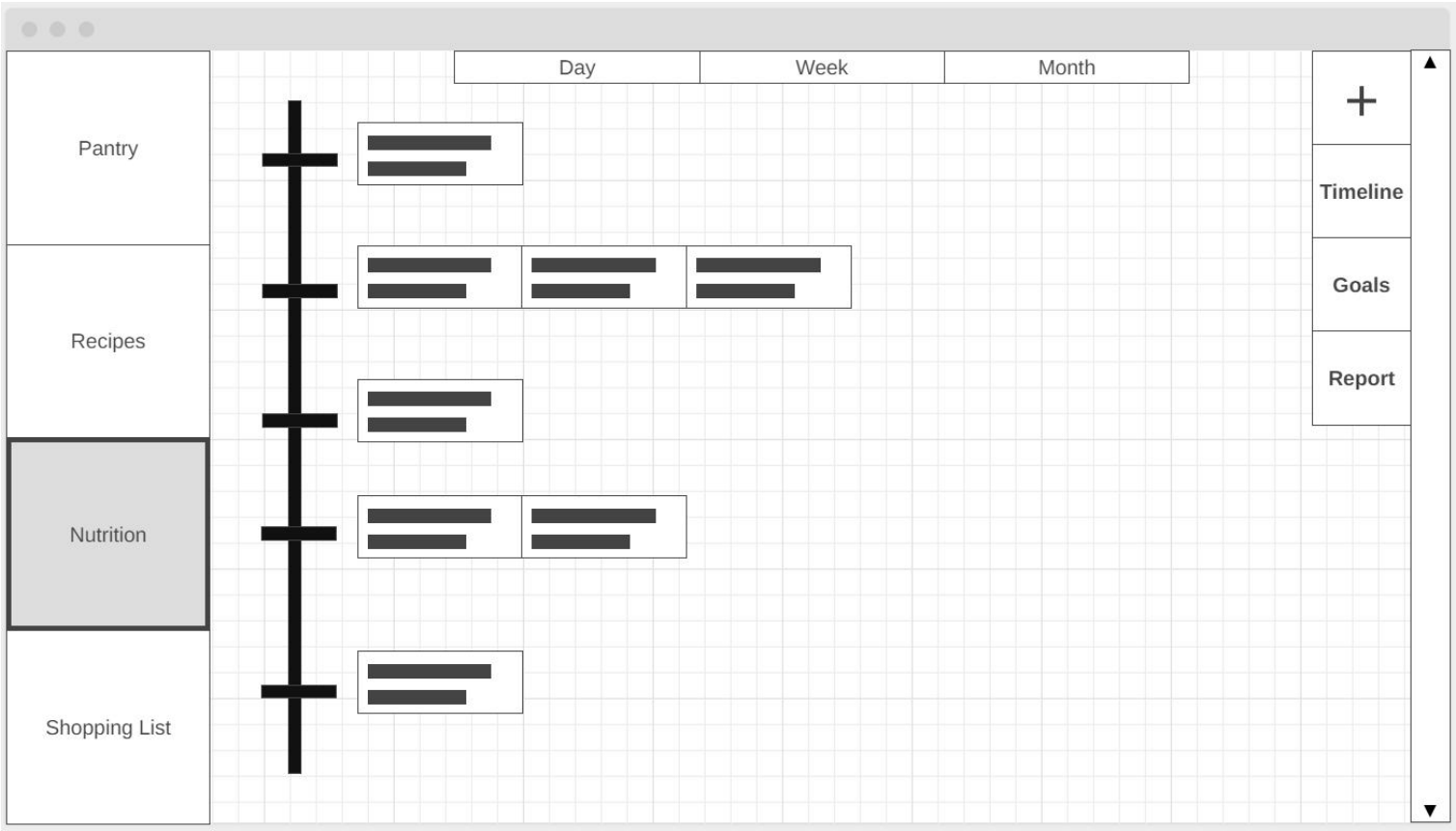

sd [SSD 2.3 Manage Nutrition Report]
User
: System
1: getNutritionReports()
1.1: reports
loop
[Until the user is satisfied]
alt
[add]
2: addReport(startDate, endDate)
2.1: new report
[edit]
3: editReport(id, startDate, endDate)
3.1: values of report modified
[remove]
4: removeReport(id)
4.1: selected report removed
Powered By Visual Paradigm Community Edition

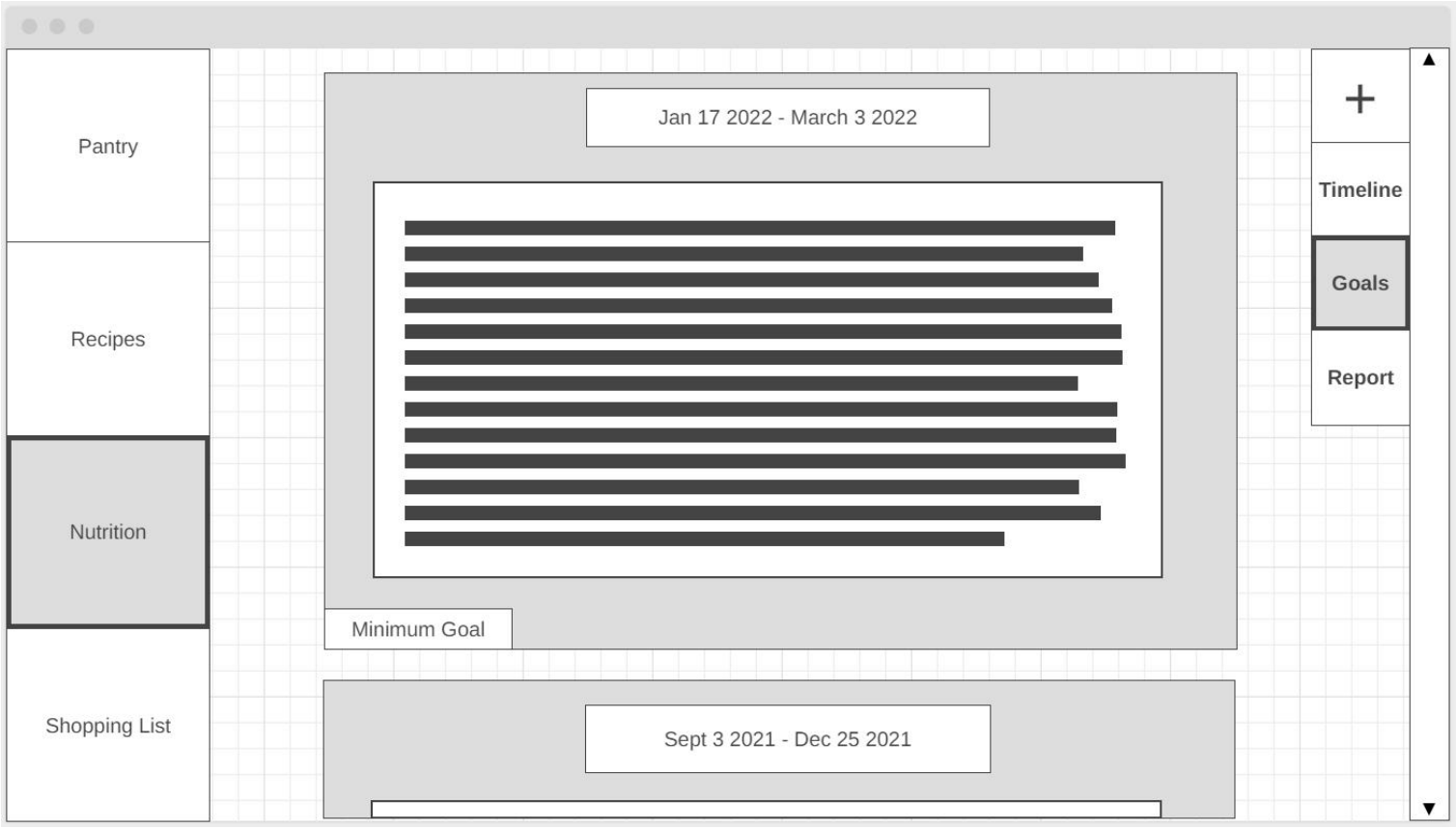

Pantry
Recipes
Nutrition
Shopping List
+
Search
Modify

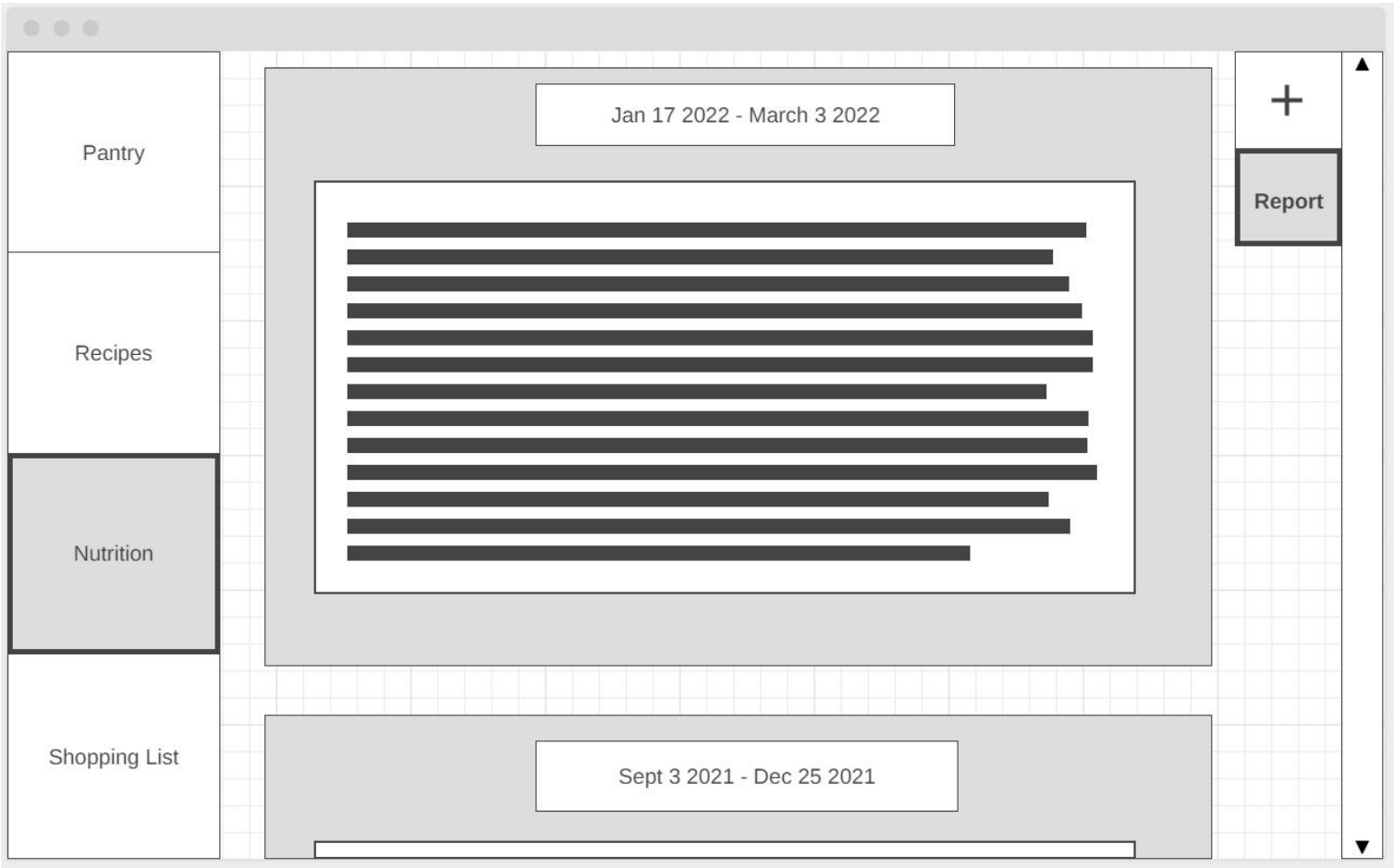

Pantry
Recipes
Nutrition
Shopping List
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
Edit Delete
+
Search
Modify

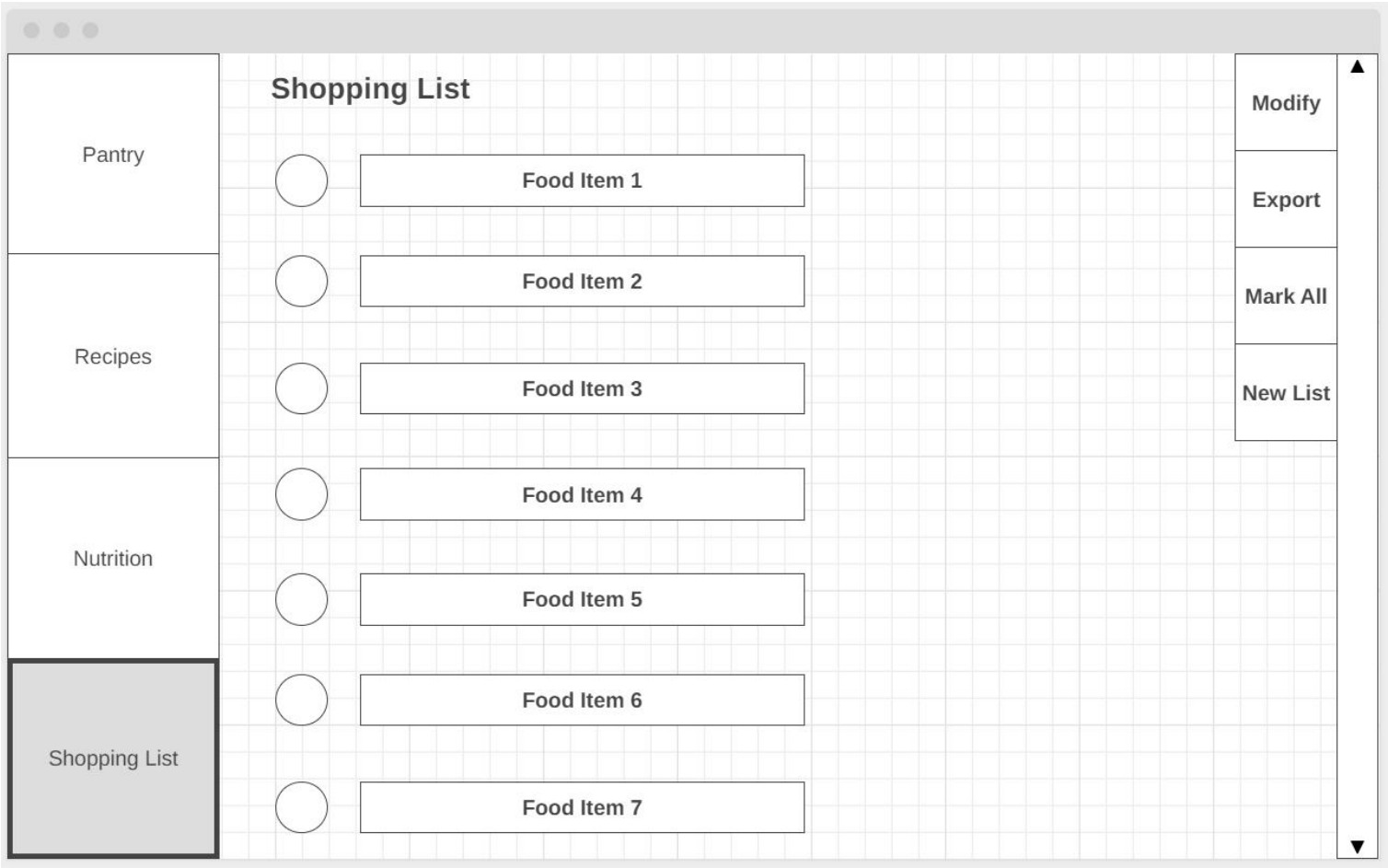

Pantry
Recipes
Nutrition
Shopping List
Recipe 1
Recipe 2
Recipe 3
Recipe 4
+
Recommend
Nutrition
Search

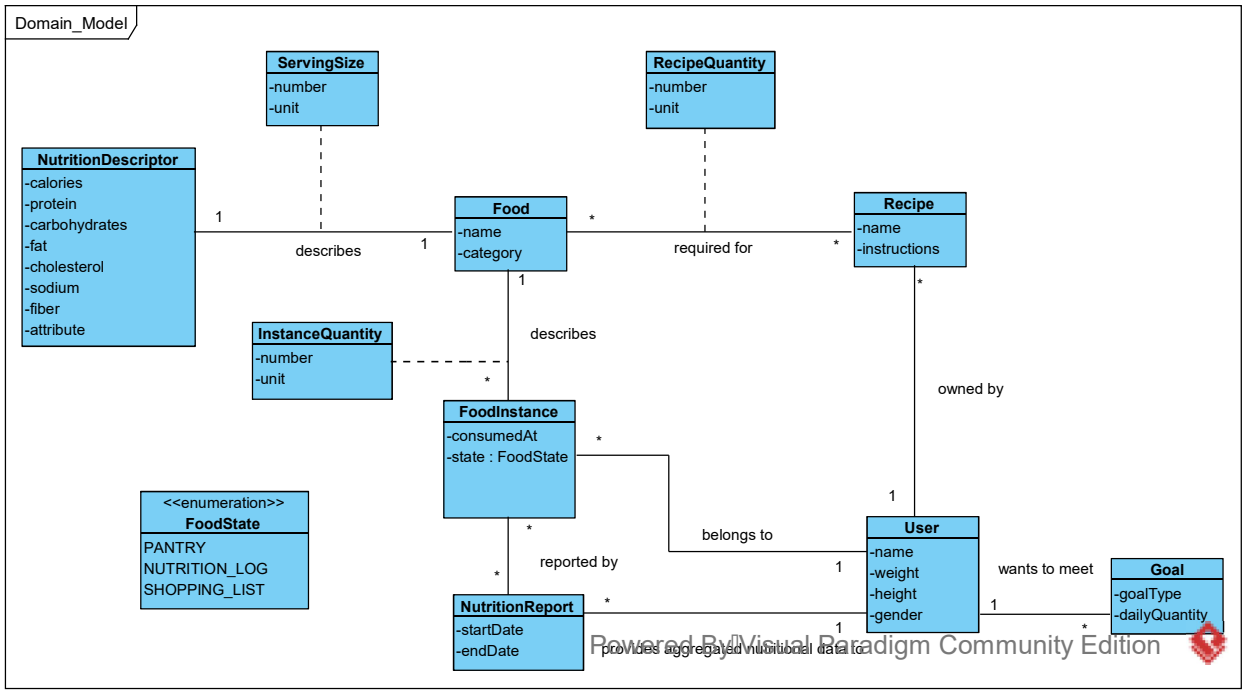

Recipe 1
Pantry
Recipes
Nutrition
Shopping List
Create Recipe
Name
Servings
Ingredients
Submit
Recipe 4
+
Recommend
Nutrition
Search


Day
Week
Month
Pantry
Recipes
Nutrition
Shopping List
Timeline
Goals
Report


Pantry
Recipes
Nutrition
Shopping List
Jan 17 2022 - March 3 2022
Minimum Goal
Sept 3 2021 - Dec 25 2021
+
Timeline
Goals
Report


Pantry
Recipes
Nutrition
Shopping List
Jan 17 2022 - March 3 2022
Sept 3 2021 - Dec 25 2021
Report


Pantry
Recipes
Nutrition
Shopping List
Shopping List
Food Item 1
Food Item 2
Food Item 3
Food Item 4
Food Item 5
Food Item 6
Food Item 7
Modify
Export
Mark All
New List


Domain_Model
ServingSize
-number
-unit
RecipeQuantity
-number
-unit
NutritionDescriptor
-calories
-protein
-carbohydrates
-fat
-cholesterol
-sodium
-fiber
-attribute
Food
-name
-category
Recipe
-name
-instructions
describes
required for
InstanceQuantity
-number
-unit
describes
owned by
FoodInstance
-consumedAt
-state : FoodState
<<enumeration>>
FoodState
PANTRY
NUTRITION_LOG
SHOPPING_LIST
belongs to
reported by
User
-name
-weight
-height
-gender
wants to meet
Goal
-goalType
-dailyQuantity
NutritionReport
-startDate
-endDate
provides aggregated nutritional data to
Powered By Visual Paradigm Community Edition

Operation Contracts

STARTUP CONTRACTS

Contract CO1: register

Operation:	register(name, info)
Cross References:	Use Cases: 5.1 Startup
Preconditions:	• Application is running
Postconditions:	• A user instance has been instantiated • User attributes have been initialized (name, opt. weight, gender) • User data has been saved to the database

PANTRY CONTRACTS

Contract CO2: getPantryItems

Operation:	getPantryItems()
Cross References:	Use Cases: 5.1 Startup, 1.1 Manage Pantry, 1.2 Search Pantry, 1.3 Consume Pantry Item
Preconditions:	• User has been registered within the system and initialized • Data has been loaded from the database
Postconditions:	• System provides pantry data to user

Contract CO3: addPantryItem

Operation:	addPantryItem(foodId, quantity)
Cross References:	Use Cases: 5.1 Startup, 1.1 Manage Pantry
Preconditions:	• Data has been loaded from the database • Food already exists within the food database
Postconditions:	• Added food item exists within the user's pantry • Pantry quantity of food item has been incremented to quantity + old quantity • Pantry database updated

Contract CO4: editPantryItem

Operation:	editPantryItem(foodId, quantity)
Cross References:	Use Cases: 1.1 Manage Pantry
Preconditions:	● Data has been loaded from the database ● Food exists within the food database
Postconditions:	● Food item is updated with new quantity ● Pantry database is updated

Contract CO5: removePantryItem

Operation:	removePantryItem(foodId)
Cross References:	Use Cases: 5.1 Startup, 1.1 Manage Pantry
Preconditions:	● Data has been loaded from the database ● An item of the given food type exists within the pantry
Postconditions:	● Food item with given name has its instance removed from pantry ● Pantry database is updated

Contract CO6: searchPantryByFoodName

Operation:	searchPantryByFoodName(query)
Cross References:	Use Cases: 1.2 Search Pantry
Preconditions:	● Data has been loaded from the database
Postconditions:	● List of food items starting with item name are returned or an empty list if no pantry items match the itemName

Contract CO7: consume

Operation:	consume(foodId, quantity)
Cross References:	Use Cases: 1.3 Consume Pantry Item
Preconditions:	● Data has been loaded from the database ● FoodItem passed in exists in the pantry
Postconditions:	● Food in pantry with type foodItem has its quantity decremented ● If new quantity is 0, foodItem is removed from pantry ● Database is updated

SHOPPING LIST CONTRACTS

Contract CO8: getShoppingItems

Operation:	getShoppingItems()
Cross References:	Use Cases: 4.1 Manage Shopping List, 4.3 Mark Purchased Items
Preconditions:	• Data has been loaded from the database
Postconditions:	• System provides user with their shopping list

Contract CO9: addShoppingItem

Operation:	addShoppingItem(foodId, quantity)
Cross References:	Use Cases: 4.1 Manage Shopping List
Preconditions:	• Data has been loaded from the database • The food type exists within the list of registered food types • The quantity is valid (non-negative, etc)
Postconditions:	• Food item with given quantity added to the shopping list • Database is updated

Contract CO10: editShoppingItem

Operation:	editShoppingItem(foodId, quantity)
Cross References:	Use Cases: 4.1 Manage Shopping List
Preconditions:	• Data has been loaded from the database • The food type exists within the shopping list • The quantity is valid (non-negative, etc)
Postconditions:	• Quantity of given food item updated within the shopping list • Database is updated

Contract CO11: removeShoppingItem

Operation:	removeShoppingItem(foodId)
Cross References:	Use Cases: 4.1 Manage Shopping List
Preconditions:	• Data has been loaded from the database • The food type exists within the shopping list • The quantity is valid (non-negative, etc)
Postconditions:	• Food item removed from shopping list • Database is updated

Contract CO12: export

Operation:	export(fileFormat, destination)
Cross References:	Use Cases: 4.1 Manage Shopping List, 4.2 Export Shopping List
Preconditions:	• Data has been loaded from the database • User has access to the given destination • Given destination is valid • File format is valid
Postconditions:	• Shopping list has been exported to desired format

Contract CO13: purchaseItems

Operation:	purchaseItems(foodIds)
Cross References:	Use Cases: 4.3 Mark Purchased Items
Preconditions:	• Data has been loaded from the database • Given foods are registered in the list of foods • User purchased given items from the shopping list
Postconditions:	• System moves given items from shopping list to pantry • Database is updated

RECIPE CONTRACTS

Contract CO14: getRecipes

Operation:	getRecipes()
Cross References:	Use Cases: 3.1 Create Recipe, 3.2 View Recipe, 3.3 Add Recipe to Shopping List, 3.4 Modify Recipe, 3.5 Recommend Recipes
Preconditions:	● Data has been loaded from the database
Postconditions:	● System provides the user's list of recipes

Contract CO15: addRecipe

Operation:	addRecipe(name, ingredients, instructions, servings)
Cross References:	Use Cases: 5.1 Startup, 3.4 Modify Recipe
Preconditions:	● Data has been loaded from the database
Postconditions:	● Ingredients have been stored has food instances within the database ● The recipe represents a new recipe instance within the recipe system ● Recipe appears within the recipe list

Contract CO16: produceCookedRecipe

Operation:	produceCookedRecipe(recipeId, isUsePantry, consumedServings, leftoverServings)
Cross References:	Use Cases: 3.6 Cook Recipe
Preconditions:	● Recipe exists within the recipe system ● Ingredients exist within the food database ● Data has been loaded from the database
Postconditions:	● User is notified of any errors preventing cooking ● Consumed portion is logged to food log ● Pantry is updated to contain leftovers ● Database is updated

Contract CO17: editRecipe

Operation:	editRecipe(recipeId, recipe)
Cross References:	Use Cases: 3.4 Modify Recipe
Preconditions:	● Data has been loaded from the database ● recipeId is valid
Postconditions:	● User is notified of any errors preventing modification ● Recipe is saved to Recipe List ● Database is updated

Contract CO18: getRecipe

Operation:	getRecipe(recipeId)
Cross References:	Use Cases: 3.5 Recommend Recipes
Preconditions:	● Data has been loaded from the database ● Recipe exists within the recipe system
Postconditions:	● Recipe is displayed to the user

Contract CO19: getRecommendRecipes

Operation:	getRecommendRecipes()
Cross References:	Use Cases: 3.5 Recommend Recipe
Preconditions:	● Data has been loaded from the database
Postconditions:	● The user is presented with all recommended recipes from the Recipe List based off of items in their pantry

Contract CO20: searchByRecipeName

Operation:	searchByRecipeName(query)
Cross References:	Use Cases: 3.2 View Recipes
Preconditions:	● Data has been loaded from the database
Postconditions:	● List of matching recipes are displayed to the user

Contract CO21: addIngredientsToCart

Operation:	addIngredientsToCart(recipeId)
Cross References:	Use Cases: 3.3 Add Recipe to Shopping Cart
Preconditions:	● Data has been loaded from the database
Postconditions:	● Recipe ingredients added to shopping cart ● Database updated

Contract CO22: addMissingIngredientsToCart

Operation:	addMissingIngredientsToCart(recipeId)
Cross References:	Use Cases: 3.3 Add Recipe to Shopping Cart
Preconditions:	● Data has been loaded from the database
Postconditions:	● Recipe ingredients not in pantry added to shopping cart ● Database updated

NUTRITION LOG CONTRACTS

Contract CO23: addNutritionItem

Operation:	addNutritionItem(foodId, quantity, consumedAt)
Cross References:	Use Cases: 3.6 Cook Recipe, 2.1 Manage Nutrition Log
Preconditions:	● Data has been loaded from the database ● Food exists within the recipe system ● Quantity is valid (non-negative, valid unit, etc.)
Postconditions:	● Food of given quantity logged to nutrition log at given time ● Nutrition log is updated with new nutrition instance ● Database is updated

Contract CO24: getNutritionItems

Operation:	getNutritionItems(startDate, endDate)
Cross References:	Use Cases: 2.1 Manage Nutrition Log
Preconditions:	● Data has been loaded from the database
Postconditions:	● System provides the list of nutrition instances from the given time window

Contract CO25: editNutritionItem

Operation:	editNutritionItem(foodId, quantity, consumedAt)
Cross References:	Use Cases: 2.1 Manage Nutrition Log
Preconditions:	● Data has been loaded from the database ● Quantity is valid (non-negative, valid unit, etc.)
Postconditions:	● The selected item has had its values changed as per user request ● Database is updated

Contract CO26: removeNutritionItem

Operation:	removeNutritionItem(foodId, consumedAt)
Cross References:	Use Cases: 2.1 Manage Nutrition Log
Preconditions:	● Data has been loaded from the database ● The given nutrition item is already in the food log
Postconditions:	● The selected item has been removed from the log ● Database is updated

Contract CO27: addGoal

Operation:	addGoal(nutrient, quantity, goalType)
Cross References:	Use Cases: 2.2 Nutrition Goals
Preconditions:	● Data has been loaded from the database ● The input values are valid and logical
Postconditions:	● A nutrition goal has been created using the given fields ● Database is updated

Contract CO28: editGoal

Operation:	editGoal(id, nutrient, quantity, goalType)
Cross References:	Use Cases: 2.2 Nutrition Goals
Preconditions:	● Data has been loaded from the database ● The input values are valid and logical
Postconditions:	● The selected goal has had its values changed as per user request ● Database is updated

Contract CO29: removeGoal

Operation:	removeGoal(id)
Cross References:	Use Cases: 2.2 Nutrition Goals
Preconditions:	● Data has been loaded from the database ● The given goal exists within the user's list of goals
Postconditions:	● The selected goal has been removed ● Database is updated

Contract CO30: addReport

Operation:	addReport(startDate, endDate)
Cross References:	Use Cases: 2.3 Manage Nutrition Report
Preconditions:	● The user is in the nutrition reports menu ● The input values are valid and logical
Postconditions:	● A nutrition report has been created using the given fields

Contract CO31: editReport

Operation:	editReport(id, startDate, endDate)
Cross References:	Use Cases: 2.3 Manage Nutrition Report
Preconditions:	● The user has selected a report to be edited ● The input values are valid and logical
Postconditions:	● A nutrition report has been edited

Contract CO32: removeReport

Operation:	deleteReport(id)
Cross References:	Use Cases: 2.3 Manage Nutrition Report
Preconditions:	● The user is in the nutrition report menu ● At least one report exists within the menu which the user wants to delete
Postconditions:	● The selected report has been removed from the report menu

FOOD CONTRACTS

Contract CO33: getFood

Operation:	getFood(id)
Cross References:	Use Cases: 6.1 Manage Food Types, 6.2 Create New Food Types, 2.1 Manage Nutrition Log
Preconditions:	● Data has been loaded from the database
Postconditions:	● System provides user with the food of the given id

Contract CO34: addFood

Operation:	addFood(name, category, nutrition), 6.2 Create New Food Types
Cross References:	Use Cases: 6.1 Manage Food Types, 4.1 Manage Shopping List, 1.1 Manage Pantry, 1.2 Search Pantry, 2.1 Manage Nutrition Log
Preconditions:	● Data has been loaded from the database ● The name is not already in use by another food ● The name is not empty ● The nutrition info is valid (non-negative macros, etc)
Postconditions:	● System adds food to list of foods ● Database is updated

Contract CO35: editFood

Operation:	editFood(id, name, category, nutrition)
Cross References:	Use Cases: 6.1 Manage Food Types
Preconditions:	• Data has been loaded from the database • The info is valid • The id refers to an already registered food
Postconditions:	• System updates food in list of foods • Database is updated

CO36: removeFood

Operation:	removeFood(id)
Cross References:	Use Cases: 6.1 Manage Food Types
Preconditions:	• Data has been loaded from the database • The id refers to an already registered food
Postconditions:	• System removes given food from the list of food • Database is updated

CO37: searchFoodsByName

Operation:	searchFoodsByName(query)
Cross References:	Use Cases: 4.1 Manage Shopping List
Preconditions:	• Data has been loaded from the database
Postconditions:	• List of food items starting with item name are returned or an empty list if no pantry items match the itemName

FoodPants-Gantt

Feb 23, 2022

http://

Boothverse

Project manager	Austin Huizinga
Project dates	Jan 18, 2022 - Apr 28, 2022
Completion	0%
Tasks	28
Resources	4

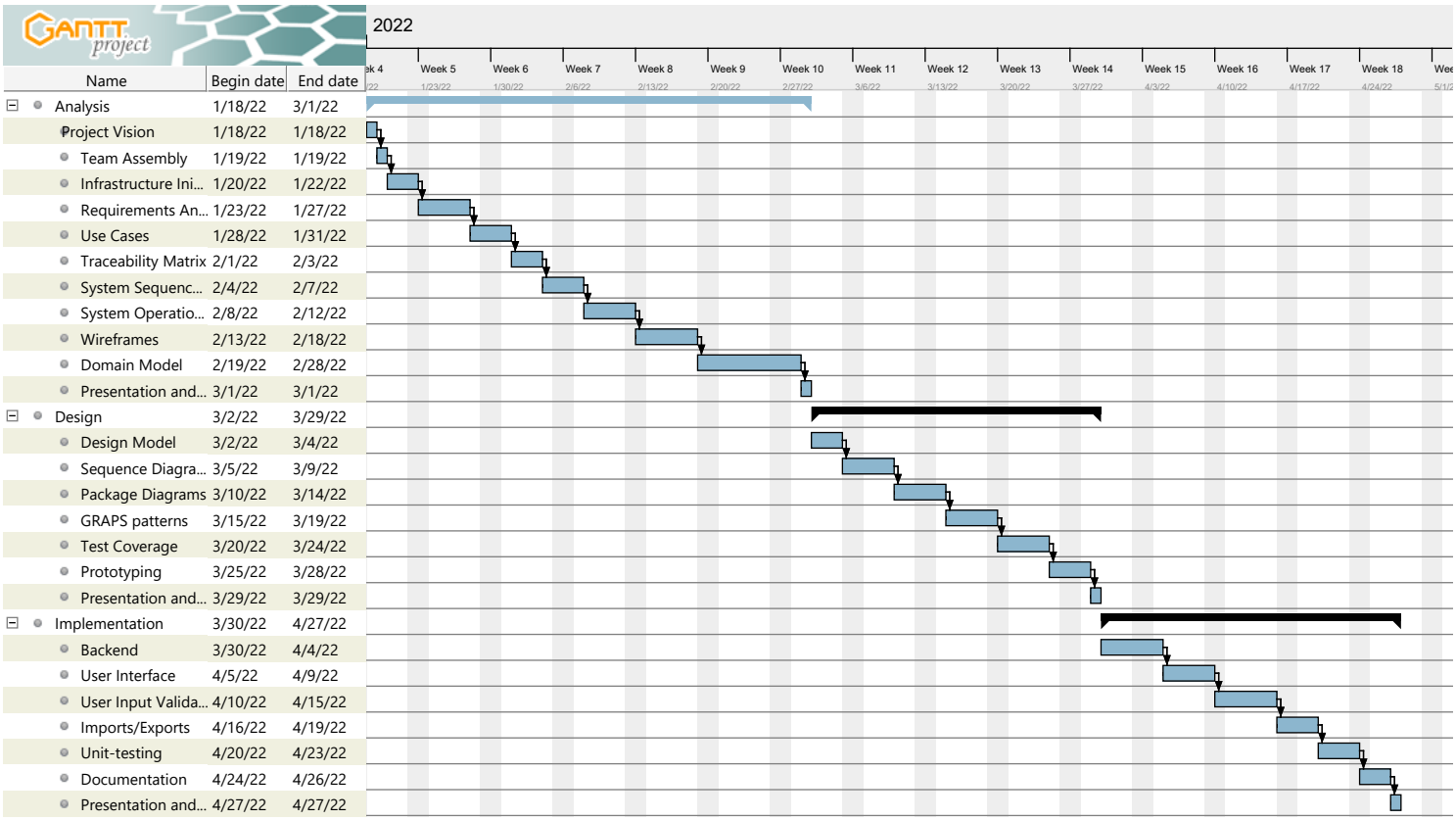
Tasks

Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/19/22
Infrastructure Initialization	1/20/22	1/22/22
Requirements Analysis	1/23/22	1/27/22
Use Cases	1/28/22	1/31/22
Traceability Matrix	2/1/22	2/3/22
System Sequence Diagrams	2/4/22	2/7/22
System Operations	2/8/22	2/12/22
Wireframes	2/13/22	2/18/22
Domain Model	2/19/22	2/28/22
Presentation and Reporting	3/1/22	3/1/22
Design	3/2/22	3/29/22
Design Model	3/2/22	3/4/22
Sequence Diagrams	3/5/22	3/9/22
Package Diagrams	3/10/22	3/14/22
GRAPS patterns	3/15/22	3/19/22
Test Coverage	3/20/22	3/24/22
Prototyping	3/25/22	3/28/22
Presentation and Reporting	3/29/22	3/29/22
Implementation	3/30/22	4/27/22
Backend	3/30/22	4/4/22
User Interface	4/5/22	4/9/22
User Input Validation	4/10/22	4/15/22
Imports/Exports	4/16/22	4/19/22
Unit-testing	4/20/22	4/23/22
Documentation	4/24/22	4/26/22
Presentation and Reporting	4/27/22	4/27/22

Resources

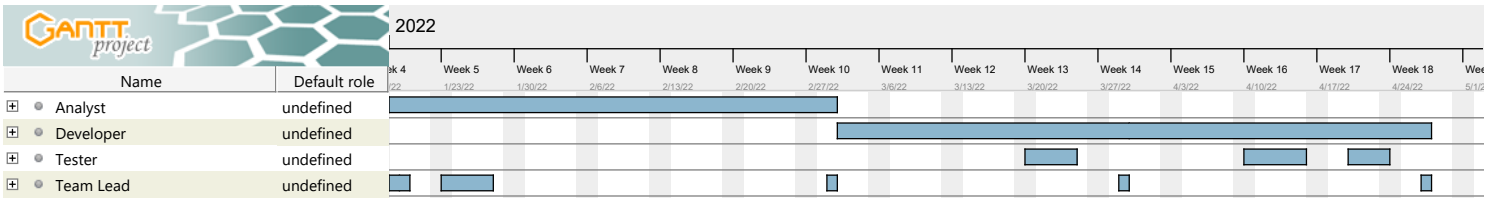
Name	Default role
Analyst	undefined
Developer	undefined
Tester	undefined
Team Lead	undefined

Gantt Chart

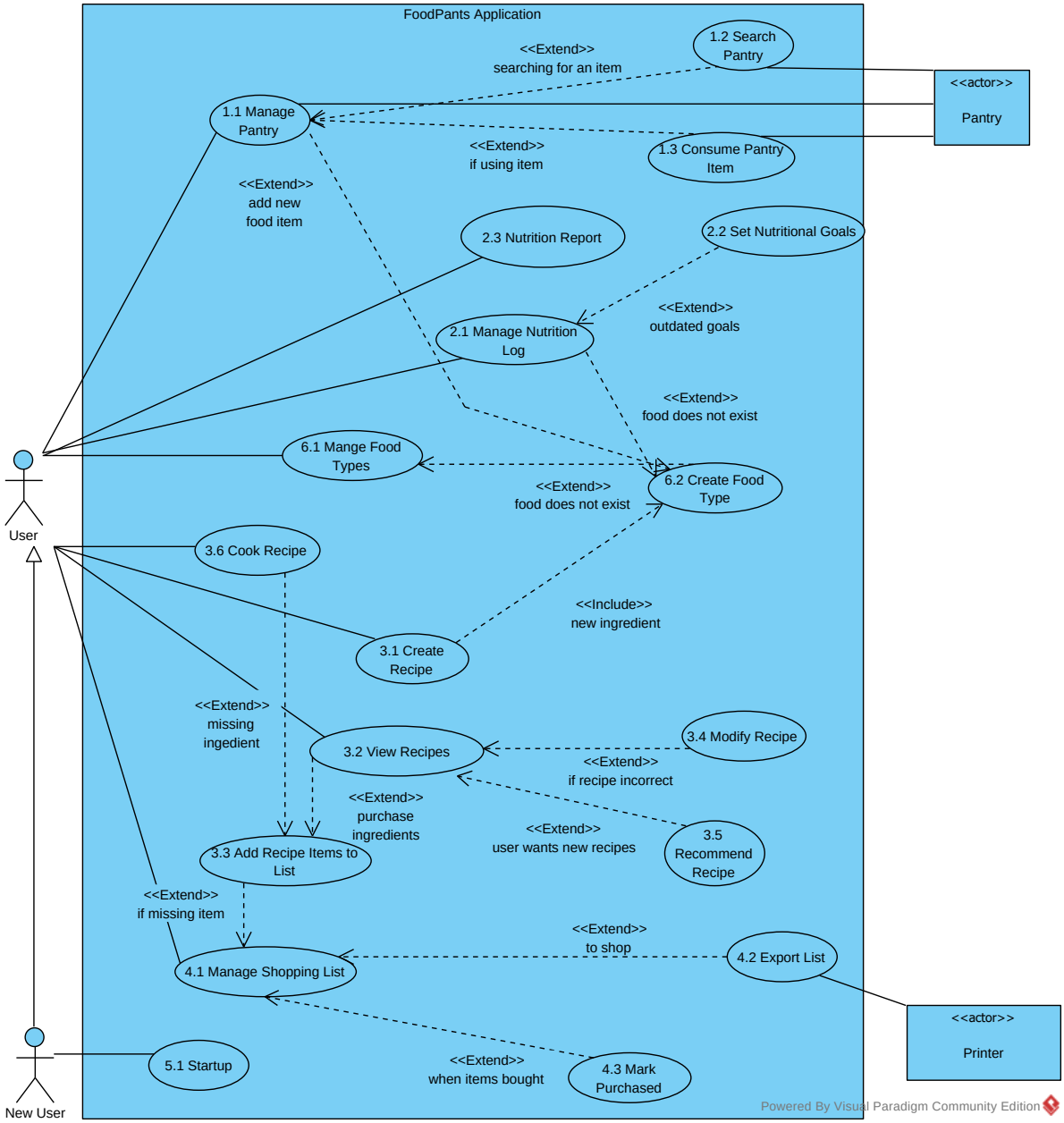


Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/19/22
Infrastructure Ini...	1/20/22	1/22/22
Requirements An...	1/23/22	1/27/22
Use Cases	1/28/22	1/31/22
Traceability Matrix	2/1/22	2/3/22
System Sequenc...	2/4/22	2/7/22
System Operatio...	2/8/22	2/12/22
Wireframes	2/13/22	2/18/22
Domain Model	2/19/22	2/28/22
Presentation and...	3/1/22	3/1/22
Design	3/2/22	3/29/22
Design Model	3/2/22	3/4/22
Sequence Diagra...	3/5/22	3/9/22
Package Diagrams	3/10/22	3/14/22
GRAPS patterns	3/15/22	3/19/22
Test Coverage	3/20/22	3/24/22
Prototyping	3/25/22	3/28/22
Presentation and...	3/29/22	3/29/22
Implementation	3/30/22	4/27/22
Backend	3/30/22	4/4/22
User Interface	4/5/22	4/9/22
User Input Valida...	4/10/22	4/15/22
Imports/Exports	4/16/22	4/19/22
Unit-testing	4/20/22	4/23/22
Documentation	4/24/22	4/26/22
Presentation and...	4/27/22	4/27/22

Resources Chart



Name	Default role
Analyst	undefined
Developer	undefined
Tester	undefined
Team Lead	undefined


FoodPants Application
1.2 Search Pantry
<<Extend>>
searching for an item
<<actor>>
Pantry
1.1 Manage Pantry
<<Extend>>
if using item
1.3 Consume Pantry Item
<<Extend>>
add new food item
2.3 Nutrition Report
2.2 Set Nutritional Goals
<<Extend>>
outdated goals
2.1 Manage Nutrition Log
<<Extend>>
food does not exist
6.1 Mange Food Types
6.2 Create Food Type
<<Extend>>
food does not exist
User
3.6 Cook Recipe
<<Include>>
new ingredient
3.1 Create Recipe
<<Extend>>
missing ingedient
3.4 Modify Recipe
3.2 View Recipes
<<Extend>>
if recipe incorrect
<<Extend>>
purchase ingredients
<<Extend>>
user wants new recipes
3.5 Recommend Recipe
3.3 Add Recipe Items to List
<<Extend>>
if missing item
<<Extend>>
to shop
4.2 Export List
4.1 Manage Shopping List
<<actor>>
Printer
5.1 Startup
<<Extend>>
when items bought
4.3 Mark Purchased
New User
Powered By Visual Paradigm Community Edition

FoodPants

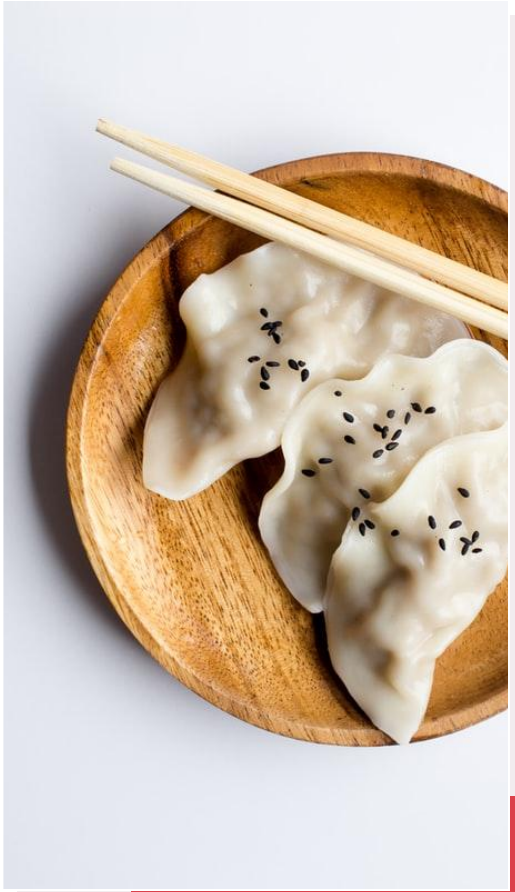
Iteration I

Sick Intro Video

Project Vision

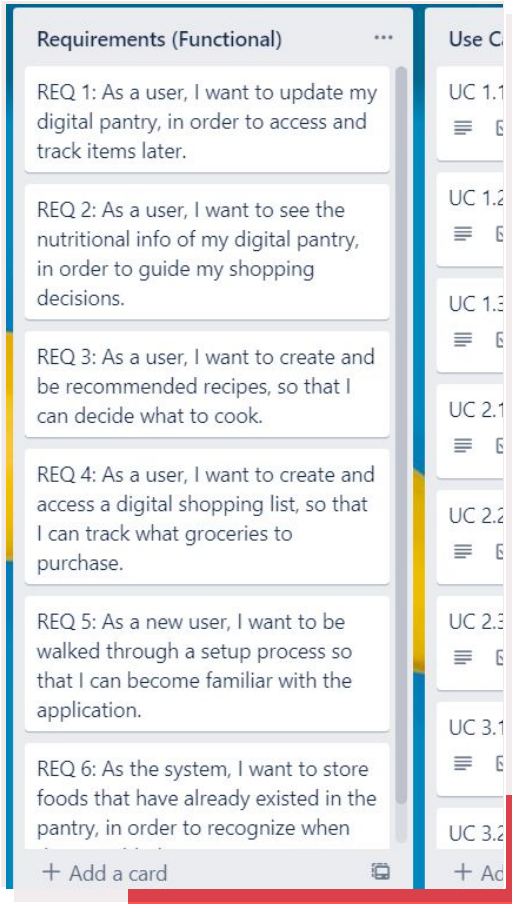
Our team aims to integrate your pantry, recipes, nutritional info, and shopping list into a single application for the ultimate culinary experience.

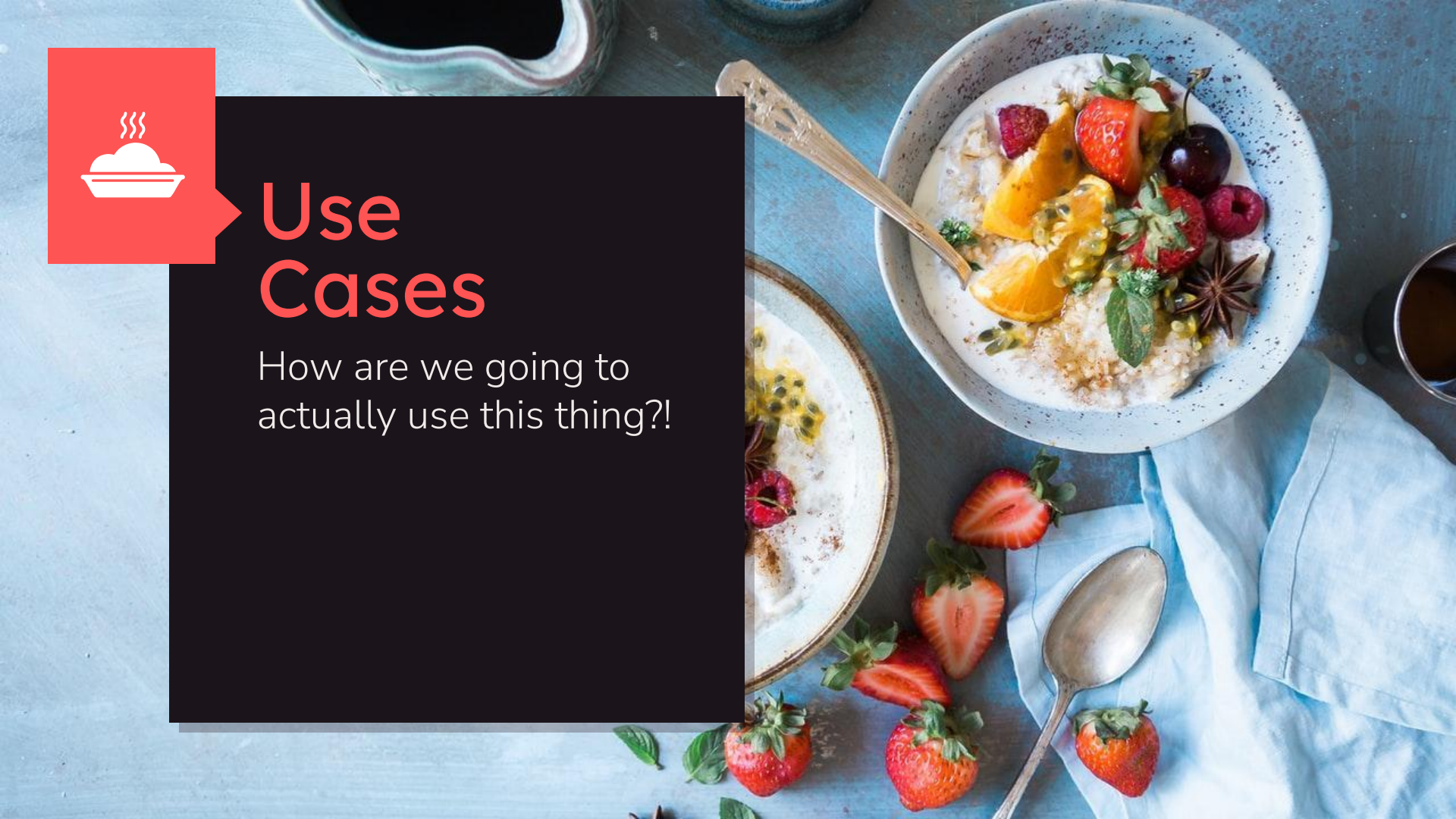
Many applications currently encompass some of these features, but we aim to provide the entire toolset for digitizing your kitchen.



Requirements

- Maintain digital pantry
- Keep track of nutrition info
- Store recipe database
- Provide a shopping list
- Provide a startup experience
- Keep track of types of food





Use Cases

How are we going to actually use this thing?!

Use Cases!

- Manage Pantry
- Search Pantry
- Consume Pantry Item
- Manage Nutrition Log
- Nutrition Goals
- Nutritional Report
- Create Recipe
- View Recipes
- Add Recipe Items to Shopping List
- Modify Recipe
- Recommend Recipe
- Cook Recipe
- Manage Shopping List
- Export Shopping List
- Mark Purchased Items
- Startup Tutorial
- Manage Food Types
- Create Food Type

Traceability Matrix

Use Case Name	Use Case ID	REQ 1	REQ 2	REQ 3	REQ 4	REQ 5	REQ 6	REQ 7	REQ 8	REQ 9	REQ 10
Manage Pantry	1.1	x						x	x		x
Search Pantry	1.2	x							x		x
Consume Pantry Item	1.3	x							x		x
Manage Nutrition Log	2.1		x					x	x		x
Nutrition Goals	2.2		x						x	x	x
Nutritional Report	2.3		x						x		x
Create Recipe	3.1			x			x	x	x	x	
View Recipes	3.2			x				x	x		
Add Recipe Items to Shopping List	3.3			x			x		x		x
Modify Recipe	3.4			x					x	x	
Recommend Recipe	3.5			x				x	x		x
Cook Recipe	3.6	x		x					x		x
Manage Shopping List	4.1				x		x	x	x		x
Export Shopping List	4.2				x				x		x
Mark Purchased Items	4.3				x				x		x
Startup Tutorial	5.1	x	x	x	x	x	x	x	x	x	x
Manage Food Types	6.1						x	x	x	x	
Create New Food Type	6.2						x			x	

ID UC 1.1 Manage Pantry
Scope Pantry system
Level User goal
Stakeholders and interests

- **User:** Person interested in managing a digital pantry. Wants the ability to view food items currently in the pantry. Wants the ability to add food items to their pantry. Wants the ability to modify food items in their pantry. Wants the ability to remove food items from their pantry.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

Precondition: User has gone through setup process.
Postcondition: Pantry is updated with any user modifications.

Main success scenario:

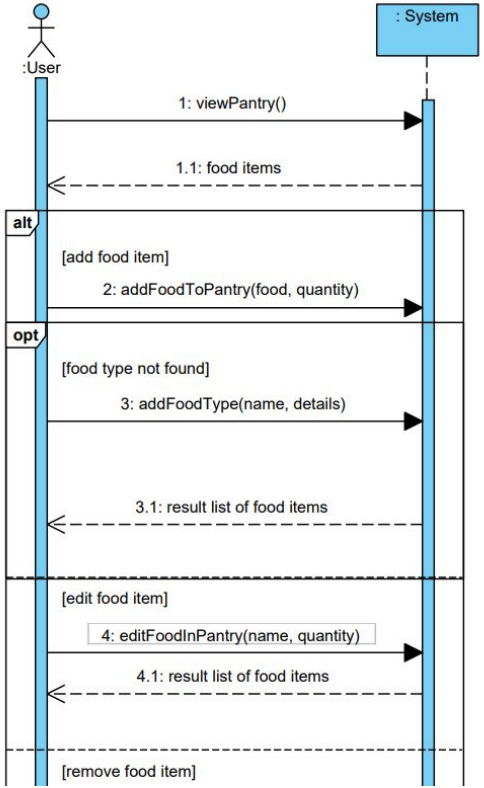
1. User wants to manage pantry.
2. User navigates to the "Pantry" page.
3. System displays a list of all food items in the pantry.
4. User presses the "Add Item" button.
5. User enters the name of the food type to add to pantry in a search box.
6. System displays food type search results in a drop-down menu.
7. User selects the food type to add.
8. User enters the quantity/amount of the food item to add to pantry.
9. System adds food item to pantry.

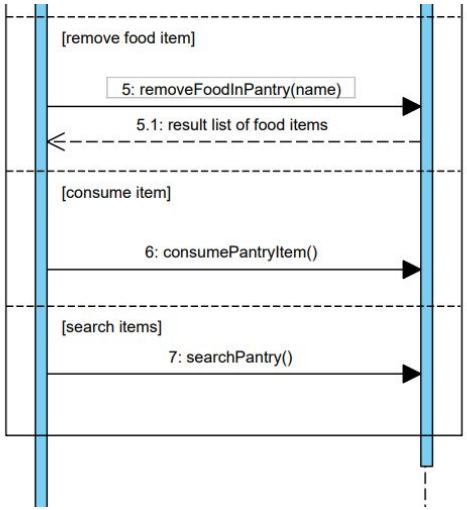
User repeats steps 4-9 until satisfied with the pantry

Extensions:

Example Use Case: Manage Pantry

SSD: 1.1


:User
: System
1: viewPantry()
1.1: food items
alt
[add food item]
2: addFoodToPantry(food, quantity)
opt
[food type not found]
3: addFoodType(name, details)
3.1: result list of food items
[edit food item]
4: editFoodInPantry(name, quantity)
4.1: result list of food items
[remove food item]


[remove food item]
5: removeFoodInPantry(name)
5.1: result list of food items
[consume item]
6: consumePantryItem()
[search items]
7: searchPantry()

ID UC 2.2 Manage Nutrition Goals
Scope Nutrition Log system
Level User goal
Stakeholders and Interests

- **User:** A person utilizing the Nutrition Log system within the FoodPants application to set and manage nutritional goals.
- **System maintainer:** Person responsible for application execution.

Precondition: FoodPants is installed as an executable on the user's machine and the user is currently in their nutrition log.
Postcondition: A nutrition goal has been added, edited, or removed.

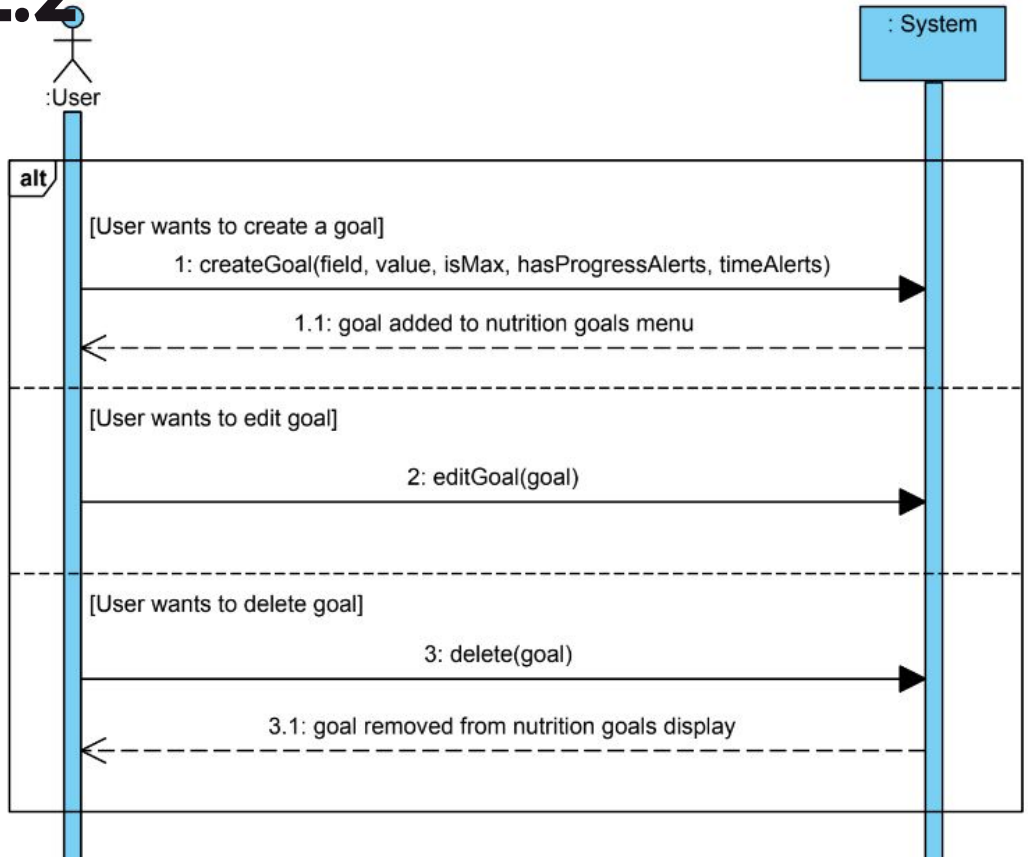
Main success scenario:

1. User wants to manage their nutrition goals
2. User selects the nutrition goals button to navigate to the nutrition goals menu
3. User wants to set a nutrition goal
4. User selects the "+" button and the set nutrition goal menu is displayed
5. User selects the nutrition goal field (calories, carbs, fat, protein, item, cholesterol, sodium)
6. User enters the desired goal value
7. User selects whether the goal is a minimum or a maximum
8. User checks whether the want an alert when nearing their goal
9. User selects time period for time based alerts on goal progress
10. User selects create goal and confirms the creation at the systems prompting

User repeats steps 3-10 until satisfied with the management of their nutrition goals

Example Use Case: Nutrition Goals

SSD: 2.2


:User
: System
alt
[User wants to create a goal]
1: createGoal(field, value, isMax, hasProgressAlerts, timeAlerts)
1.1: goal added to nutrition goals menu
[User wants to edit goal]
2: editGoal(goal)
[User wants to delete goal]
3: delete(goal)
3.1: goal removed from nutrition goals display

ID UC 3.6 Cook Recipe
Scope Recipe system
Level User goal
Stakeholders and Interests

- **User:** Person who has stored recipes and items within the FoodPants digital pantry, wants to use stored items to make a recipe and log the results to their nutrition log.
- **System maintainer:** Person responsible for application execution. Wants to satisfy customer interests.

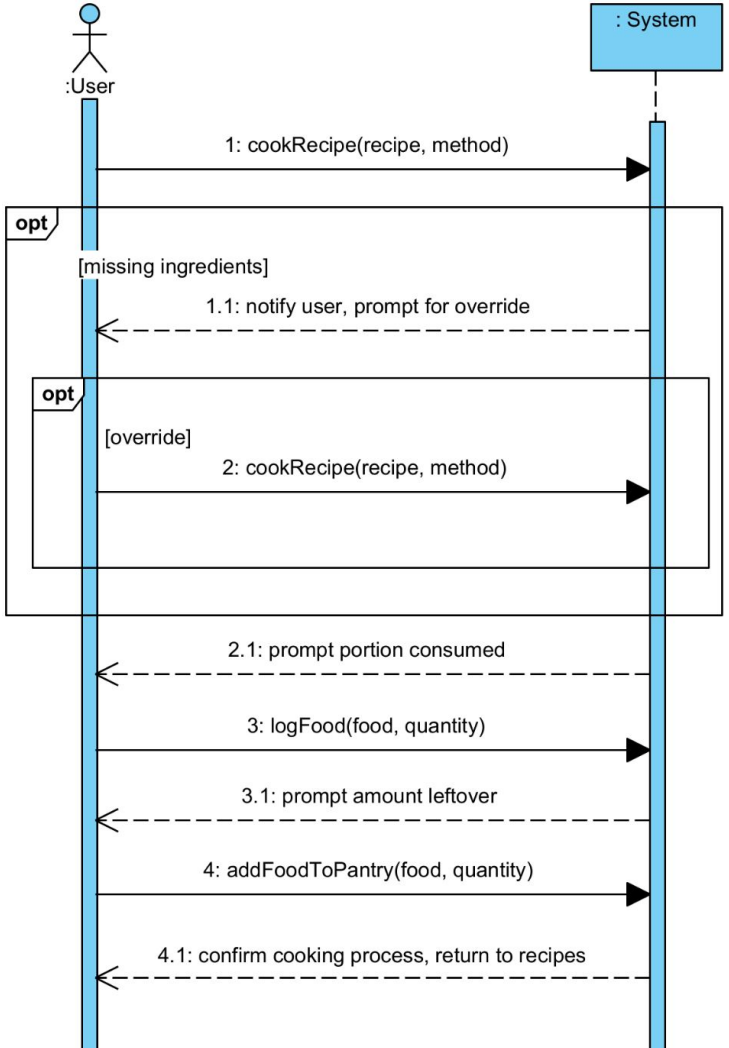
Precondition: The user has created the recipe within the recipe system, and has decided how much of the recipe they will eat (they can perform this step after cooking if necessary)
Postcondition: Used items will have been consumed from the user's pantry, and the recipe and nutritional info will be stored in the user's nutrition log.

Main success scenario:

1. The user navigates to the recipe page.
2. User selects desired recipe to produce.
3. User selects to cook the recipe.
4. The system confirms the user's consumption of the recipe.
5. The system removes items from the digital pantry that were used to make the recipe.
6. The User is prompted for how much of the recipe they consumed.
7. The amount of the recipe consumed is added to the nutrition log.
8. include (manageNutritionLog)

Example Use Case: Cook Recipe

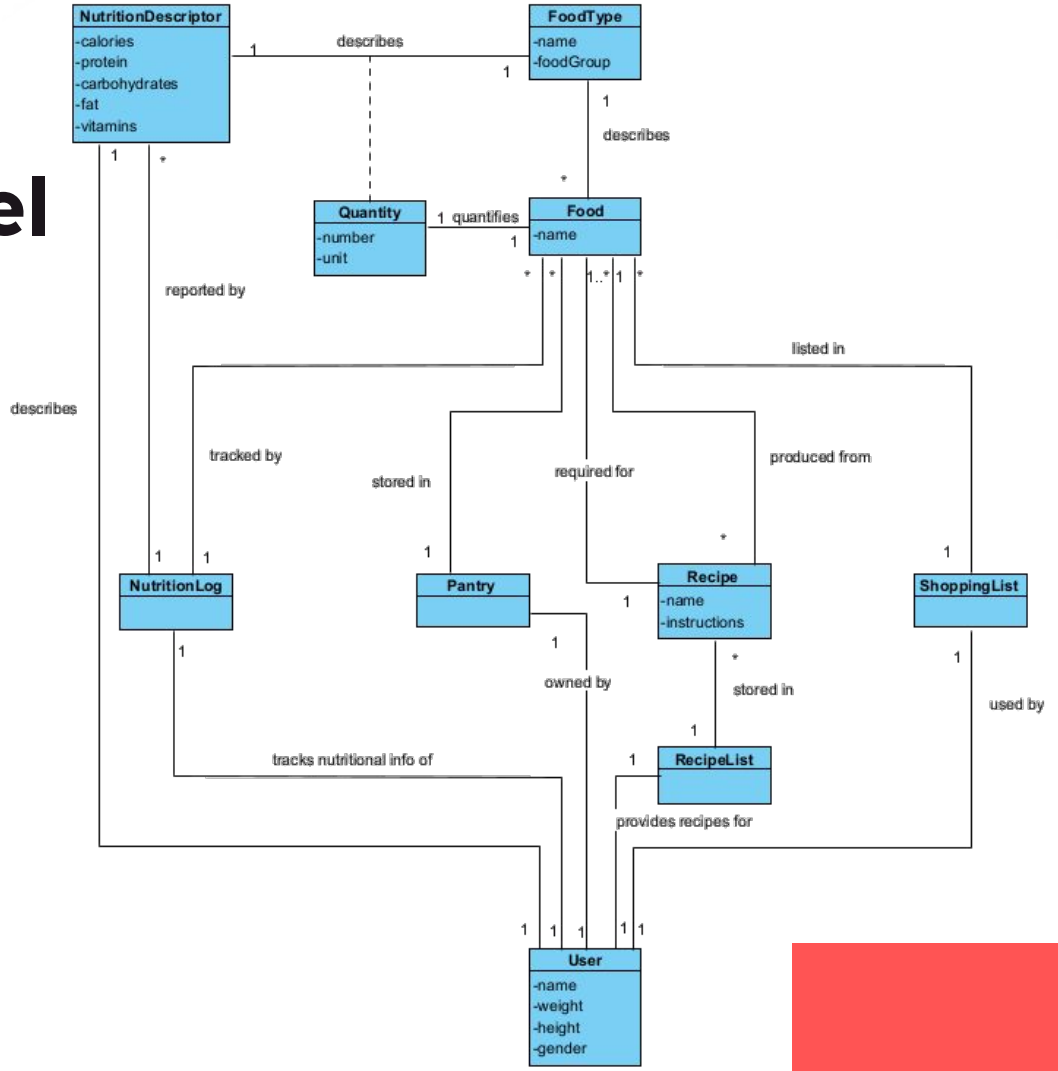
SSD: 3.6


:User
: System
1: cookRecipe(recipe, method)
opt
[missing ingredients]
1.1: notify user, prompt for override
opt
[override]
2: cookRecipe(recipe, method)
2.1: prompt portion consumed
3: logFood(food, quantity)
3.1: prompt amount leftover
4: addFoodToPantry(food, quantity)
4.1: confirm cooking process, return to recipes

Domain Model

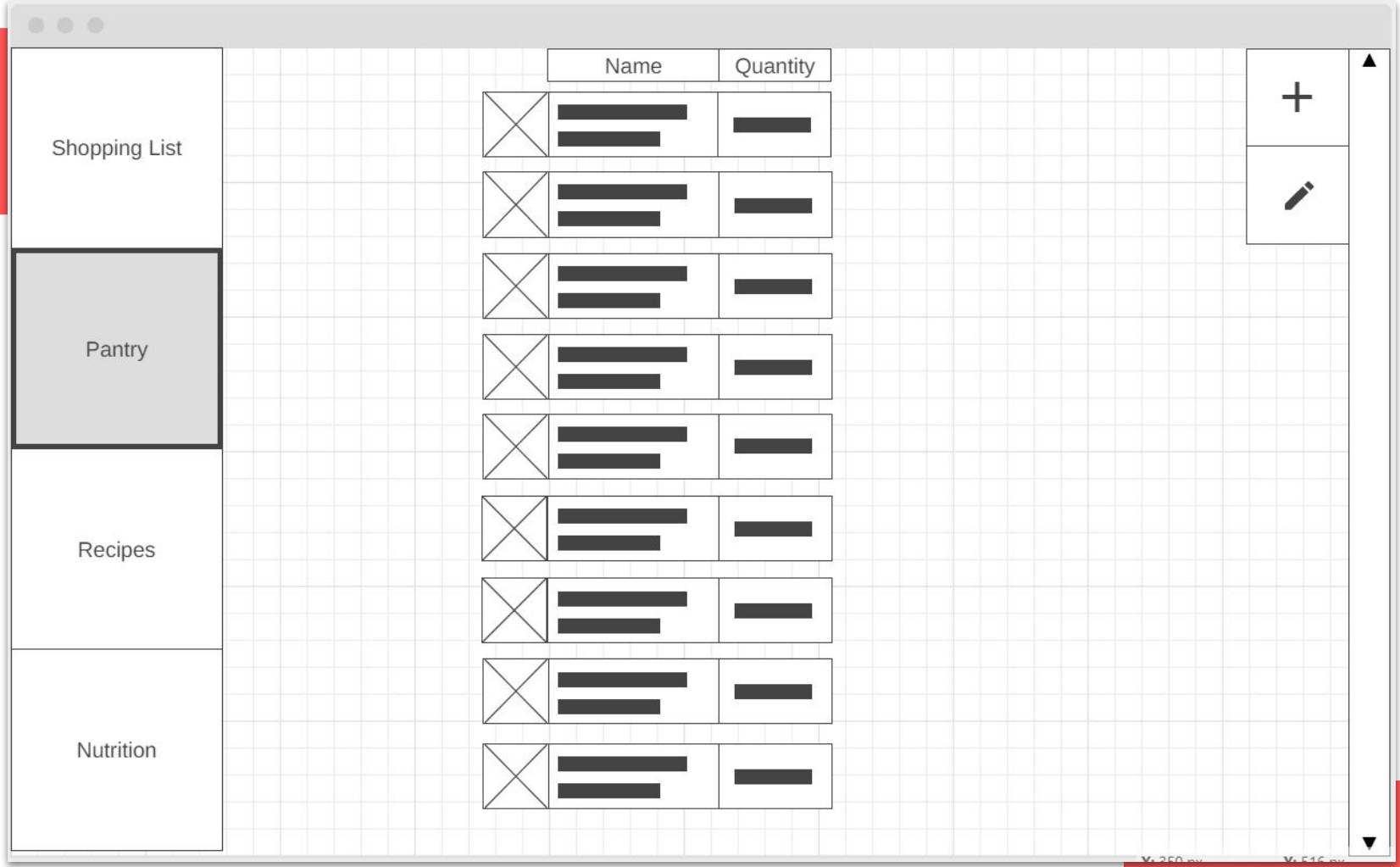
Moving to Concepts...

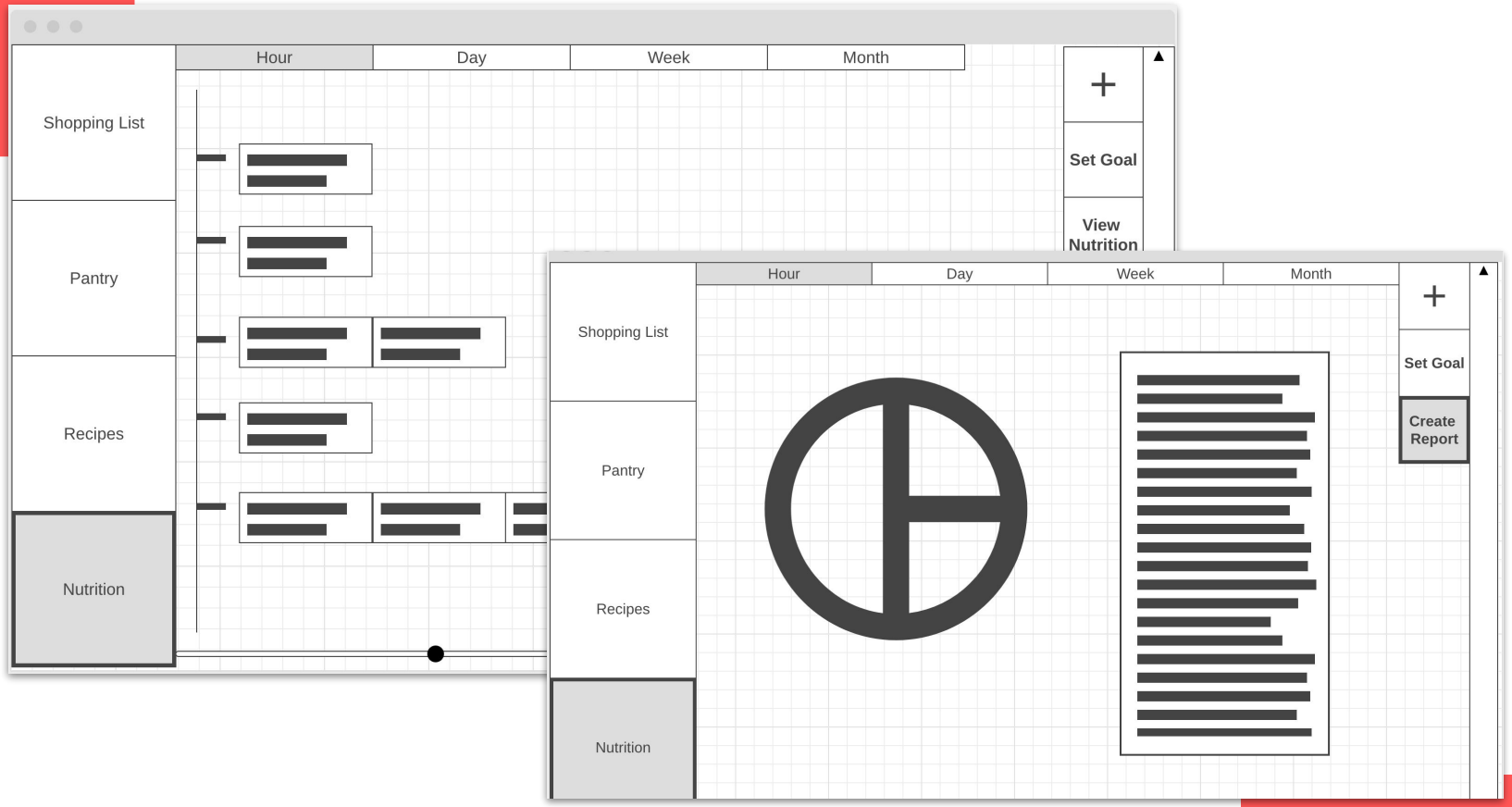
Domain Model

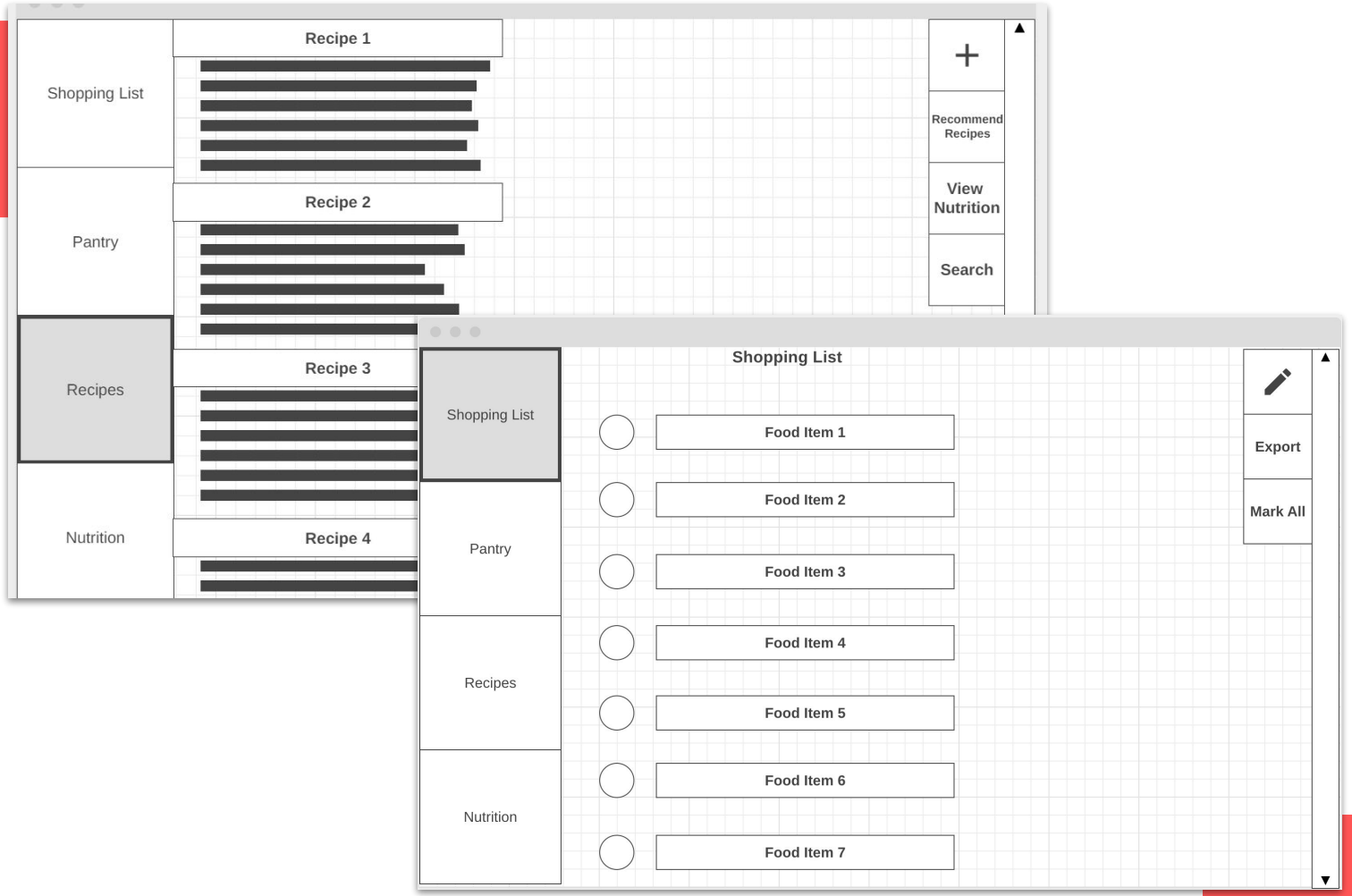


Wireframes

How will this thing look?!


Shopping List
Pantry
Recipes
Nutrition
Name
Quantity


Hour
Day
Week
Month
Shopping List
Pantry
Recipes
Nutrition
+
Set Goal
View Nutrition
Hour
Day
Week
Month
Shopping List
Pantry
Recipes
Nutrition
+
Set Goal
Create Report


Shopping List
Pantry
Recipes
Nutrition
Recipe 1
Recipe 2
Recipe 3
Recipe 4
Recommend Recipes
View Nutrition
Search
Shopping List
Shopping List
Pantry
Recipes
Nutrition
Food Item 1
Food Item 2
Food Item 3
Food Item 4
Food Item 5
Food Item 6
Food Item 7
Export
Mark All

Team Distribution

Daniel Luper

- Use Cases
- Domain Modeling
- Git

PJ Wallace

- Use Cases
- Requirement Analysis
- Wireframes

Kurt Wokoek

- Use Cases
- Traceability Matrix
- Reporting

Patrick Harris

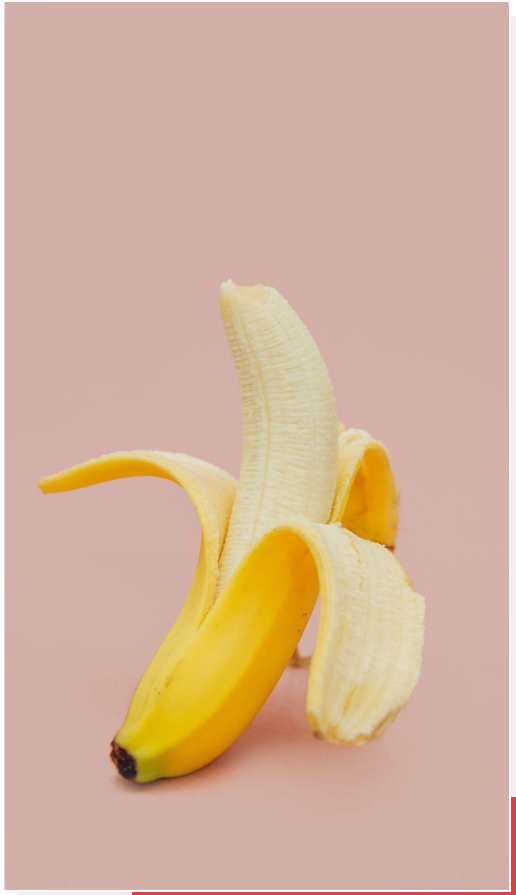
- Use Cases
- Issue Tracking
- System Operations

Luka Lelovic

- Use Cases
- Website
- Wireframes

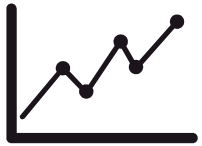
Austin Huizinga

- Use Cases
- Use Case Diagram
- Presentation

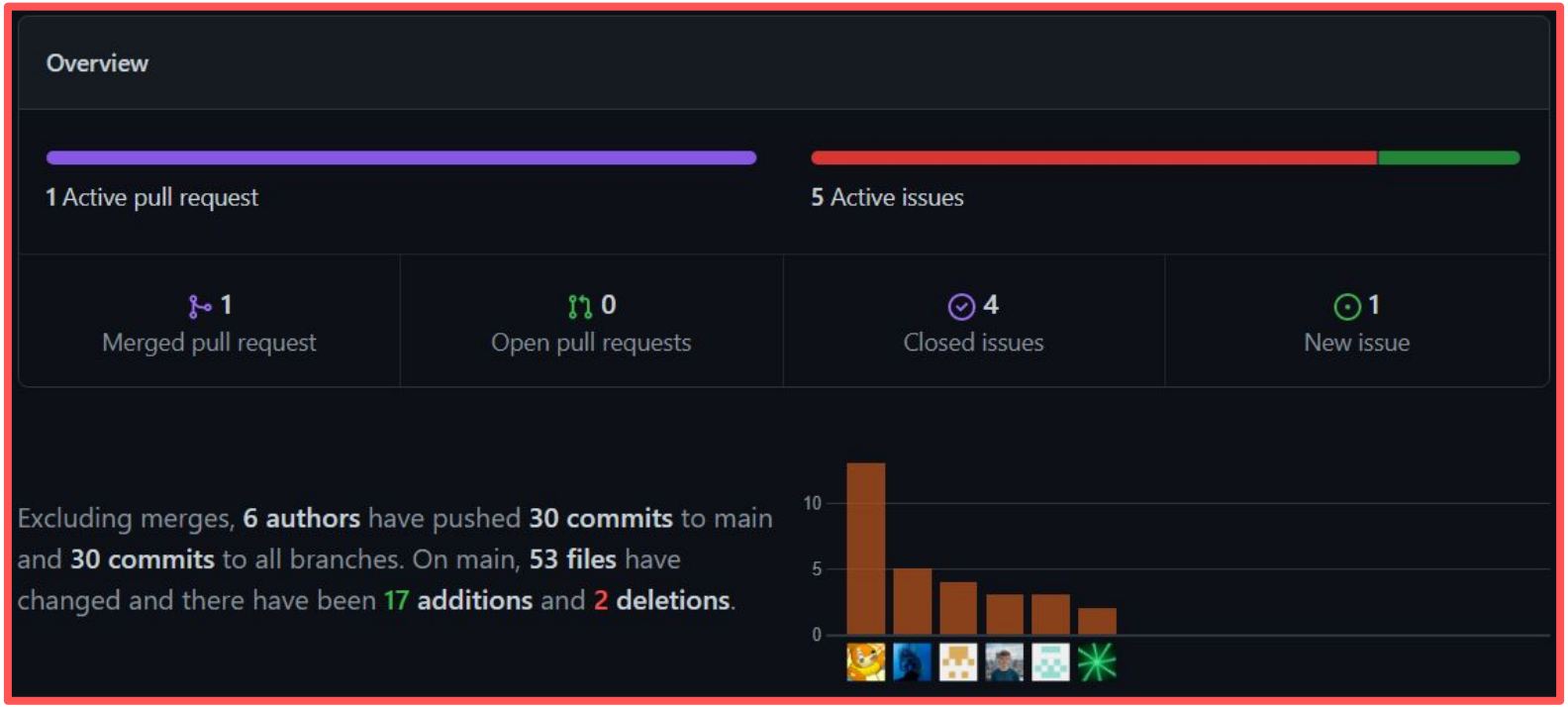


85 hours

Tracked via Chronos in Trello



Users	Σ	03 Thu	04 Fri	05 Sat	06 Sun	07 Mon	08 Tue	09 Wed	10 Thu	11 Fri	12 Sat	13 Sun	14 Mon	15 Tue	16 Wed	17 Thu	18 Fri	19 Sat	20 Sun	21 Mon	22 Tue	23 Wed
Austin_Huizinga1	18h 44m	30m			30m		30m					2h 0m					4h 0m	1h 30m	3h 0m		1h 30m	5h 14m
Daniel Luper	12h 2m																		8h 56m	40m	1h 0m	1h 24m
Kurt_Wokoek1	13h 10m						30m					2h 0m						3h 19m	2h 35m			4h 46m
Patrick_Harris3	13h 30m						30m					1h 30m				2h 30m	2h 50m			2h 30m		3h 40m
PJ_Wallace1	15h 0m						30m				1h 30m	2h 0m						5h 30m		4h 0m	30m	1h 0m
Luka_Lelovic1	13h 15m						1h 0m					1h 30m									2h 0m	8h 45m



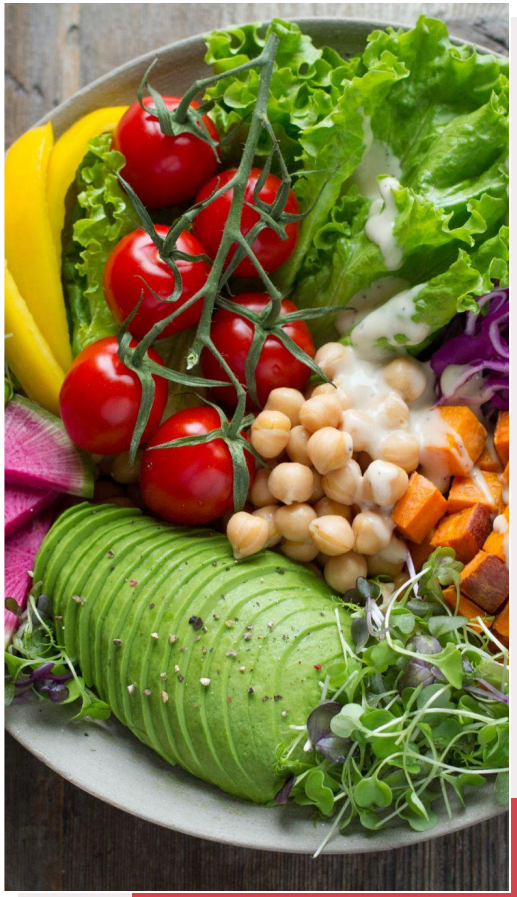
Git Insights

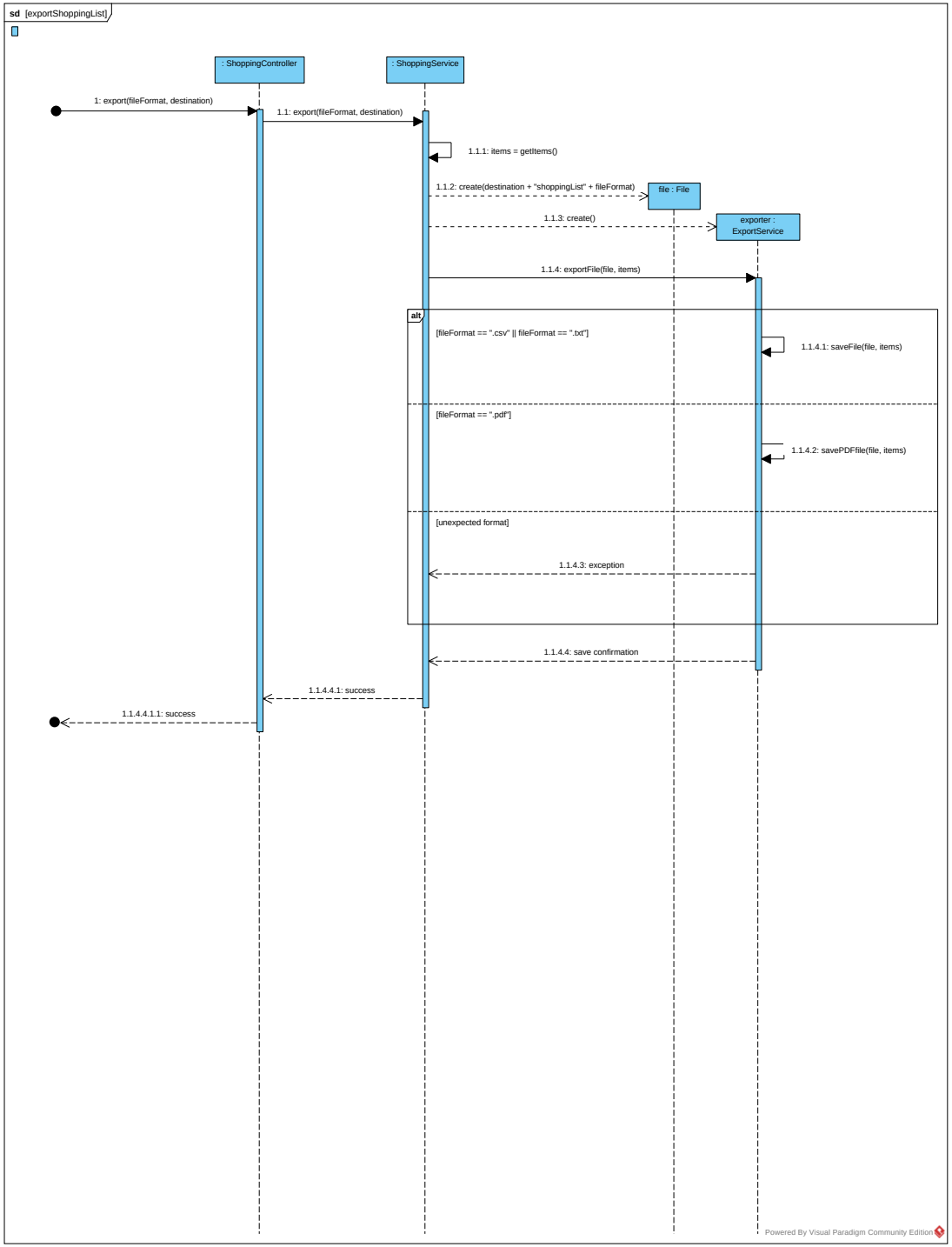
Q/A

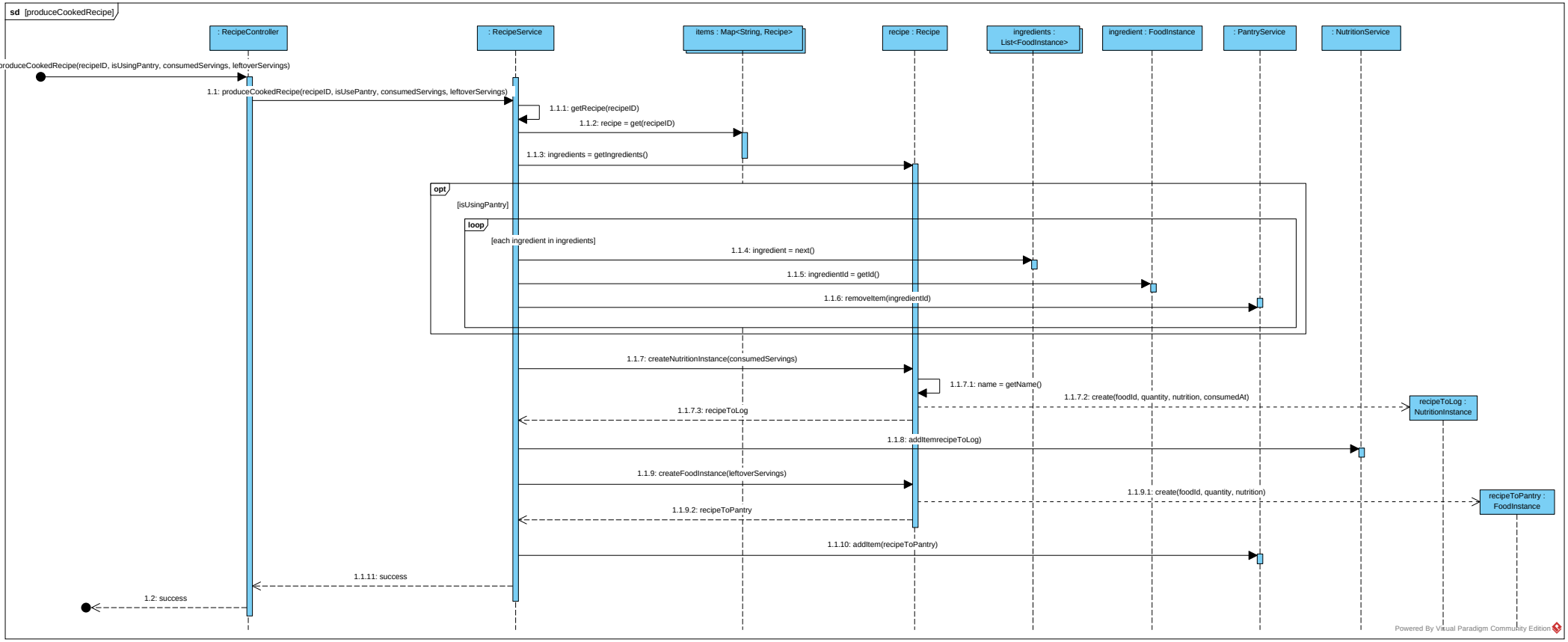
Credits

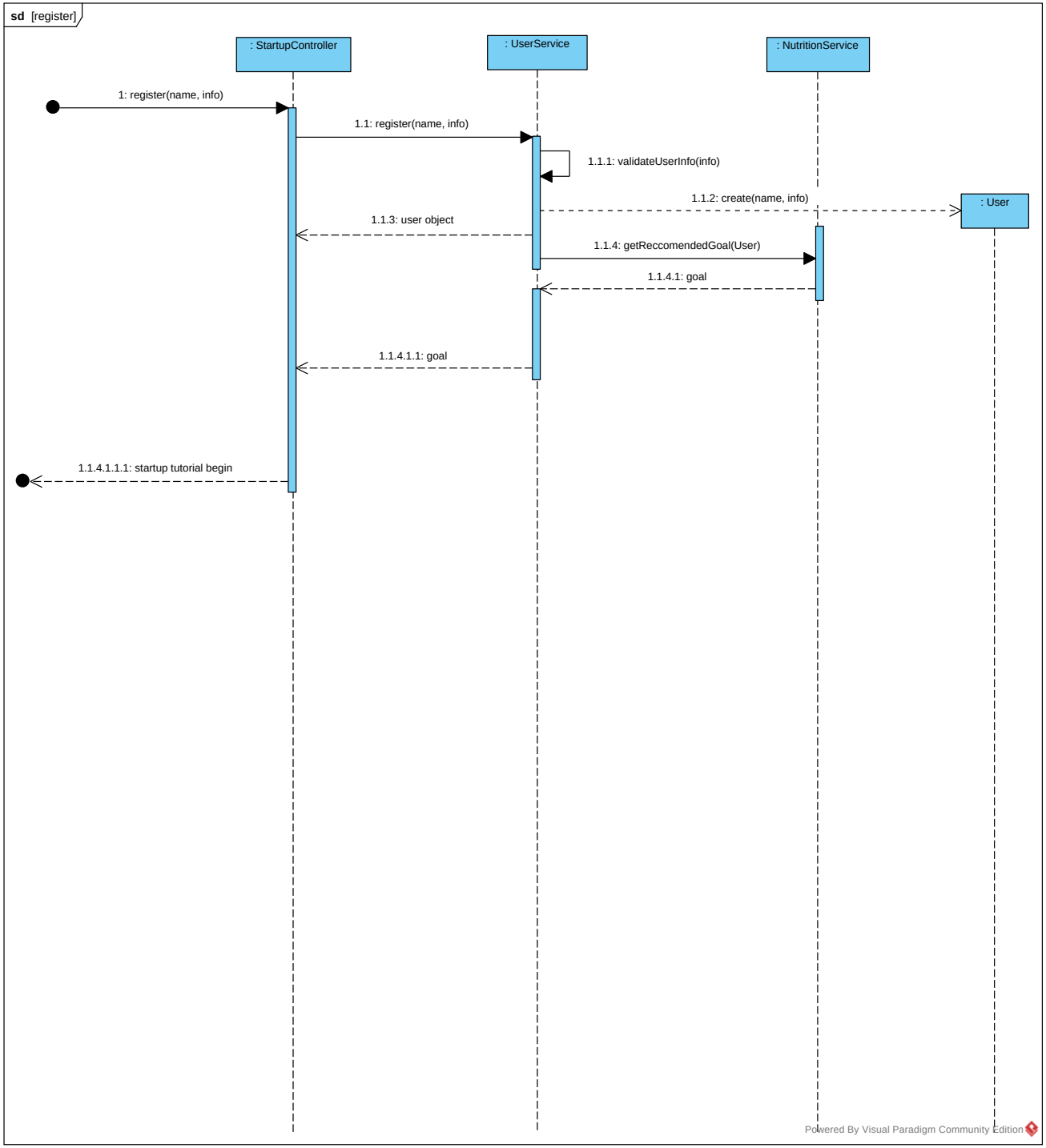
Special thanks to all the people who made and released these awesome resources for free:

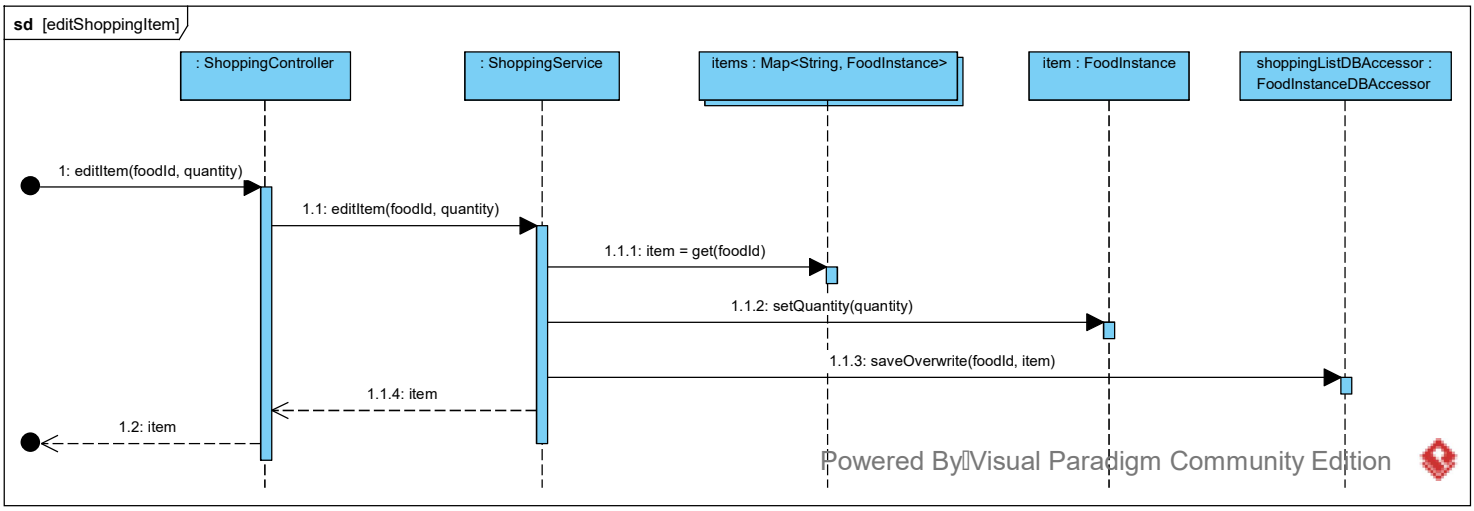
- Presentation template by SlidesCarnival
- Photographs by Unsplash

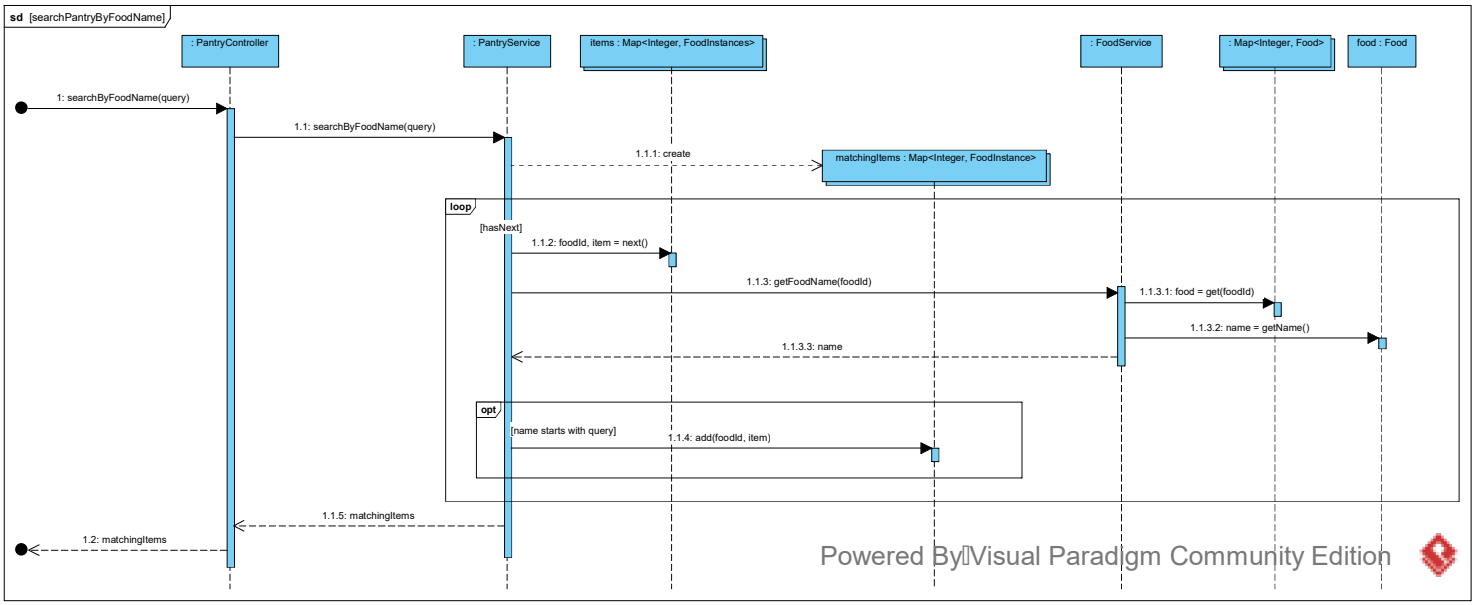


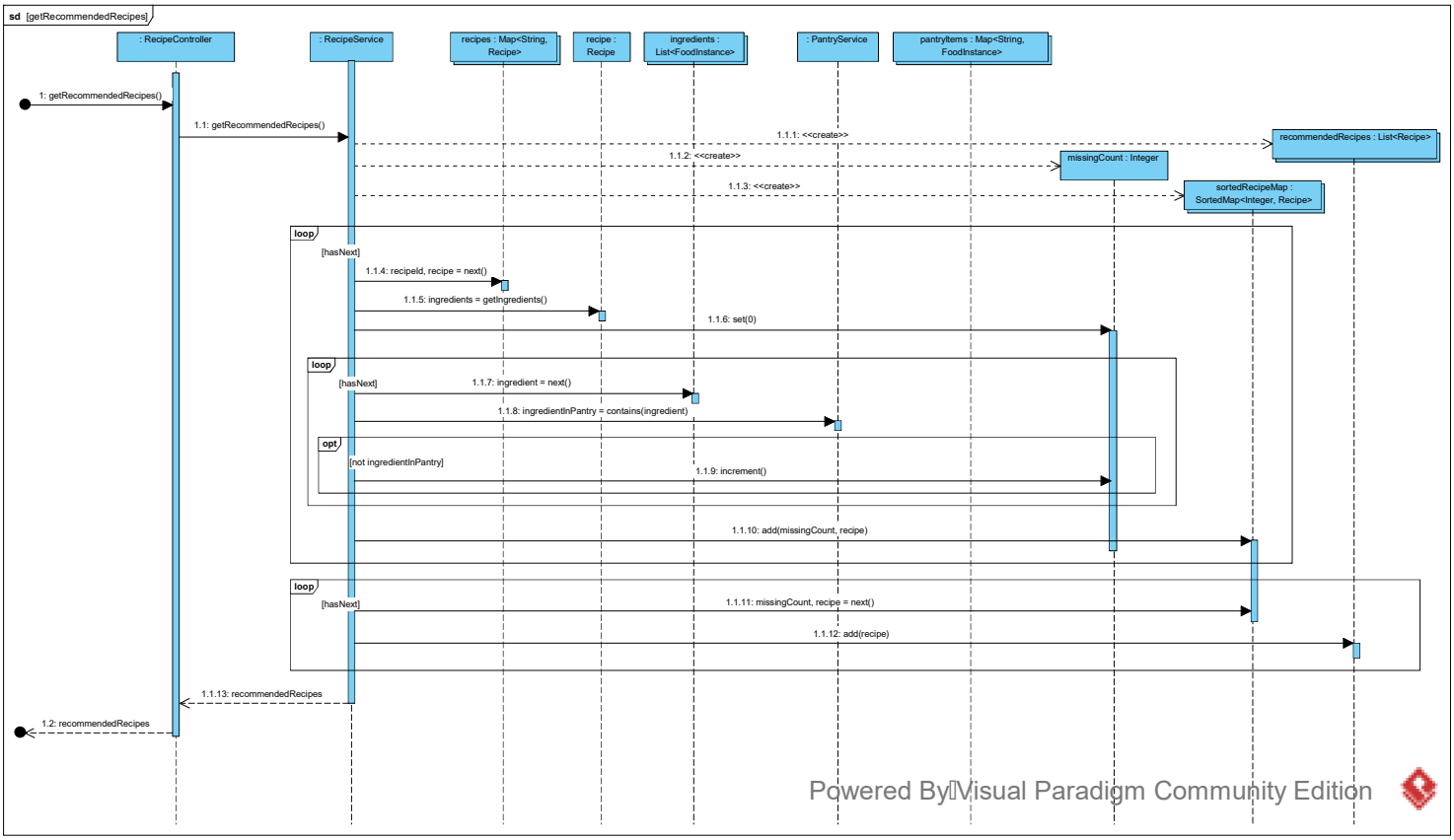

sd [exportShoppingList]
: ShoppingController
: ShoppingService
1: export(fileFormat, destination)
1.1: export(fileFormat, destination)
1.1.1: items = getItems()
1.1.2: create(destination + "shoppingList" + fileFormat)
file : File
1.1.3: create()
exporter : ExportService
1.1.4: exportFile(file, items)
alt
[fileFormat == ".csv" || fileFormat == ".txt"]
1.1.4.1: saveFile(file, items)
[fileFormat == ".pdf"]
1.1.4.2: savePDFfile(file, items)
[unexpected format]
1.1.4.3: exception
1.1.4.4: save confirmation
1.1.4.4.1: success
1.1.4.4.1.1: success
Powered By Visual Paradigm Community Edition

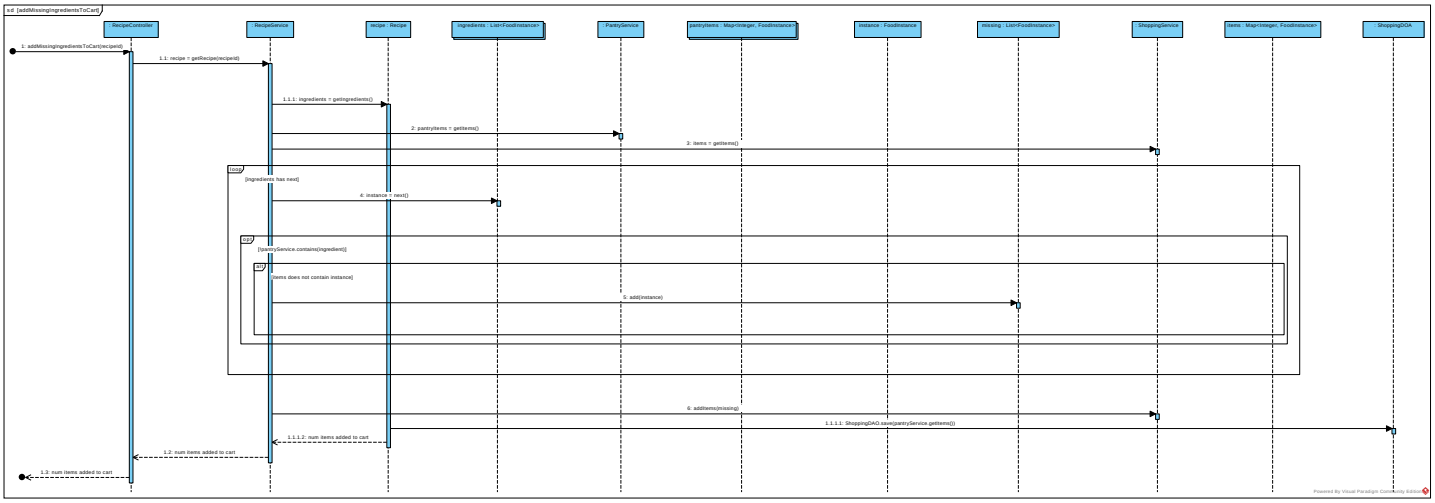

sd [produceCookedRecipe]
: RecipeController
: RecipeService
items : Map<String, Recipe>
recipe : Recipe
ingredients : List<FoodInstance>
ingredient : FoodInstance
: PantryService
: NutritionService
1: produceCookedRecipe(recipeID, isUsingPantry, consumedServings, leftoverServings)
1.1: produceCookedRecipe(recipeID, isUsePantry, consumedServings, leftoverServings)
1.1.1: getRecipe(recipeID)
1.1.2: recipe = get(recipeID)
1.1.3: ingredients = getIngredients()
opt
[isUsingPantry]
loop
[each ingredient in ingredients]
1.1.4: ingredient = next()
1.1.5: ingredientId = getId()
1.1.6: removeItem(ingredientId)
1.1.7: createNutritionInstance(consumedServings)
1.1.7.1: name = getName()
1.1.7.2: create(foodId, quantity, nutrition, consumedAt)
recipeToLog : NutritionInstance
1.1.7.3: recipeToLog
1.1.8: addItemrecipeToLog)
1.1.9: createFoodInstance(leftoverServings)
1.1.9.1: create(foodId, quantity, nutrition)
recipeToPantry : FoodInstance
1.1.9.2: recipeToPantry
1.1.10: addItem(recipeToPantry)
1.1.11: success
1.2: success
Powered By Visual Paradigm Community Edition

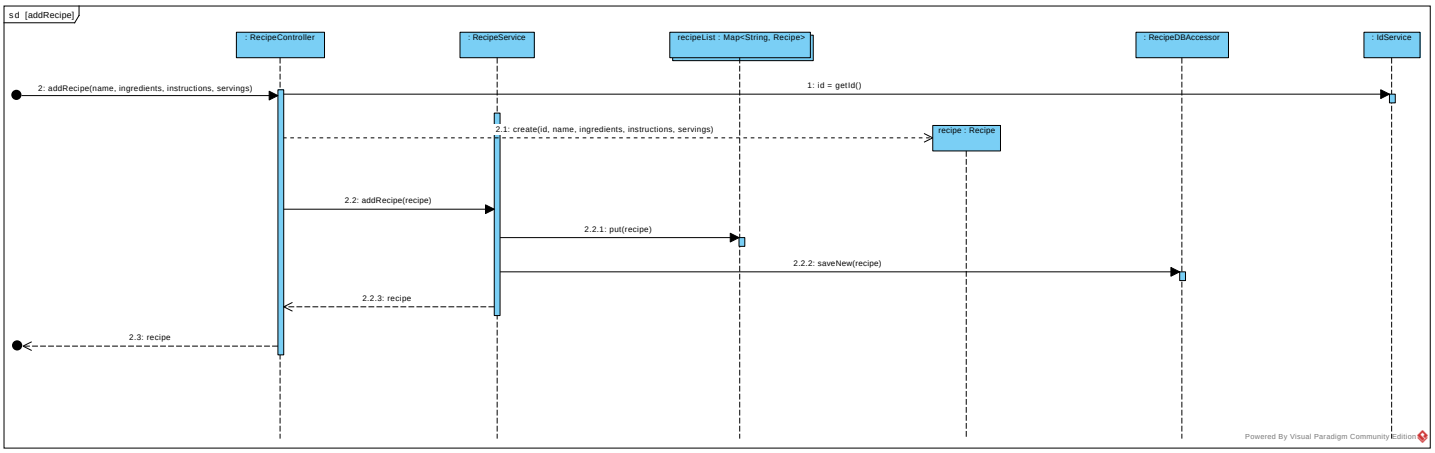

sd [register]
: StartupController
: UserService
: NutritionService
: User
1: register(name, info)
1.1: register(name, info)
1.1.1: validateUserInfo(info)
1.1.2: create(name, info)
1.1.3: user object
1.1.4: getReccomendedGoal(User)
1.1.4.1: goal
1.1.4.1.1: goal
1.1.4.1.1.1: startup tutorial begin
Powered By Visual Paradigm Community Edition

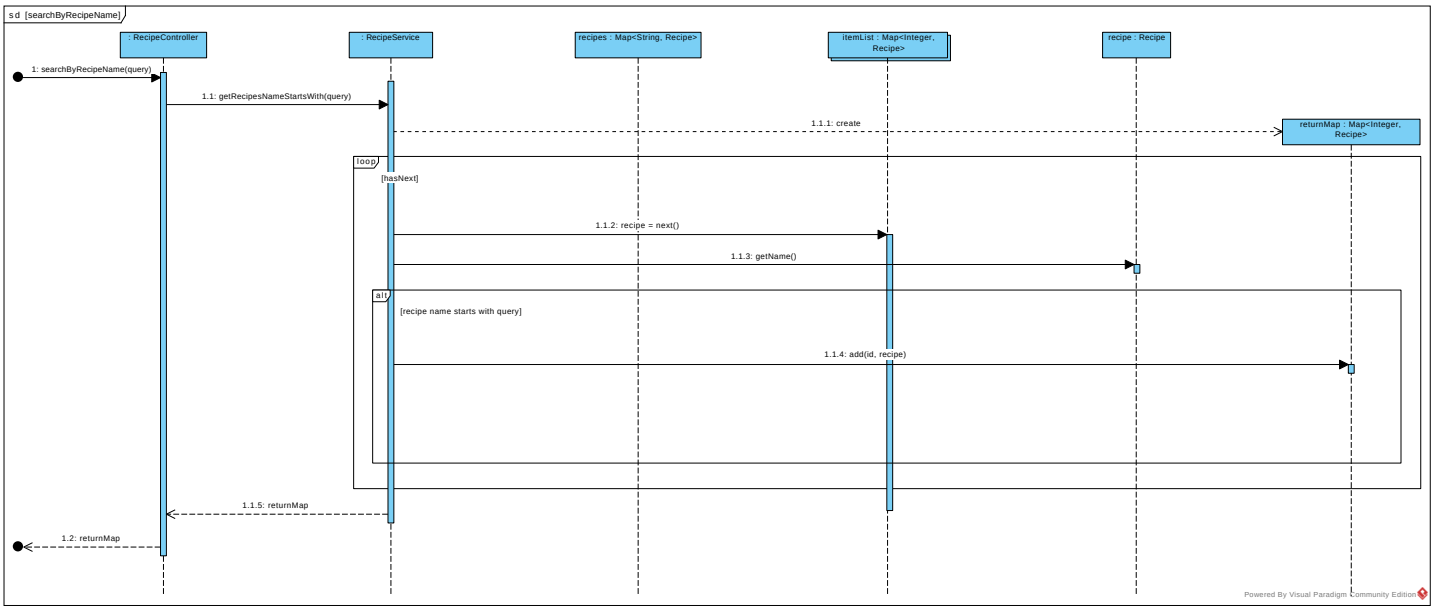

sd [editShoppingItem]
: ShoppingController
: ShoppingService
items : Map<String, FoodInstance>
item : FoodInstance
shoppingListDBAccessor : FoodInstanceDBAccessor
1: editItem(foodId, quantity)
1.1: editItem(foodId, quantity)
1.1.1: item = get(foodId)
1.1.2: setQuantity(quantity)
1.1.3: saveOverwrite(foodId, item)
1.1.4: item
1.2: item
Powered By Visual Paradigm Community Edition

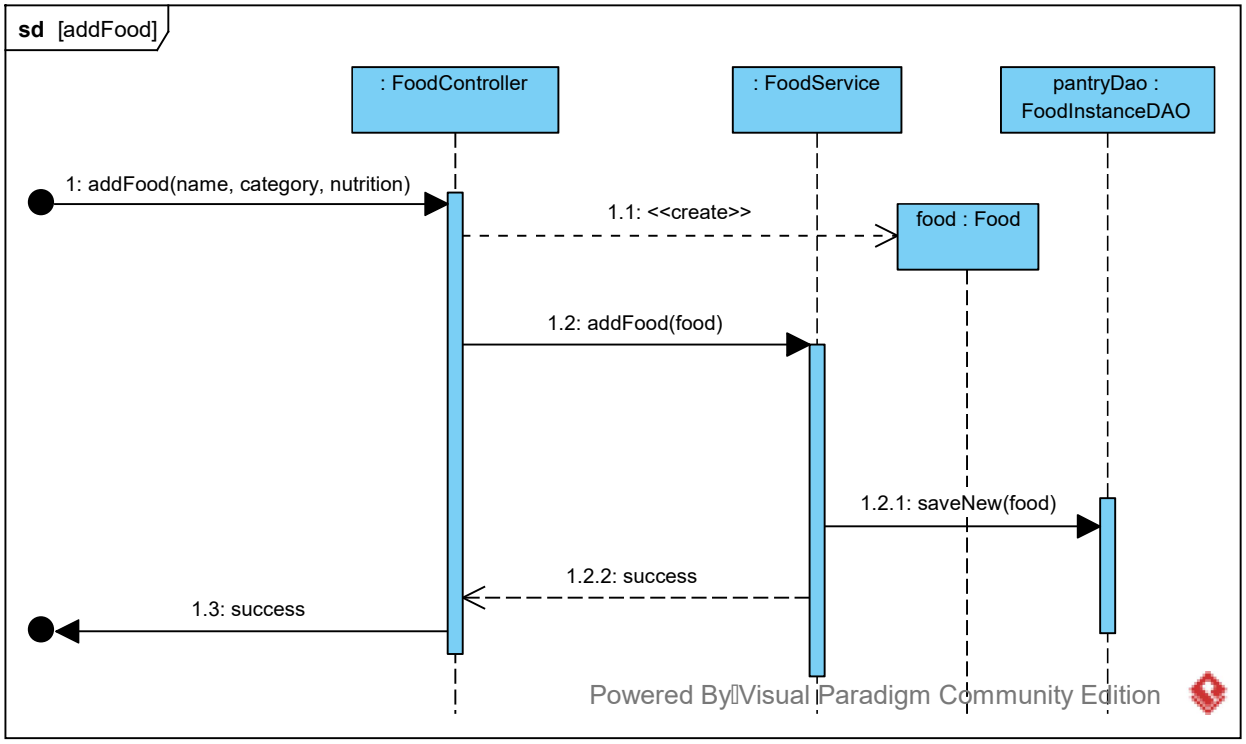

sd [searchPantryByFoodName]
: PantryController
: PantryService
items : Map<Integer, FoodInstances>
: FoodService
: Map<Integer, Food>
food : Food
1: searchByFoodName(query)
1.1: searchByFoodName(query)
1.1.1: create
matchingItems : Map<Integer, FoodInstance>
loop
[hasNext]
1.1.2: foodId, item = next()
1.1.3: getFoodName(foodId)
1.1.3.1: food = get(foodId)
1.1.3.2: name = getName()
1.1.3.3: name
opt
[name starts with query]
1.1.4: add(foodId, item)
1.1.5: matchingItems
1.2: matchingItems
Powered By Visual Paradigm Community Edition

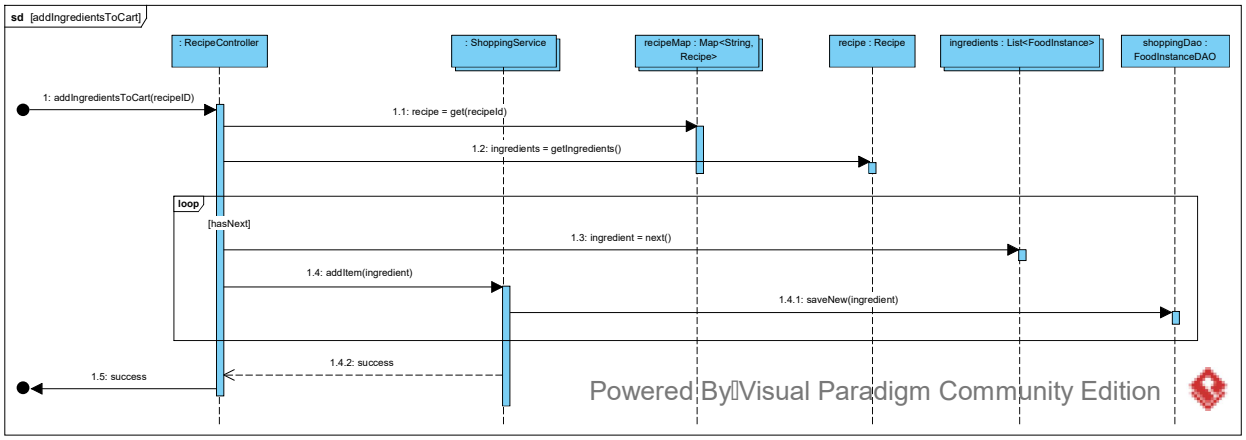

sd [getRecommendedRecipes]
: RecipeController
: RecipeService
recipes : Map<String, Recipe>
recipe : Recipe
ingredients : List<FoodInstance>
: PantryService
pantryItems : Map<String, FoodInstance>
1: getRecommendedRecipes()
1.1: getRecommendedRecipes()
1.1.1: <<create>>
recommendedRecipes : List<Recipe>
1.1.2: <<create>>
missingCount : Integer
1.1.3: <<create>>
sortedRecipeMap : SortedMap<Integer, Recipe>
loop
[hasNext]
1.1.4: recipeId, recipe = next()
1.1.5: ingredients = getIngredients()
1.1.6: set(0)
loop
[hasNext]
1.1.7: ingredient = next()
1.1.8: ingredientInPantry = contains(ingredient)
opt
[not ingredientInPantry]
1.1.9: increment()
1.1.10: add(missingCount, recipe)
loop
[hasNext]
1.1.11: missingCount, recipe = next()
1.1.12: add(recipe)
1.1.13: recommendedRecipes
1.2: recommendedRecipes
Powered By Visual Paradigm Community Edition

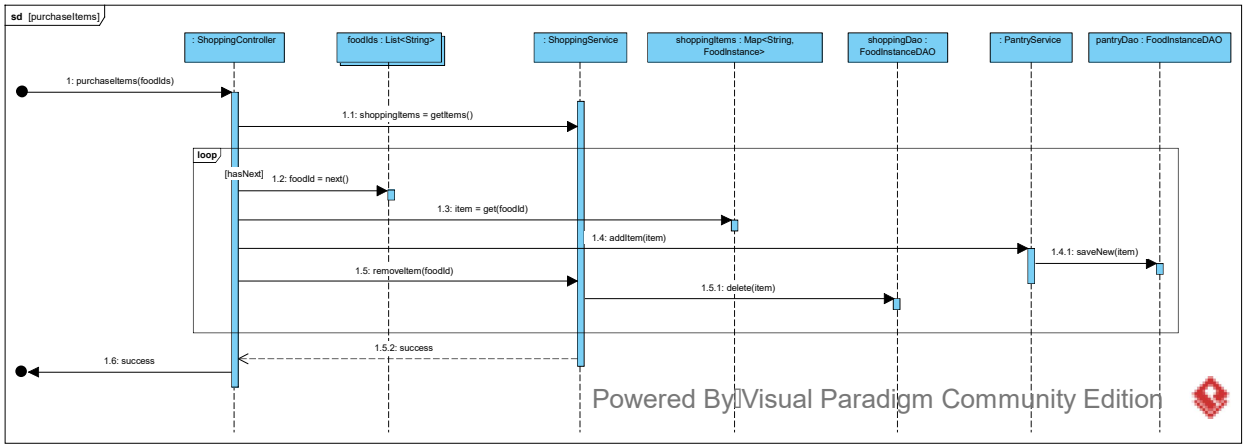

sd [addMissingIngredientsToCart()]
: RecipeController
: RecipeService
recipe : Recipe
ingredients : List<FoodInstance>
: PantryService
pantryItems : Map<Integer, FoodInstance>
instance : FoodInstance
missing : List<FoodInstance>
: ShoppingService
items : Map<Integer, FoodInstance>
: ShoppingDOA
1: addMissingIngredientsToCart(recipeId)
1.1: recipe = getRecipe(recipeId)
1.1.1: ingredients = getIngredients()
2: pantryItems = getItems()
3: items = getItems()
loop
[ingredients has next]
4: instance = next()
alt
[!pantryService.contains(ingredient)]
alt
[items does not contain instance]
5: add(instance)
6: addItems(missing)
1.1.1.1: ShoppingDAO.save(pantryService.getItems())
1.1.1.2: num items added to cart
1.2: num items added to cart
1.3: num items added to cart
Powered By Visual Paradigm Community Edition

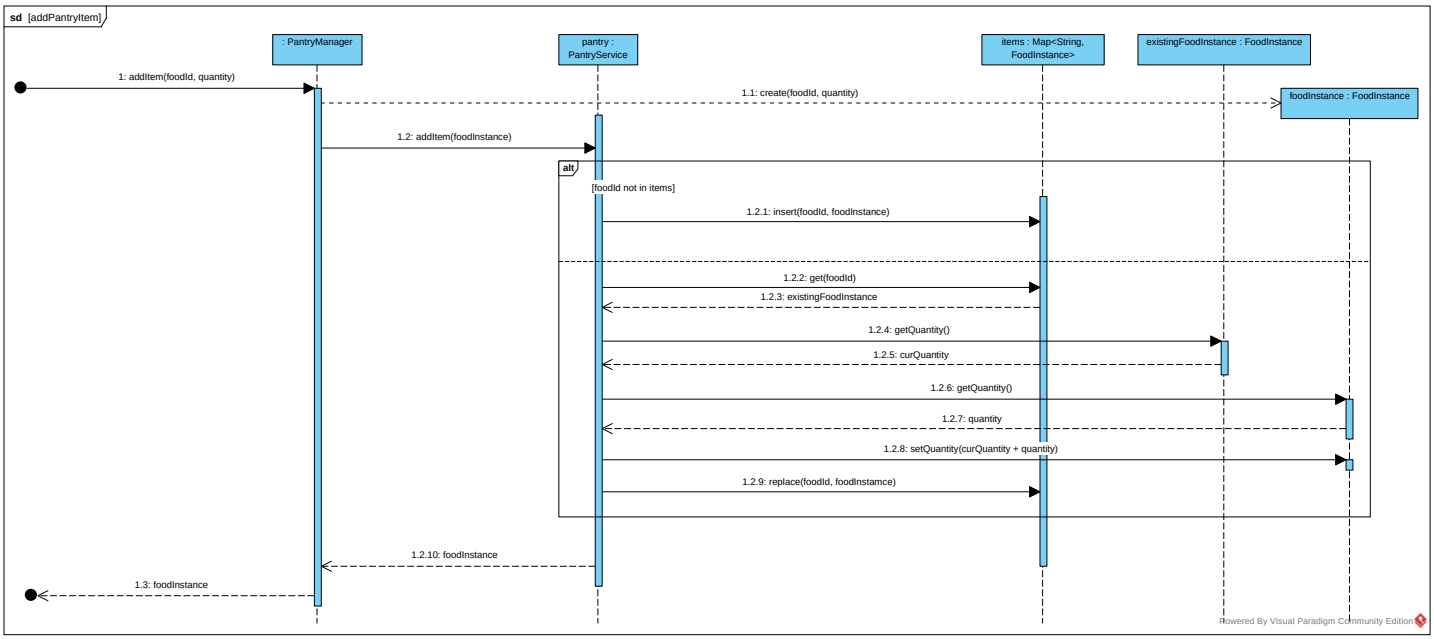

sd [addRecipe]
: RecipeController
: RecipeService
recipeList : Map<String, Recipe>
: RecipeDBAccessor
: IdService
2: addRecipe(name, ingredients, instructions, servings)
1: id = getId()
2.1: create(id, name, ingredients, instructions, servings)
recipe : Recipe
2.2: addRecipe(recipe)
2.2.1: put(recipe)
2.2.2: saveNew(recipe)
2.2.3: recipe
2.3: recipe
Powered By Visual Paradigm Community Edition

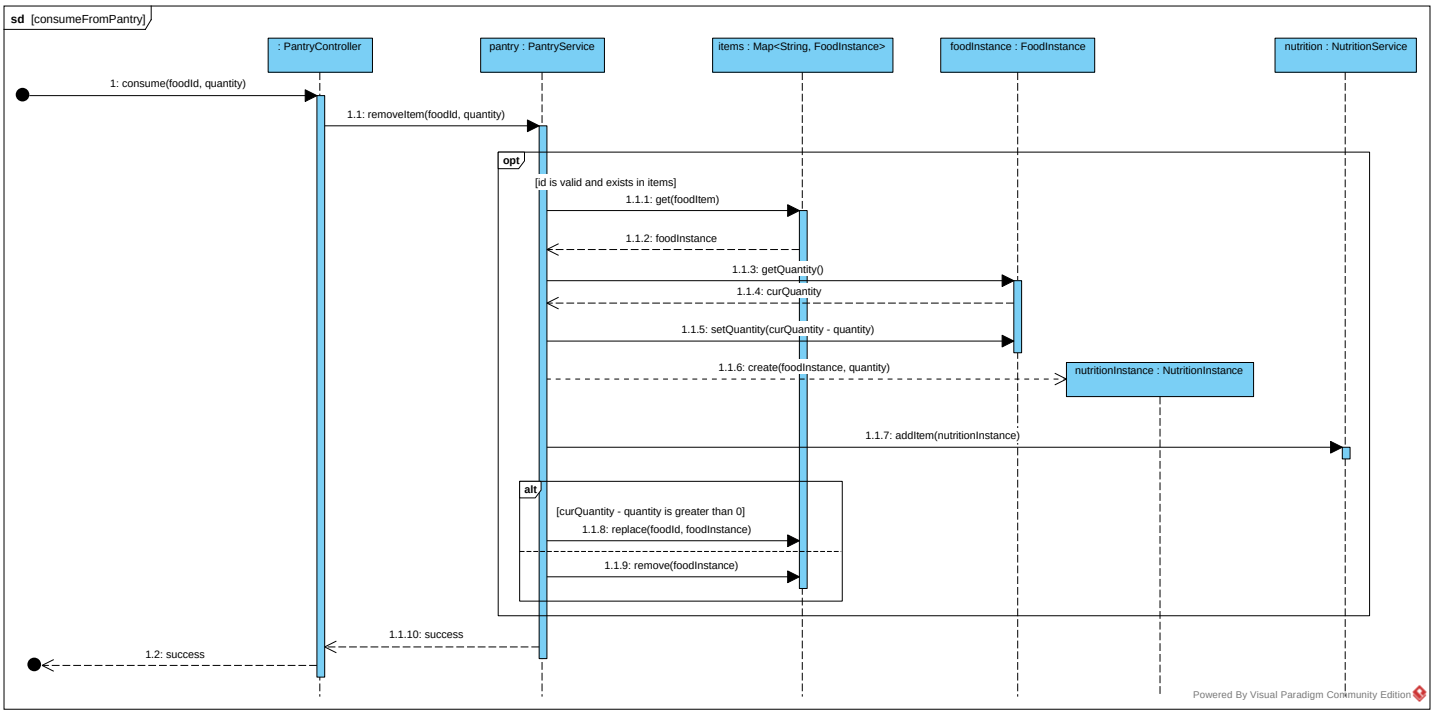

sd [searchByRecipeName]
: RecipeController
: RecipeService
recipes : Map<String, Recipe>
itemList : Map<Integer, Recipe>
recipe : Recipe
returnMap : Map<Integer, Recipe>
1: searchByRecipeName(query)
1.1: getRecipesNameStartsWith(query)
1.1.1: create
loop
[hasNext]
1.1.2: recipe = next()
1.1.3: getName()
alt
[recipe name starts with query]
1.1.4: add(id, recipe)
1.1.5: returnMap
1.2: returnMap
Powered By Visual Paradigm Community Edition

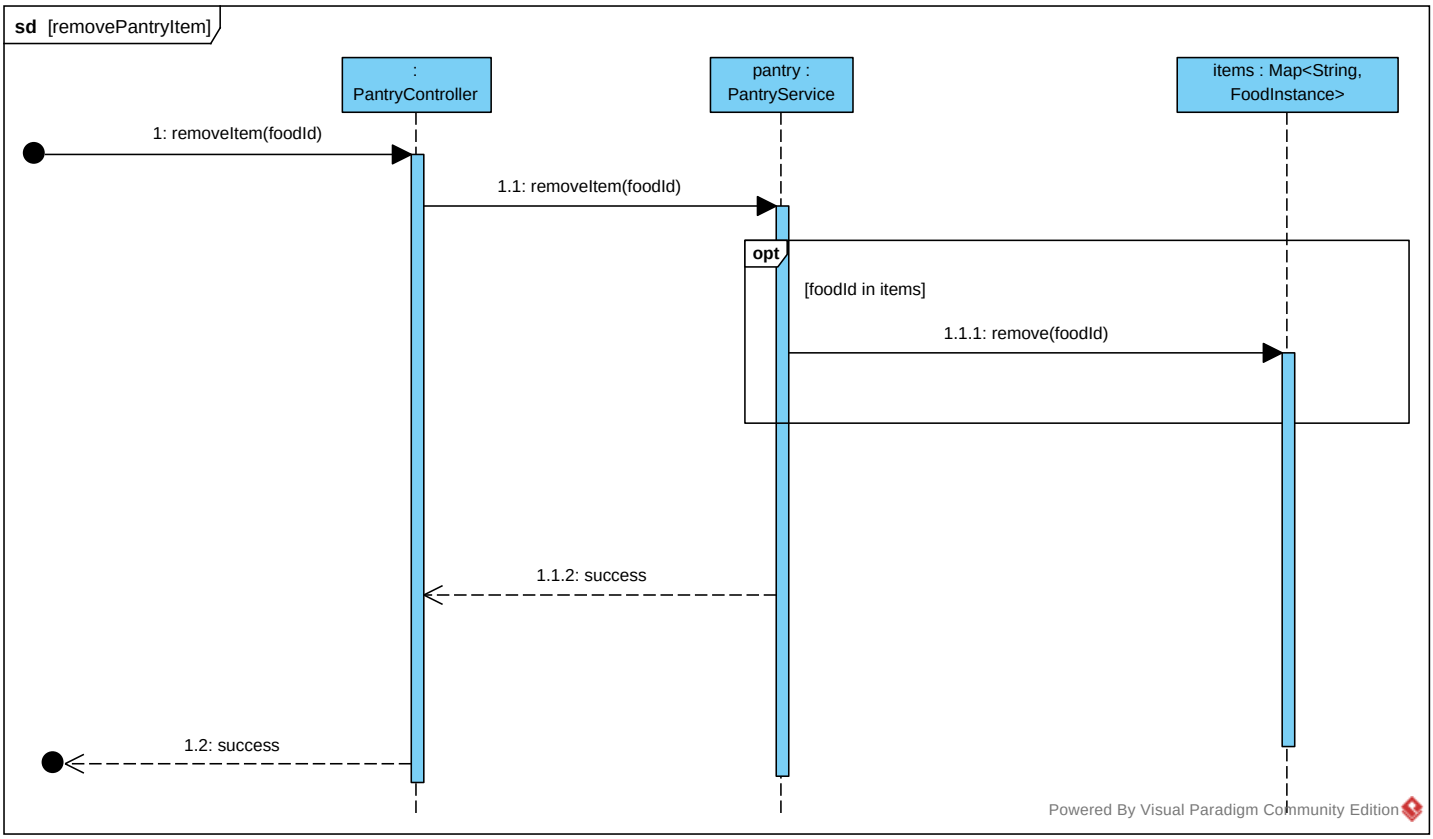

sd [addFood]
: FoodController
: FoodService
pantryDao : FoodInstanceDAO
1: addFood(name, category, nutrition)
1.1: <<create>>
food : Food
1.2: addFood(food)
1.2.1: saveNew(food)
1.2.2: success
1.3: success
Powered By Visual Paradigm Community Edition

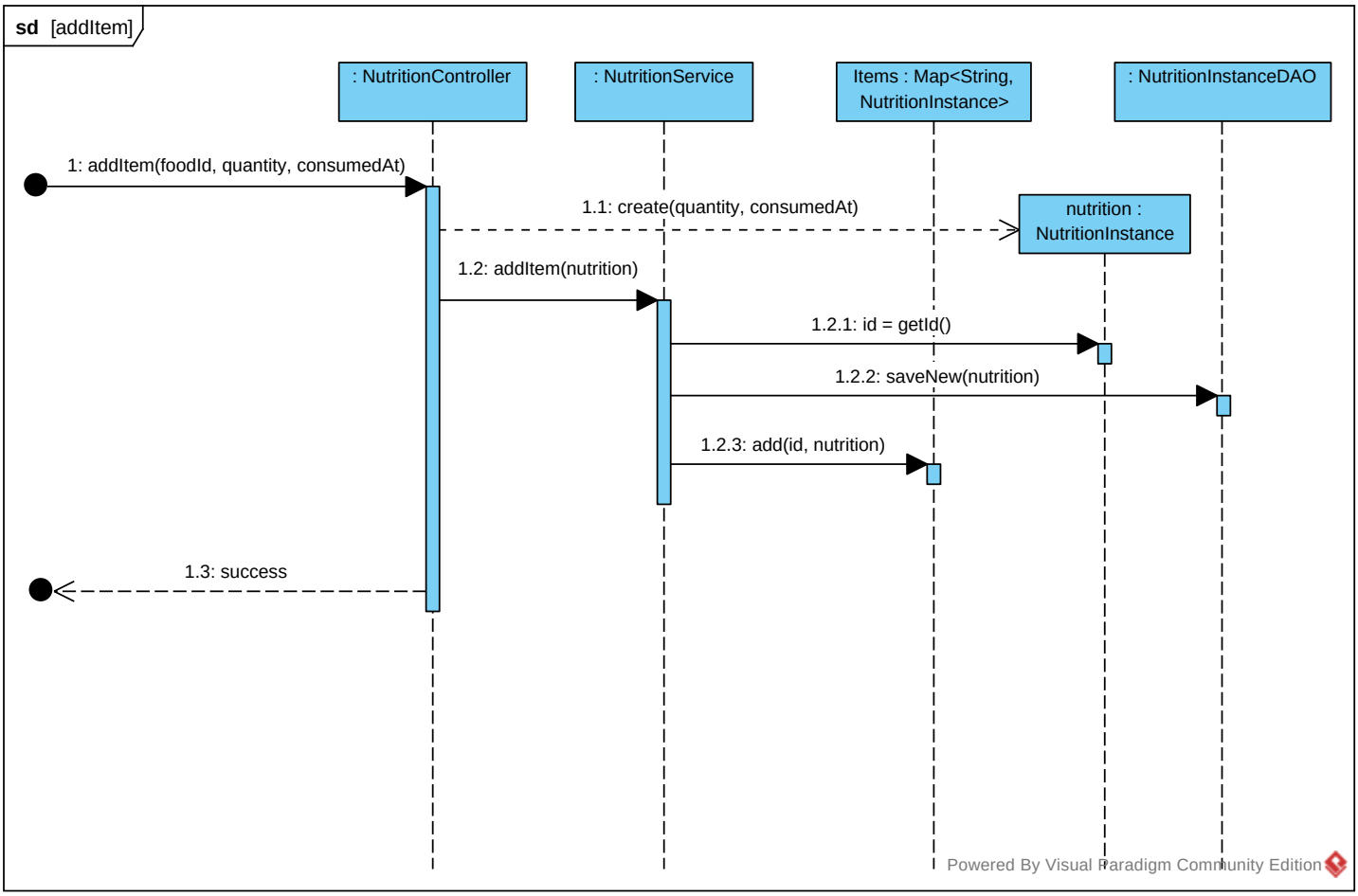

sd [addIngredientsToCart]
: RecipeController
: ShoppingService
recipeMap : Map<String, Recipe>
recipe : Recipe
ingredients : List<FoodInstance>
shoppingDao : FoodInstanceDAO
1: addIngredientsToCart(recipeID)
1.1: recipe = get(recipeId)
1.2: ingredients = getIngredients()
loop
[hasNext]
1.3: ingredient = next()
1.4: addItem(ingredient)
1.4.1: saveNew(ingredient)
1.4.2: success
1.5: success
Powered By Visual Paradigm Community Edition

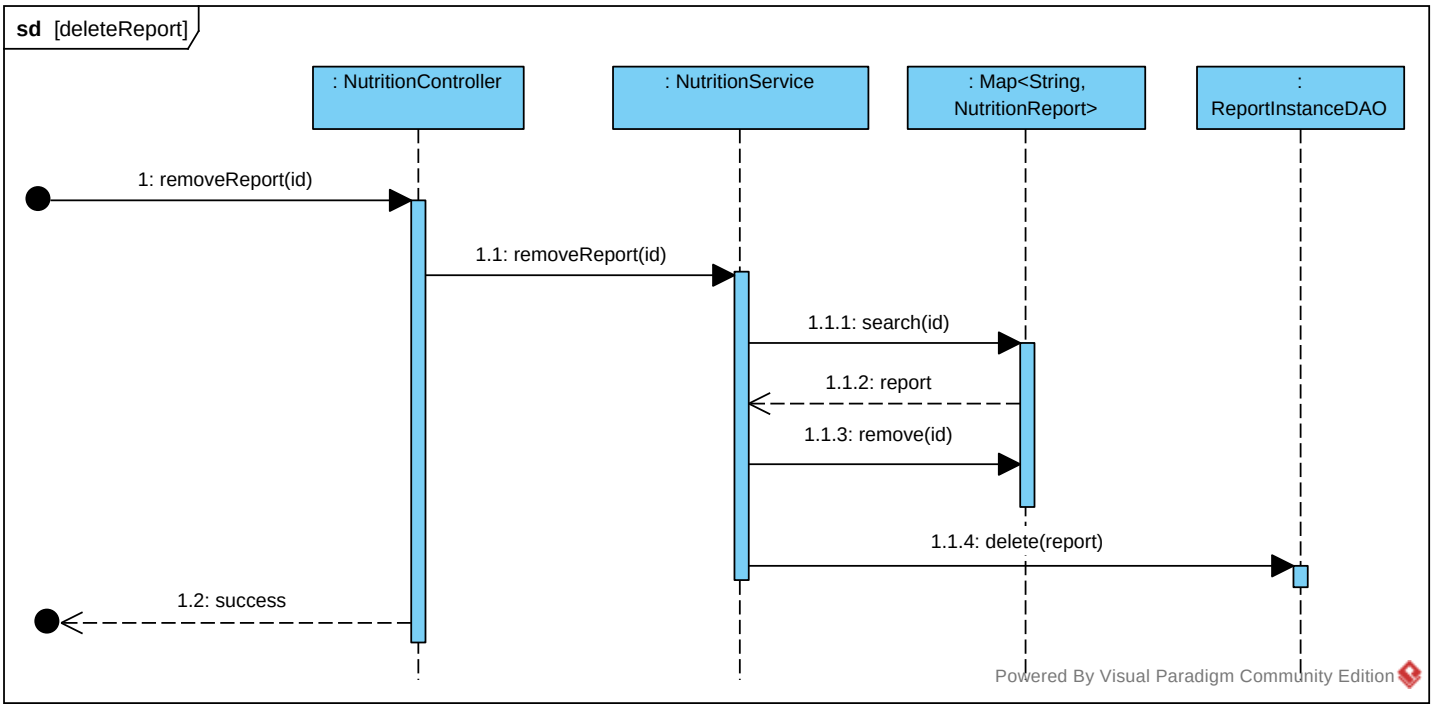

sd [purchaseItems]
: ShoppingController
foodIds : List<String>
: ShoppingService
shoppingItems : Map<String, FoodInstance>
shoppingDao : FoodInstanceDAO
: PantryService
pantryDao : FoodInstanceDAO
1: purchaseItems(foodIds)
1.1: shoppingItems = getItems()
loop
[hasNext]
1.2: foodId = next()
1.3: item = get(foodId)
1.4: addItem(item)
1.4.1: saveNew(item)
1.5: removeItem(foodId)
1.5.1: delete(item)
1.5.2: success
1.6: success
Powered By Visual Paradigm Community Edition

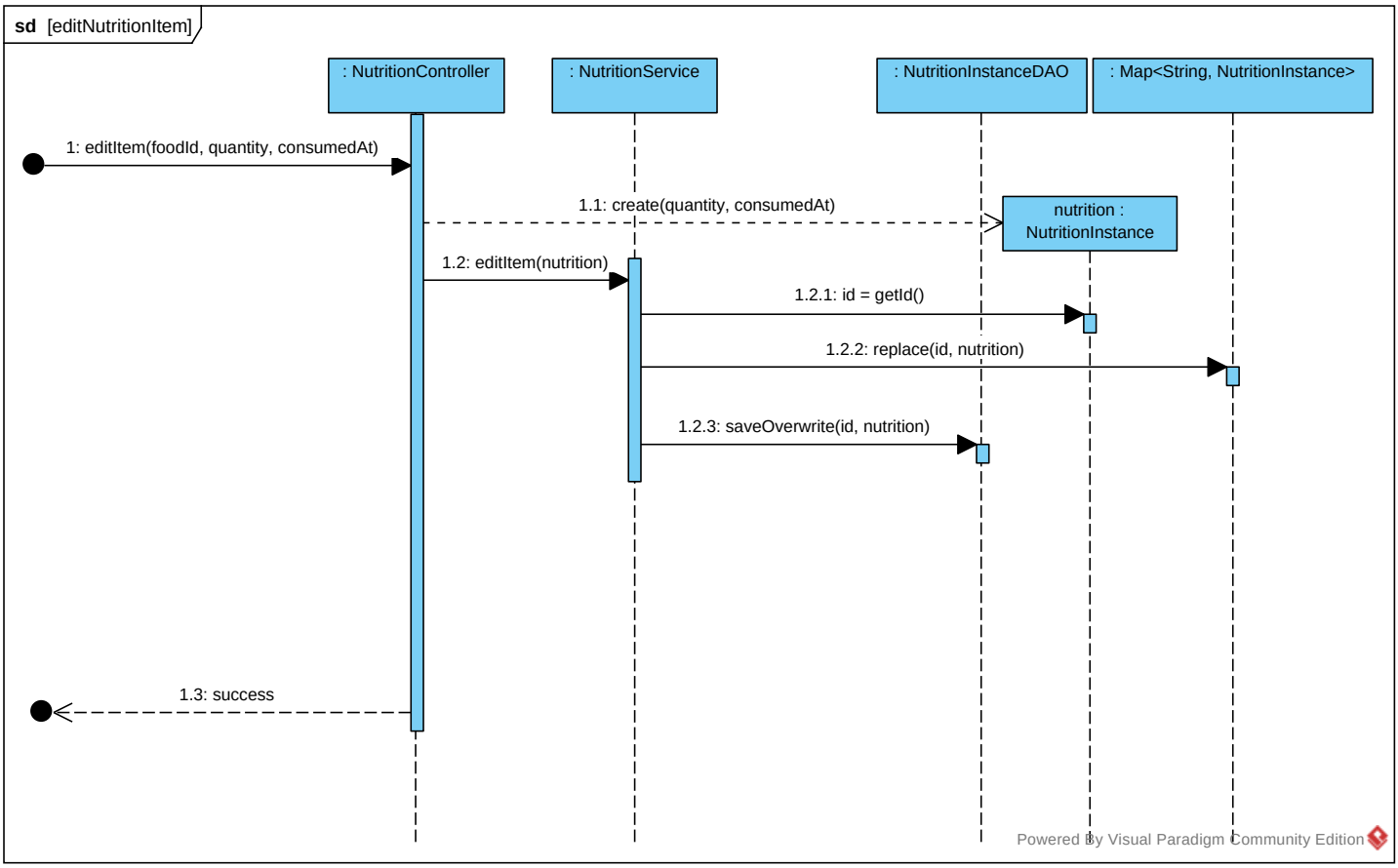

sd [addPantryItem]
: PantryManager
pantry : PantryService
items : Map<String, FoodInstance>
existingFoodInstance : FoodInstance
foodInstance : FoodInstance
1: addItem(foodId, quantity)
1.1: create(foodId, quantity)
1.2: addItem(foodInstance)
alt
[foodId not in items]
1.2.1: insert(foodId, foodInstance)
1.2.2: get(foodId)
1.2.3: existingFoodInstance
1.2.4: getQuantity()
1.2.5: curQuantity
1.2.6: getQuantity()
1.2.7: quantity
1.2.8: setQuantity(curQuantity + quantity)
1.2.9: replace(foodId, foodInstamce)
1.2.10: foodInstance
1.3: foodInstance
Powered By Visual Paradigm Community Edition

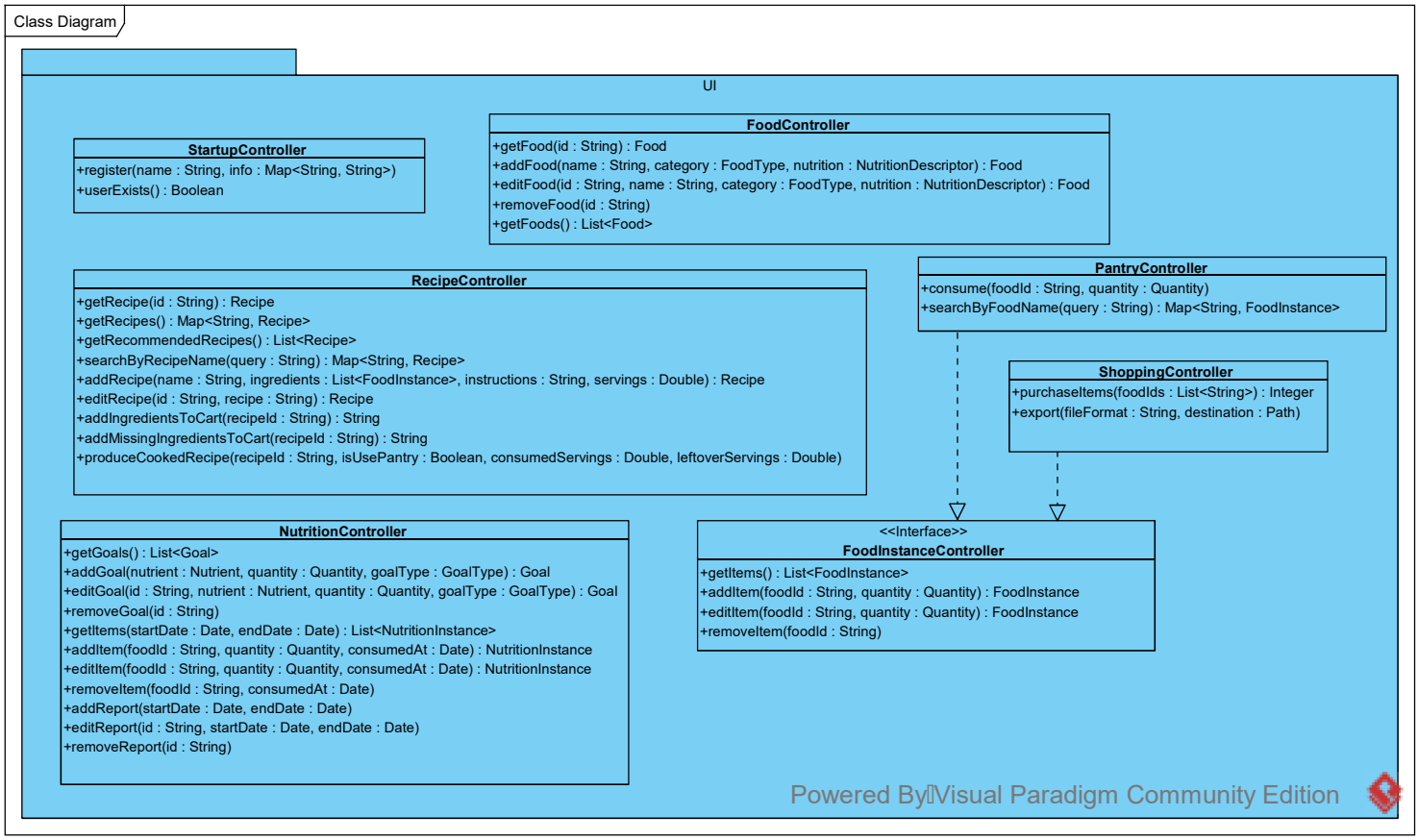

sd [consumeFromPantry]
: PantryController
pantry : PantryService
items : Map<String, FoodInstance>
foodInstance : FoodInstance
nutrition : NutritionService
1: consume(foodId, quantity)
1.1: removeItem(foodId, quantity)
opt
[id is valid and exists in items]
1.1.1: get(foodItem)
1.1.2: foodInstance
1.1.3: getQuantity()
1.1.4: curQuantity
1.1.5: setQuantity(curQuantity - quantity)
1.1.6: create(foodInstance, quantity)
nutritionInstance : NutritionInstance
1.1.7: addItem(nutritionInstance)
alt
[curQuantity - quantity is greater than 0]
1.1.8: replace(foodId, foodInstance)
1.1.9: remove(foodInstance)
1.1.10: success
1.2: success
Powered By Visual Paradigm Community Edition

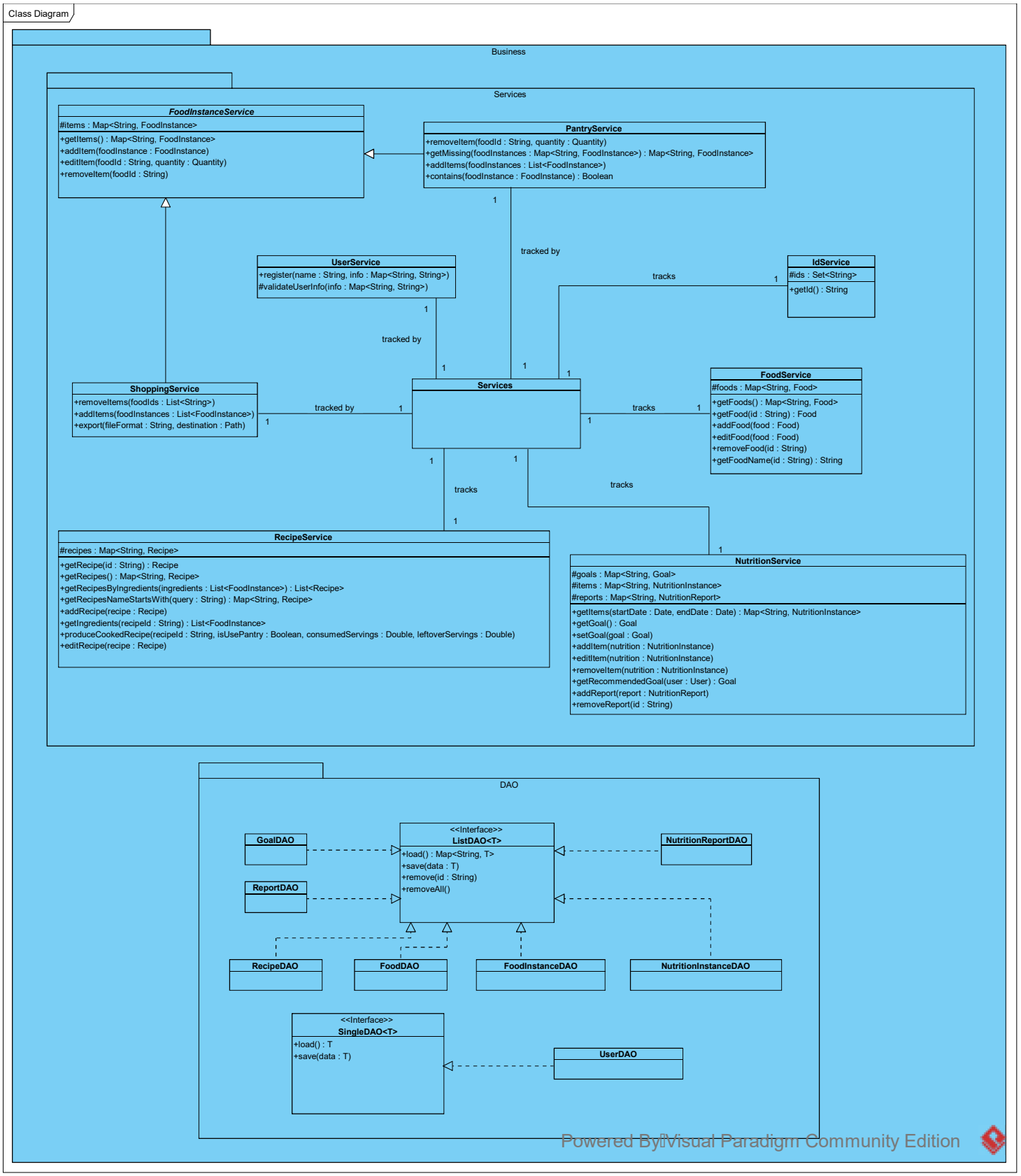

sd [removePantryItem]
: PantryController
pantry : PantryService
items : Map<String, FoodInstance>
1: removeItem(foodId)
1.1: removeItem(foodId)
opt
[foodId in items]
1.1.1: remove(foodId)
1.1.2: success
1.2: success
Powered By Visual Paradigm Community Edition

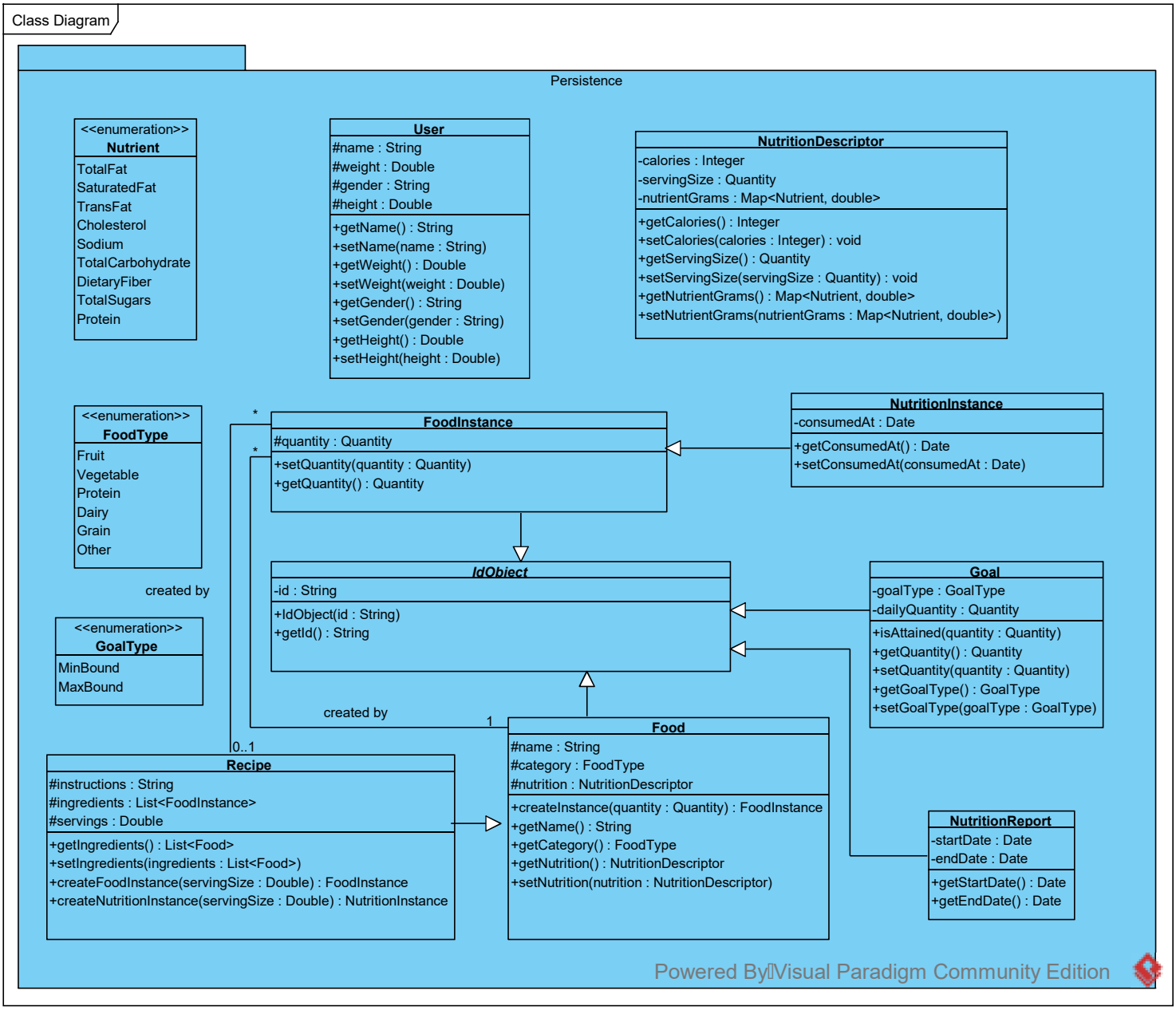

sd [addItem]
: NutritionController
: NutritionService
Items : Map<String, NutritionInstance>
: NutritionInstanceDAO
1: addItem(foodId, quantity, consumedAt)
1.1: create(quantity, consumedAt)
nutrition : NutritionInstance
1.2: addItem(nutrition)
1.2.1: id = getId()
1.2.2: saveNew(nutrition)
1.2.3: add(id, nutrition)
1.3: success
Powered By Visual Paradigm Community Edition


sd [deleteReport]
: NutritionController
: NutritionService
: Map<String, NutritionReport>
: ReportInstanceDAO
1: removeReport(id)
1.1: removeReport(id)
1.1.1: search(id)
1.1.2: report
1.1.3: remove(id)
1.1.4: delete(report)
1.2: success
Powered By Visual Paradigm Community Edition


sd [editNutritionItem]
: NutritionController
: NutritionService
: NutritionInstanceDAO
: Map<String, NutritionInstance>
1: editItem(foodId, quantity, consumedAt)
1.1: create(quantity, consumedAt)
nutrition : NutritionInstance
1.2: editItem(nutrition)
1.2.1: id = getId()
1.2.2: replace(id, nutrition)
1.2.3: saveOverwrite(id, nutrition)
1.3: success
Powered By Visual Paradigm Community Edition


Class Diagram
UI
StartupController
+register(name : String, info : Map<String, String>)
+userExists() : Boolean
FoodController
+getFood(id : String) : Food
+addFood(name : String, category : FoodType, nutrition : NutritionDescriptor) : Food
+editFood(id : String, name : String, category : FoodType, nutrition : NutritionDescriptor) : Food
+removeFood(id : String)
+getFoods() : List<Food>
RecipeController
+getRecipe(id : String) : Recipe
+getRecipes() : Map<String, Recipe>
+getRecommendedRecipes() : List<Recipe>
+searchByRecipeName(query : String) : Map<String, Recipe>
+addRecipe(name : String, ingredients : List<FoodInstance>, instructions : String, servings : Double) : Recipe
+editRecipe(id : String, recipe : String) : Recipe
+addIngredientsToCart(recipeId : String) : String
+addMissingIngredientsToCart(recipeId : String) : String
+produceCookedRecipe(recipeId : String, isUsePantry : Boolean, consumedServings : Double, leftoverServings : Double)
PantryController
+consume(foodId : String, quantity : Quantity)
+searchByFoodName(query : String) : Map<String, FoodInstance>
ShoppingController
+purchaseItems(foodIds : List<String>) : Integer
+export(fileFormat : String, destination : Path)
NutritionController
+getGoals() : List<Goal>
+addGoal(nutrient : Nutrient, quantity : Quantity, goalType : GoalType) : Goal
+editGoal(id : String, nutrient : Nutrient, quantity : Quantity, goalType : GoalType) : Goal
+removeGoal(id : String)
+getItems(startDate : Date, endDate : Date) : List<NutritionInstance>
+addItem(foodId : String, quantity : Quantity, consumedAt : Date) : NutritionInstance
+editItem(foodId : String, quantity : Quantity, consumedAt : Date) : NutritionInstance
+removeItem(foodId : String, consumedAt : Date)
+addReport(startDate : Date, endDate : Date)
+editReport(id : String, startDate : Date, endDate : Date)
+removeReport(id : String)
<<Interface>>
FoodInstanceController
+getItems() : List<FoodInstance>
+addItem(foodId : String, quantity : Quantity) : FoodInstance
+editItem(foodId : String, quantity : Quantity) : FoodInstance
+removeItem(foodId : String)
Powered By Visual Paradigm Community Edition


Class Diagram
Business
Services
FoodInstanceService
#items : Map<String, FoodInstance>
+getItems() : Map<String, FoodInstance>
+addItem(foodInstance : FoodInstance)
+editItem(foodId : String, quantity : Quantity)
+removeItem(foodId : String)
PantryService
+removeItem(foodId : String, quantity : Quantity)
+getMissing(foodInstances : Map<String, FoodInstance>) : Map<String, FoodInstance>
+addItems(foodInstances : List<FoodInstance>)
+contains(foodInstance : FoodInstance) : Boolean
1
tracked by
UserService
+register(name : String, info : Map<String, String>)
#validateUserInfo(info : Map<String, String>)
tracks
1
IdService
#ids : Set<String>
+getId() : String
1
tracked by
1
1
1
ShoppingService
+removeItems(foodIds : List<String>)
+addItems(foodInstances : List<FoodInstance>)
+export(fileFormat : String, destination : Path)
tracked by
1
1
Services
1
tracks
1
FoodService
#foods : Map<String, Food>
+getFoods() : Map<String, Food>
+getFood(id : String) : Food
+addFood(food : Food)
+editFood(food : Food)
+removeFood(id : String)
+getFoodName(id : String) : String
1
1
tracks
tracks
1
1
RecipeService
#recipes : Map<String, Recipe>
+getRecipe(id : String) : Recipe
+getRecipes() : Map<String, Recipe>
+getRecipesByIngredients(ingredients : List<FoodInstance>) : List<Recipe>
+getRecipesNameStartsWith(query : String) : Map<String, Recipe>
+addRecipe(recipe : Recipe)
+getIngredients(recipeId : String) : List<FoodInstance>
+produceCookedRecipe(recipeId : String, isUsePantry : Boolean, consumedServings : Double, leftoverServings : Double)
+editRecipe(recipe : Recipe)
NutritionService
#goals : Map<String, Goal>
#items : Map<String, NutritionInstance>
#reports : Map<String, NutritionReport>
+getItems(startDate : Date, endDate : Date) : Map<String, NutritionInstance>
+getGoal() : Goal
+setGoal(goal : Goal)
+addItem(nutrition : NutritionInstance)
+editItem(nutrition : NutritionInstance)
+removeItem(nutrition : NutritionInstance)
+getRecommendedGoal(user : User) : Goal
+addReport(report : NutritionReport)
+removeReport(id : String)
DAO
GoalDAO
<<Interface>>
ListDAO<T>
+load() : Map<String, T>
+save(data : T)
+remove(id : String)
+removeAll()
NutritionReportDAO
ReportDAO
RecipeDAO
FoodDAO
FoodInstanceDAO
NutritionInstanceDAO
<<Interface>>
SingleDAO<T>
+load() : T
+save(data : T)
UserDAO
Powered By Visual Paradigm Community Edition


Class Diagram
Persistence
<<enumeration>>
Nutrient
TotalFat
SaturatedFat
TransFat
Cholesterol
Sodium
TotalCarbohydrate
DietaryFiber
TotalSugars
Protein
User
#name : String
#weight : Double
#gender : String
#height : Double
+getName() : String
+setName(name : String)
+getWeight() : Double
+setWeight(weight : Double)
+getGender() : String
+setGender(gender : String)
+getHeight() : Double
+setHeight(height : Double)
NutritionDescriptor
-calories : Integer
-servingSize : Quantity
-nutrientGrams : Map<Nutrient, double>
+getCalories() : Integer
+setCalories(calories : Integer) : void
+getServingSize() : Quantity
+setServingSize(servingSize : Quantity) : void
+getNutrientGrams() : Map<Nutrient, double>
+setNutrientGrams(nutrientGrams : Map<Nutrient, double>)
<<enumeration>>
FoodType
Fruit
Vegetable
Protein
Dairy
Grain
Other
created by
<<enumeration>>
GoalType
MinBound
MaxBound
*
*
FoodInstance
#quantity : Quantity
+setQuantity(quantity : Quantity)
+getQuantity() : Quantity
NutritionInstance
-consumedAt : Date
+getConsumedAt() : Date
+setConsumedAt(consumedAt : Date)
IdObject
-id : String
+IdObject(id : String)
+getId() : String
Goal
-goalType : GoalType
-dailyQuantity : Quantity
+isAttained(quantity : Quantity)
+getQuantity() : Quantity
+setQuantity(quantity : Quantity)
+getGoalType() : GoalType
+setGoalType(goalType : GoalType)
created by
1
0..1
Recipe
#instructions : String
#ingredients : List<FoodInstance>
#servings : Double
+getIngredients() : List<Food>
+setIngredients(ingredients : List<Food>)
+createFoodInstance(servingSize : Double) : FoodInstance
+createNutritionInstance(servingSize : Double) : NutritionInstance
Food
#name : String
#category : FoodType
#nutrition : NutritionDescriptor
+createInstance(quantity : Quantity) : FoodInstance
+getName() : String
+getCategory() : FoodType
+getNutrition() : NutritionDescriptor
+setNutrition(nutrition : NutritionDescriptor)
NutritionReport
-startDate : Date
-endDate : Date
+getStartDate() : Date
+getEndDate() : Date
Powered By Visual Paradigm Community Edition

GRASP Patterns

Options:
Information Expert - knowledge of having data and who has the data
Creator - responsibility to create another class if aggregates, contains, records, or has initializing data of another class (expert)
Controller - Assigns responsibility of dealing with system events that represents the overall system or a use case scenario (non-UI)
Low Coupling - assign responsibilities so that coupling remains low
High Cohesion - keep objects appropriately focuses and manageable, used in support of low coupling
Indirection - supports low coupling and reuse by assigning mediation to an intermediary object
Polymorphism - responsibility for defining variation of behaviors based on type is assigned to the type for which this variation happens, use polymorphic options instead of branching
Pure Fabrication - does not represent a concept in problem domain, only made to achieve low coupling, high cohesion, and reuse potential

Class Name	StartupController
GRASP Pattern	Controller
Description	A controller for the first time startup of the application. Handles user input relating to user registration.

Class Name	RecipeController
GRASP Pattern	Controller
Description	A controller for the recipe list. This class handles input events directed at the Recipe portion of this application.

Class Name	FoodController
GRASP Pattern	Controller
Description	A controller for food types. This class handles input events directed at adding, editing and removing foods from the food database accessor.

Class Name	FoodInstanceController
GRASP Pattern	Pure Fabrication, Controller
Description	This class is an abstract class that is used as a framework for its children which are the NutritionController, RecipeController, ShoppingController, and PantryController. This class has a map of ids and FoodInstances which all child Controllers use to manage data.

Class Name	PantryController
GRASP Pattern	Controller
Description	A controller for the pantry. This class handles input events directed at the Pantry portion of the application.

Class Name	NutritionController
GRASP Pattern	Controller
Description	A controller for nutrition tracking. This class handles input events directed at the Nutrition portion of the application.

Class Name	ShoppingController
GRASP Pattern	Controller
Description	A controller for the shopping list. This class handles input events directed at the shopping list portion of this application.

Class Name	Services
GRASP Pattern	Creator
Description	This class handles the creation of services in the business layer.

Class Name	UserService
GRASP Pattern	High Cohesion
Description	This class handles registering the user upon first-time setup. The class only has a few methods and data specifically relating to this functionality.

Class Name	User
GRASP Pattern	High Cohesion
Description	This class groups together information relating to the user such as name, weight, and gender. Putting all this in one class keeps our program highly cohesive.

Class Name	RecipeService
GRASP Pattern	High Cohesion, Low Coupling
Description	This class handles functionality relating to the recipe portion of the application. It tries to have minimal connection with other classes interacting with the RecipeController and the database accessor.

Class Name	Recipe
GRASP Pattern	Information Expert, High Cohesion
Description	This class parallels a real-life recipe. It holds data and methods relating to capturing the recipe.

Class Name	FoodService
GRASP Pattern	High Cohesion, Low Coupling
Description	This class handles functionality relating to food types within the application. It tries to have minimal connection with other classes interacting with the FoodController and the database accessor.

Class Name	Food
GRASP Pattern	Information Expert
Description	An information expert for food types. This class contains all necessary information to store a food-type object.

Class Name	IdService
GRASP Pattern	Information Expert, Single Responsibility
Description	This class handles assigning unique identifiers to objects which need them.

Class Name	IdObject
GRASP Pattern	Pure Fabrication, High Cohesion
Description	An abstract class that handles the aid attribute so that all the other objects which need ids can inherit from it.

Class Name	FoodInstance
GRASP Pattern	High Cohesion
Description	Parallels an actual food in real life as opposed to the abstract idea of a food. Used to keep track of what is in the pantry, recipe ingredients, and shopping list items. The nutrition log has a special version of this class which extends FoodInstance.

Class Name	PantryService
GRASP Pattern	High Cohesion, Low Coupling
Description	This class handles functionality relating to the pantry within the application. It tries to have minimal connection with other classes, interacting with the PantryController and the database.

Class Name	Goal
GRASP Pattern	High Cohesion
Description	This class holds data related to a nutrition goal a user has set. All methods and data related to a user goal are stored in this class.

Class Name	NutritionInstance
GRASP Pattern	High Cohesion
Description	This class is used in the Nutrition log to store data about what the user consumed when. The class inherits from FoodInstance and adds functionality related to time of consumption.

Class Name	FoodInstanceService
GRASP Pattern	High Cohesion, Low Coupling
Description	This class handles Food Instances. All methods relating to Food Instances are in one place along with the Map of Food Instances. This helps avoid coupling between the FoodInstances and the classes which use them.

Class Name	NutritionDescriptor
GRASP Pattern	Information Expert
Description	An information expert for nutrition information. This class contains all necessary information regarding nutrition.

Class Name	NutritionService
GRASP Pattern	High Cohesion, Low Coupling
Description	Inherits from FoodInstanceService. This class handles functionality relating to the Nutrition Log within the application. It tries to have minimal connection with other classes, interacting with the NutritionController and the database accessor.

Class Name	ShoppingService
GRASP Pattern	High Cohesion, Low Coupling
Description	This class handles functionality relating to the Shopping List within the application. It tries to have minimal connection with other classes, interacting with the ShoppingController and the database accessor.

Class Name	UserDBAccessor
GRASP Pattern	Low Coupling
Description	This database accessor helps separate the service from the database and keeps the services separated from each other.

Class Name	FileDBSingleAccessor
GRASP Pattern	Pure Fabrication
Description	This abstract class implements methods for database accessors that do single accesses.

Class Name	RecipeDBAccessor
GRASP Pattern	Low Coupling
Description	This database accessor helps separate the service from the database and keeps the services separated from each other.

Class Name	FoodDBAccessor
GRASP Pattern	Low Coupling
Description	This database accessor helps separate the service from the database and keeps the services separated from each other.

Class Name	NutritionInstanceDBAccessor
GRASP Pattern	Low Coupling
Description	This database accessor helps separate the service from the database and keeps the services separated from each other.

Class Name	FileDBListAccessor
GRASP Pattern	Pure Fabrication
Description	This abstract class implements methods for database accessors that access multiple elements at a time.

Test Coverage Plan

The Test Coverage Plan is designed to prescribe the scope, approach, and high-level overview of all testing activities of the FoodPants project. The plan identifies the items to be tested, the features to be tested, and the types of testing to be performed. Unit testing will be performed using JUnit. Integration testing will be performed by analyzing the UI to ensure that the interfaces among the subsystems operate correctly. System testing will be performed by analyzing the UI to make sure all requirements are met for the user.

(1) PANTRY USE CASES (Scope: Pantry system):
- 1.1 Manage pantry (add/edit/remove items manually)
    - Test successful adding of food item (Expected Result: food item should appear in pantry with correct quantity and name)
    - Test successful editing of food item details (Expected Result: food item name or quantity should appear updated to match new entered details)
    - Test successful removal of food items (Expected Result: food item should no longer appear in pantry)
    - Test adding of food item with missing details (Expected Result: program should wait until user enters all missing details or exits without adding)
    - Test adding of food item that already exists (Expected Result: program should add new quantity to already existing quantity of food item that is already in the pantry)
    - Test adding of food item that has new food type (Expected Result: program should prompt use to add a new food type)
    - Test editing of food item to match name of another food item that exists in the pantry (Expected Result: food item is removed from pantry and quantity of removed food item is added to other food item)
- 1.2 Search pantry (by name of food type)
    - Test searching pantry for specific food item (Expected Result: food item with name containing user's query should appear in results)
    - Test searching pantry for query that does not match any food item (Expected Result: program should tell user no match exists)
    - Test searching pantry with query that matches multiple food items (Expected Result: all food items with name containing user's query should appear in results)
- 1.3 Consume specific item (add to nutrition log)
    - Test successful consumption of 1 of food item in pantry (Expected Result: food item should be updated with decremented quantity)
    - Test successful consumption of more than 1 of a food item in pantry (Expected Result: food item should be updated with previous quantity – amount consumed)
    - Test consumption of quantity amount of food item in pantry (Expected Result: quantity left is 0 meaning food item should be removed from pantry)
    - Test consumption of more than quantity amount of food item in pantry (Expected Result: negative quantity left meaning food item should be removed from pantry)

(2) NUTRITION LOG USE CASES (Scope: Nutrition Log system):
- 2.1 Manage nutrition log directly
    - Test successful adding of item to nutrition log (Expected Result: new item appears in nutrition log)
    - Test successful editing of nutrition item to nutrition log (Expected Result: details of item in nutrition log are updated matching the edits that were made)

    - Test successful removal of nutrition item to nutrition log (Expected Result: item no longer appears in nutrition log)
    - Test adding of item to nutrition log with duplicate details (Expected Result: new item should not be added to nutrition log and existing nutrition item should be updated with nutritional info of original item + nutritional info of new item)
- 2.2 Set nutritional goals
    - Test successful adding of a nutrition goal (Expected Result: goal should appear in nutrition goals as well as percentage of progress towards the goal)
    - Test successful editing of a nutrition goal (Expected Result: goal should appear with updated details)
    - Test successful removal of a nutrition goal (Expected Result: goal should no longer appear in nutrition goals)
    - Test adding a duplicate nutrition goal (Expected Result: goal should not be added and user should be warned that the goal already exists and they can update the existing goal if they wish)
    - Test editing nutrition goal to match another nutrition goal (Expected Result: goal should not be updated and user should be warned that another goal already exists with the same details)
- 2.3 View nutrition report
    - Test successful viewing of nutrition report (Expected Result: all nutrition log items appear in report)
    - Test viewing of nutrition report with no nutrition logs (Expected Result: user is told there is no nutrition log items to report)

(3) RECIPE USE CASES (Scope: Recipe system):
- 3.1 Create
    - Test successful creation of recipe (Expected Result: recipe appears in list of recipes with correct details)
    - Test creation of recipe with duplicate name (Expected Result: recipe should not be added and user should be warned that recipe with same name already exists but they can rename one)
    - Test creation of recipe with new food types (Expected Result: user should be prompted to add new food types and then recipe will appear in list of recipes with correct details)
- 3.2 View (all/one) +Search
    - Test successful viewing of recipes (Expected Result: all existing recipes are visible to user)
    - Test viewing of recipes when none exist (Expected Result: user should be told there are no existing recipes)
    - Test successful search of an existing recipe by its name (Expected Result: recipe with name containing query should be displayed to user)
    - Test search for a recipe that does not exist (Expected Result: user should be told there are no matching recipes)
    - Test search with a keyword contained in multiple recipe names (Expected Result: all recipes with name containing query should be displayed to user)
- 3.3 Add items from recipe to shopping list (optional add even if already in pantry)
    - Test successful adding of all items from recipe to shopping list (Expected Result: all ingredients for recipe should appear on shopping list)
    - Test successful adding of items in recipe not in pantry to shopping list (Expected Result: all ingredients not already in the pantry with sufficient quantity for the recipe should appear in the shopping list)
    - Test adding of all items to shopping list when there are existing shopping list items (Expected Result: recipe items are appended to shopping list)

    - Test adding of all items to shopping list when there are existing shopping list items for ingredients in the recipe (Expected Result: recipe items are appended to shopping list and duplicate items are merged with two quantities added)
    - Test adding of items not in pantry to shopping list when there are existing shopping list items (Expected Result: recipe items are appended to shopping list)
    - Test adding of items not in pantry to shopping list when there are existing shopping list items for ingredients in the recipe (Expected Result: recipe items are appended to shopping list and duplicate items are merged with two quantities added)
- 3.4 Edit ("Delete" is a button within the "Edit" menu)
    - Test successful editing of recipe (Expected Result: recipe is updated with changed details)
    - Test successful removal of recipe (Expected Result: recipe no longer appears in list of recipes)
    - Test editing of recipe to match another recipe's name (Expected Result: recipe is not updated and user is warned that they cannot change one recipe to have the same name as another)
- 3.5 Get recommended (recommend food that you can make using stuff in your pantry)
    - Test successful retrieval of recommended recipes (Expected Result: random and unique recommended recipes are visible to the user)
    - Test retrieval of recommended recipes when no recipes exist (Expected Result: no recommended recipes are returned or visible)
    - Test retrieval of recommended recipes when only one recipe exists (Expected Result: only one recommended recipe is returned and visible as opposed to seeing multiple of the same recipe as recommended)
- 3.6 Cook/consume (ask user how much they ate, how much leftover → leftovers to pantry)
    - Test successful cooking of recipe (Expected Result: quantity of items in pantry is updated to reflect how much of each item a user consumed and leftover items are added to pantry)
    - Test cooking of recipe with leftovers of new food type (Expect Result: user is prompted to add new food type and leftovers are added to pantry)
    - Testing cooking of recipe that requires a greater quantity of ingredients than is available in pantry (Expected Results: ingredients all used up are removed from the pantry)
    - Testing cooking of recipe that requires the exact amount of ingredients that are available in the pantry (Expected Results: ingredients with 0 quantity remaining in pantry are removed)

(4) SHOPPING LIST USE CASES (Scope: Shopping List system):
- 4.1 Manage list (add/edit/remove items manually)
    - Test successful adding of item to shopping list (Expected Result: new item appears in shopping list)
    - Test successful editing of item in shopping list (Expected Result: item in shopping list is updated with changed values)
    - Test successful removal of item in shopping list (Expected Result: item no longer appears in shopping list)
    - Test adding of item that already exists in shopping list to list (Expected Result: duplicated item is not added to shopping list and existing item's quantity to buy is updated by adding to it duplicate item's quantity to buy)
    - Test adding of new food to shopping list (Expected Result: user is prompted to add new food type, then item appears in shopping list)
    - Test editing item in shopping list to match another item (Expected Result: current item is removed from shopping list and other item's quantity to buy is updated by adding to it current item's quantity to buy)

- 4.2 Export (Print/PDF)

    - Test successful exporting of shopping list to pdf (Expected Result: pdf is created with all of shopping list details)
    - Test export of empty shopping list to pdf (Expected Result: no pdf is created and user is told shopping list is empty)
    - Test successful exporting of shopping list to txt (Expected Result: txt is created with all of shopping list details)
    - Test export of empty shopping list to txt (Expected Result: no txt is created and user is told shopping list is empty)
    - Test successful exporting of shopping list to csv (Expected Result: csv is created with all of shopping list details)
    - Test export of empty shopping list to csv (Expected Result: no csv is created and user is told shopping list is empty)
    - Test export of shopping list to an invalid format (Expected Result: no file is created is created and user is told export format is invalid)
- 4.3 Mark item(s) as purchased (individual items one by one or Extension: all at once)
    - Test successful marking of one item as purchased (Expected Result: selected item is visibly checked off as purchased)
    - Test successful marking of all items as purchased (Expected Result: all items are visibly checked off as purchased)
    - Test marking of all items as purchased when an item is already marked as purchased (Expected Result: all items are visibly checked off as purchased)
    - Test marking all items as purchased when only one item exists in a list (Expected Result: single item in shopping list is visibly checked off as purchased)
    - Test marking all items as purchased when all items in shopping list are already marked as purchased (Expected Result: no visible change)
    - Test marking all items as purchased when no items exist in shopping list (Expected Result: no visible change)

(5) STARTUP USE CASES (Scope: Startup):
- 5.1 Startup (Level: Summary, extends several)
    - Test successful registration (Expected Result: user instance is created, user details are logged, startup tutorial is started)
    - Test registration with already existing name (Expected Result: user is told they must use a unique name)

(6) FOOD TYPE DB USE CASES (Scope: Food Type system):
- 6.1 Manage list of food types from within Food Type DB Page
    - Test successful editing of food type (Expected Result: food type details are updated in database)
    - Test successful removal of food type (Expected Result: food type is removed from database)
    - Test editing of food type to match another food type's name (Expected Result: food type is not updated and user is told that there is another food type with the same name already)
- 6.2 Create new food type (extends "Manage list of food types") (Level: Subfunction)
    - Test successful creation of new food type (Expected Result: food type and correct details are added to database)
    - Test creation of already existing food type with same name (Expected Result: food type is not added and user is told they have a food type with the same name already)
    - Test creation of food type with same name but different case (Expected Result: food type is not added and user is told they have a food type with the same name already)

FoodPantsUpdated

Mar 30, 2022

http://

Project manager	
Project dates	Jan 18, 2022 - Apr 28, 2022
Completion	0%
Tasks	28
Resources	4

Tasks

Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/20/22
Infastructure Initialization	1/21/22	1/21/22
Requirements Analysis	1/24/22	1/25/22
Use Cases	1/26/22	1/31/22
Traceability Matrix	2/1/22	2/1/22
System Sequence Diagrams	2/2/22	2/14/22
System Operations	2/15/22	2/18/22
Wireframes	2/21/22	2/23/22
Domain Model	2/24/22	2/28/22
Presentation and Reporting	3/1/22	3/1/22
Design	3/2/22	3/31/22
Design Model	3/2/22	3/10/22
Sequence Diagrams	3/11/22	3/21/22
Package Diagrams	3/22/22	3/22/22
GRASP Patterns	3/23/22	3/25/22
Test Coverage	3/28/22	3/28/22
Prototyping	3/29/22	3/30/22
Presentation and Reporting	3/31/22	3/31/22
Implementation	4/1/22	4/27/22
Backend	4/1/22	4/8/22
User Interface	4/11/22	4/15/22
User Input Validation	4/18/22	4/19/22
Imports/Exports	4/20/22	4/21/22
Unit-Testing	4/22/22	4/25/22
Documentation	4/26/22	4/26/22

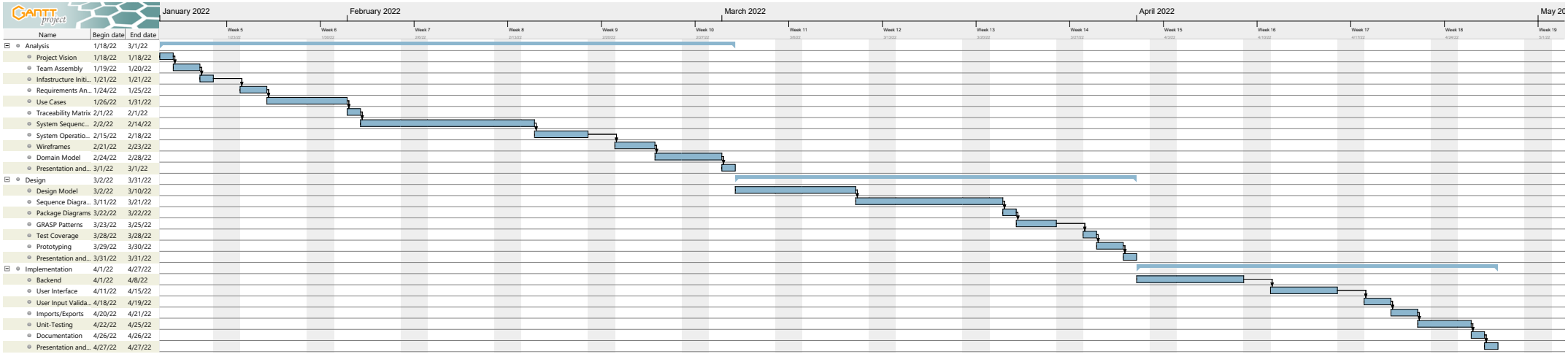
Tasks

Name	Begin date	End date
Presentation and Reporting	4/27/22	4/27/22

Resources

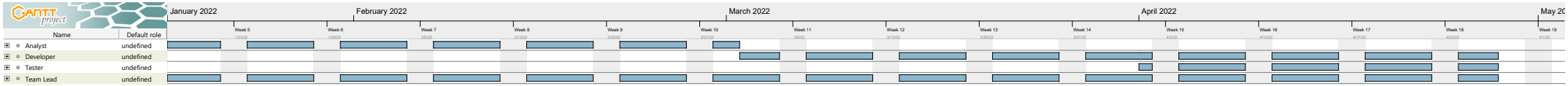
Name	Default role
Analyst	undefined
Developer	undefined
Tester	undefined
Team Lead	undefined

Gantt Chart



Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/20/22
Infastructure Initi...	1/21/22	1/21/22
Requirements An...	1/24/22	1/25/22
Use Cases	1/26/22	1/31/22
Traceability Matrix	2/1/22	2/1/22
System Sequenc...	2/2/22	2/14/22
System Operatio...	2/15/22	2/18/22
Wireframes	2/21/22	2/23/22
Domain Model	2/24/22	2/28/22
Presentation and...	3/1/22	3/1/22
Design	3/2/22	3/31/22
Design Model	3/2/22	3/10/22
Sequence Diagra...	3/11/22	3/21/22
Package Diagrams	3/22/22	3/22/22
GRASP Patterns	3/23/22	3/25/22
Test Coverage	3/28/22	3/28/22
Prototyping	3/29/22	3/30/22
Presentation and...	3/31/22	3/31/22
Implementation	4/1/22	4/27/22
Backend	4/1/22	4/8/22
User Interface	4/11/22	4/15/22
User Input Valida...	4/18/22	4/19/22
Imports/Exports	4/20/22	4/21/22
Unit-Testing	4/22/22	4/25/22
Documentation	4/26/22	4/26/22
Presentation and...	4/27/22	4/27/22

Resources Chart



FoodPants

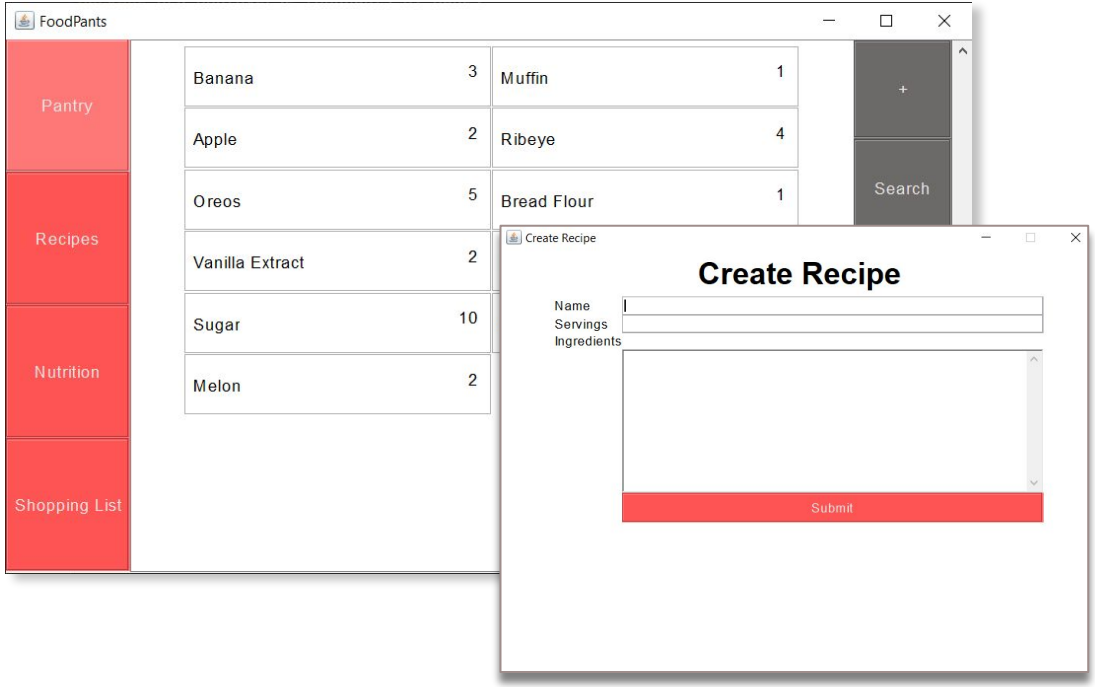
Iteration II

“

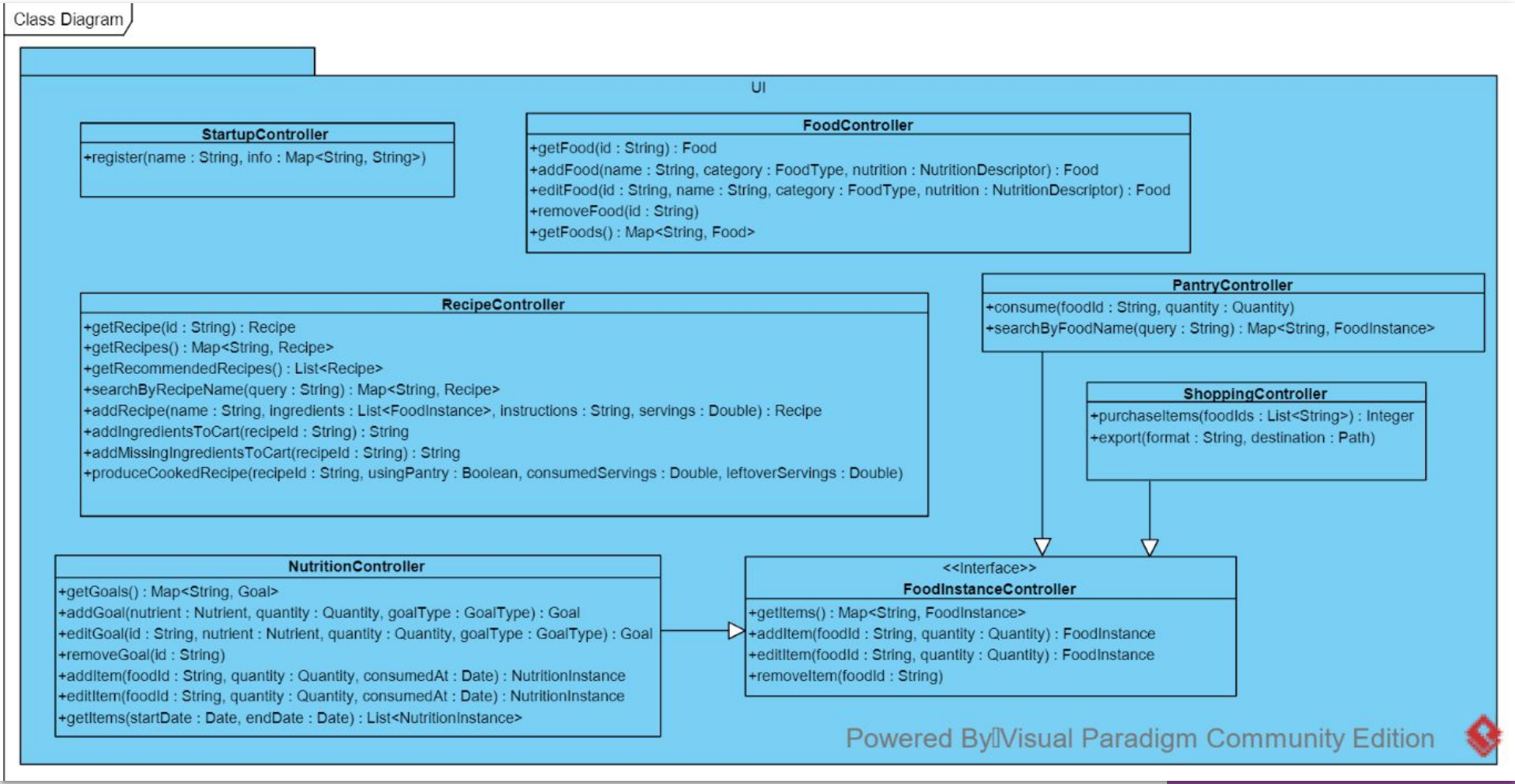
Food pants is the greatest new app coming to the market. I wish I found it sooner.”

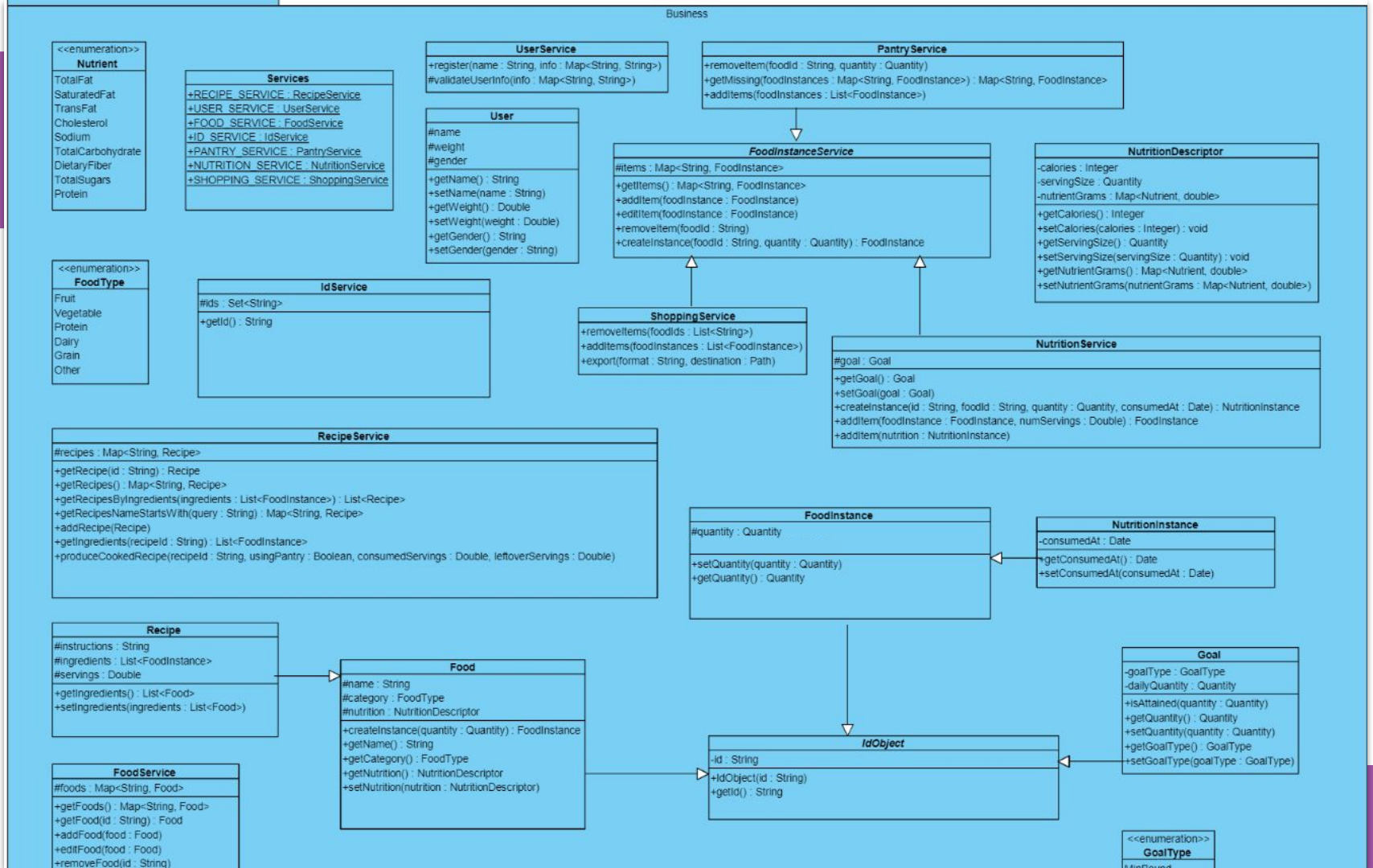
- Abe Lincolns

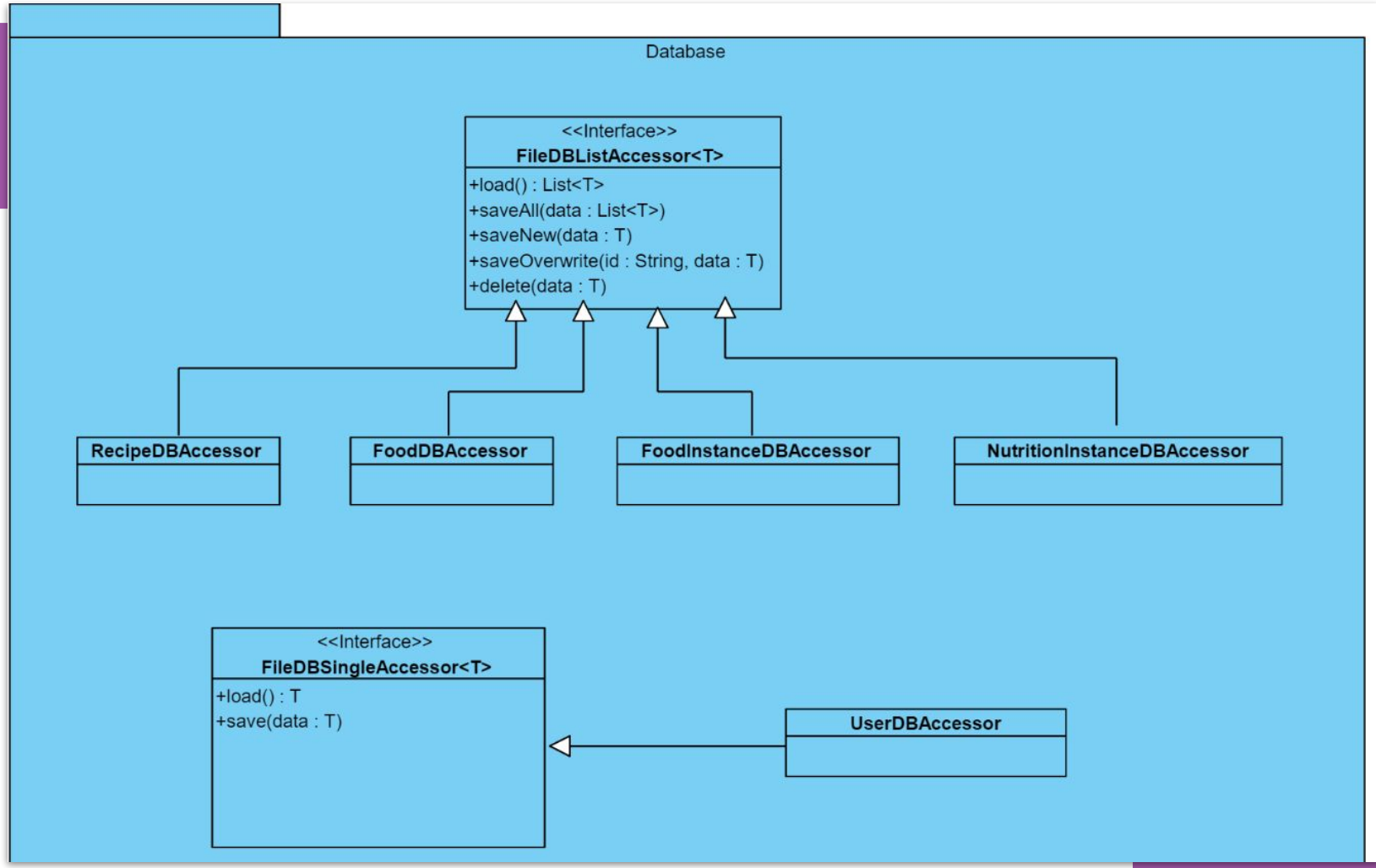
Demo User Interface



Class Diagrams


Class Diagram
UI
StartupController
+register(name : String, info : Map<String, String>)
FoodController
+getFood(id : String) : Food
+addFood(name : String, category : FoodType, nutrition : NutritionDescriptor) : Food
+editFood(id : String, name : String, category : FoodType, nutrition : NutritionDescriptor) : Food
+removeFood(id : String)
+getFoods() : Map<String, Food>
RecipeController
+getRecipe(id : String) : Recipe
+getRecipes() : Map<String, Recipe>
+getRecommendedRecipes() : List<Recipe>
+searchByRecipeName(query : String) : Map<String, Recipe>
+addRecipe(name : String, ingredients : List<FoodInstance>, instructions : String, servings : Double) : Recipe
+addIngredientsToCart(recipeId : String) : String
+addMissingIngredientsToCart(recipeId : String) : String
+produceCookedRecipe(recipeId : String, usingPantry : Boolean, consumedServings : Double, leftoverServings : Double)
PantryController
+consume(foodId : String, quantity : Quantity)
+searchByFoodName(query : String) : Map<String, FoodInstance>
ShoppingController
+purchaseItems(foodIds : List<String>) : Integer
+export(format : String, destination : Path)
NutritionController
+getGoals() : Map<String, Goal>
+addGoal(nutrient : Nutrient, quantity : Quantity, goalType : GoalType) : Goal
+editGoal(id : String, nutrient : Nutrient, quantity : Quantity, goalType : GoalType) : Goal
+removeGoal(id : String)
+addItem(foodId : String, quantity : Quantity, consumedAt : Date) : NutritionInstance
+editItem(foodId : String, quantity : Quantity, consumedAt : Date) : NutritionInstance
+getItems(startDate : Date, endDate : Date) : List<NutritionInstance>
<<Interface>>
FoodInstanceController
+getItems() : Map<String, FoodInstance>
+addItem(foodId : String, quantity : Quantity) : FoodInstance
+editItem(foodId : String, quantity : Quantity) : FoodInstance
+removeItem(foodId : String)
Powered By Visual Paradigm Community Edition


Business
<<enumeration>>
Nutrient
TotalFat
SaturatedFat
TransFat
Cholesterol
Sodium
TotalCarbohydrate
DietaryFiber
TotalSugars
Protein
Services
+RECIPE_SERVICE : RecipeService
+USER_SERVICE : UserService
+FOOD_SERVICE : FoodService
+ID_SERVICE : IdService
+PANTRY_SERVICE : PantryService
+NUTRITION_SERVICE : NutritionService
+SHOPPING_SERVICE : ShoppingService
UserService
+register(name : String, info : Map<String, String>)
#validateUserInfo(info : Map<String, String>)
User
#name
#weight
#gender
+getName() : String
+setName(name : String)
+getWeight() : Double
+setWeight(weight : Double)
+getGender() : String
+setGender(gender : String)
PantryService
+removeItem(foodId : String, quantity : Quantity)
+getMissing(foodInstances : Map<String, FoodInstance>) : Map<String, FoodInstance>
+addItems(foodInstances : List<FoodInstance>)
FoodInstanceService
#items : Map<String, FoodInstance>
+getItems() : Map<String, FoodInstance>
+addItem(foodInstance : FoodInstance)
+editItem(foodInstance : FoodInstance)
+removeItem(foodId : String)
+createInstance(foodId : String, quantity : Quantity) : FoodInstance
NutritionDescriptor
-calories : Integer
-servingSize : Quantity
-nutrientGrams : Map<Nutrient, double>
+getCalories() : Integer
+setCalories(calories : Integer) : void
+getServingSize() : Quantity
+setServingSize(servingSize : Quantity) : void
+getNutrientGrams() : Map<Nutrient, double>
+setNutrientGrams(nutrientGrams : Map<Nutrient, double>)
<<enumeration>>
FoodType
Fruit
Vegetable
Protein
Dairy
Grain
Other
IdService
#ids : Set<String>
+getId() : String
ShoppingService
+removeItems(foodIds : List<String>)
+addItems(foodInstances : List<FoodInstance>)
+export(format : String, destination : Path)
NutritionService
#goal : Goal
+getGoal() : Goal
+setGoal(goal : Goal)
+createInstance(id : String, foodId : String, quantity : Quantity, consumedAt : Date) : NutritionInstance
+addItem(foodInstance : FoodInstance, numServings : Double) : FoodInstance
+addItem(nutrition : NutritionInstance)
RecipeService
#recipes : Map<String, Recipe>
+getRecipe(id : String) : Recipe
+getRecipes() : Map<String, Recipe>
+getRecipesByIngredients(ingredients : List<FoodInstance>) : List<Recipe>
+getRecipesNameStartsWith(query : String) : Map<String, Recipe>
+addRecipe(Recipe)
+getIngredients(recipeId : String) : List<FoodInstance>
+produceCookedRecipe(recipeId : String, usingPantry : Boolean, consumedServings : Double, leftoverServings : Double)
FoodInstance
#quantity : Quantity
+setQuantity(quantity : Quantity)
+getQuantity() : Quantity
NutritionInstance
-consumedAt : Date
+getConsumedAt() : Date
+setConsumedAt(consumedAt : Date)
Recipe
#instructions : String
#ingredients : List<FoodInstance>
#servings : Double
+getIngredients() : List<Food>
+setIngredients(ingredients : List<Food>)
Food
#name : String
#category : FoodType
#nutrition : NutritionDescriptor
+createInstance(quantity : Quantity) : FoodInstance
+getName() : String
+getCategory() : FoodType
+getNutrition() : NutritionDescriptor
+setNutrition(nutrition : NutritionDescriptor)
IdObject
-id : String
+IdObject(id : String)
+getId() : String
Goal
-goalType : GoalType
-dailyQuantity : Quantity
+isAttained(quantity : Quantity)
+getQuantity() : Quantity
+setQuantity(quantity : Quantity)
+getGoalType() : GoalType
+setGoalType(goalType : GoalType)
FoodService
#foods : Map<String, Food>
+getFoods() : Map<String, Food>
+getFood(id : String) : Food
+addFood(food : Food)
+editFood(food : Food)
+removeFood(id : String)
<<enumeration>>
GoalType


Database
<<Interface>>
FileDBListAccessor<T>
+load() : List<T>
+saveAll(data : List<T>)
+saveNew(data : T)
+saveOverwrite(id : String, data : T)
+delete(data : T)
RecipeDBAccessor
FoodDBAccessor
FoodInstanceDBAccessor
NutritionInstanceDBAccessor
<<Interface>>
FileDBSingleAccessor<T>
+load() : T
+save(data : T)
UserDBAccessor

Class Name	PantryController
GRASP Pattern	Controller
Description	A controller for the pantry. This class handles input events directed at the Pantry portion of the application. It also serves to separate the UI from business logic to maximize reusability.

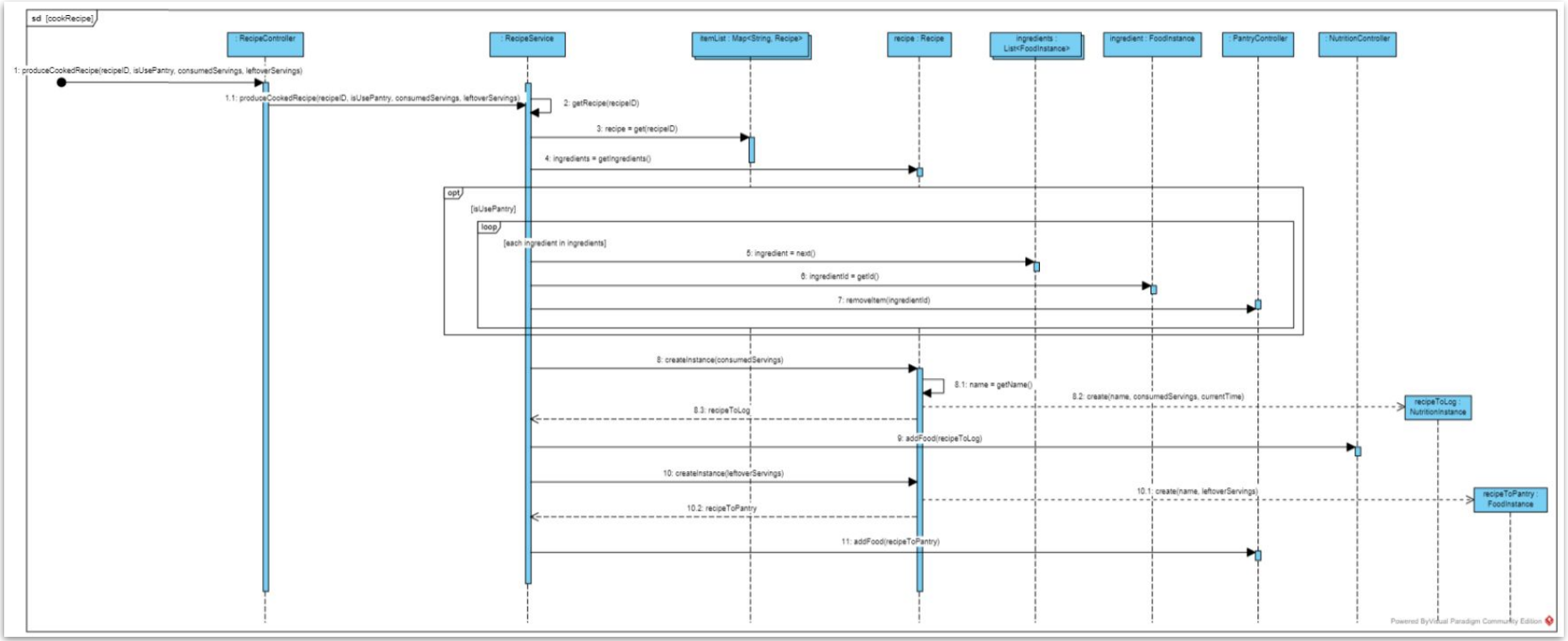
Class Name	Food
GRASP Pattern	Information Expert
Description	An information expert for food types. This class contains all necessary information to store a food type object.

Class Name	PantryService
GRASP Pattern	Pure Fabrication
Description	This class handles functionality relating to the pantry within the application. It is dedicated to handling business logic.

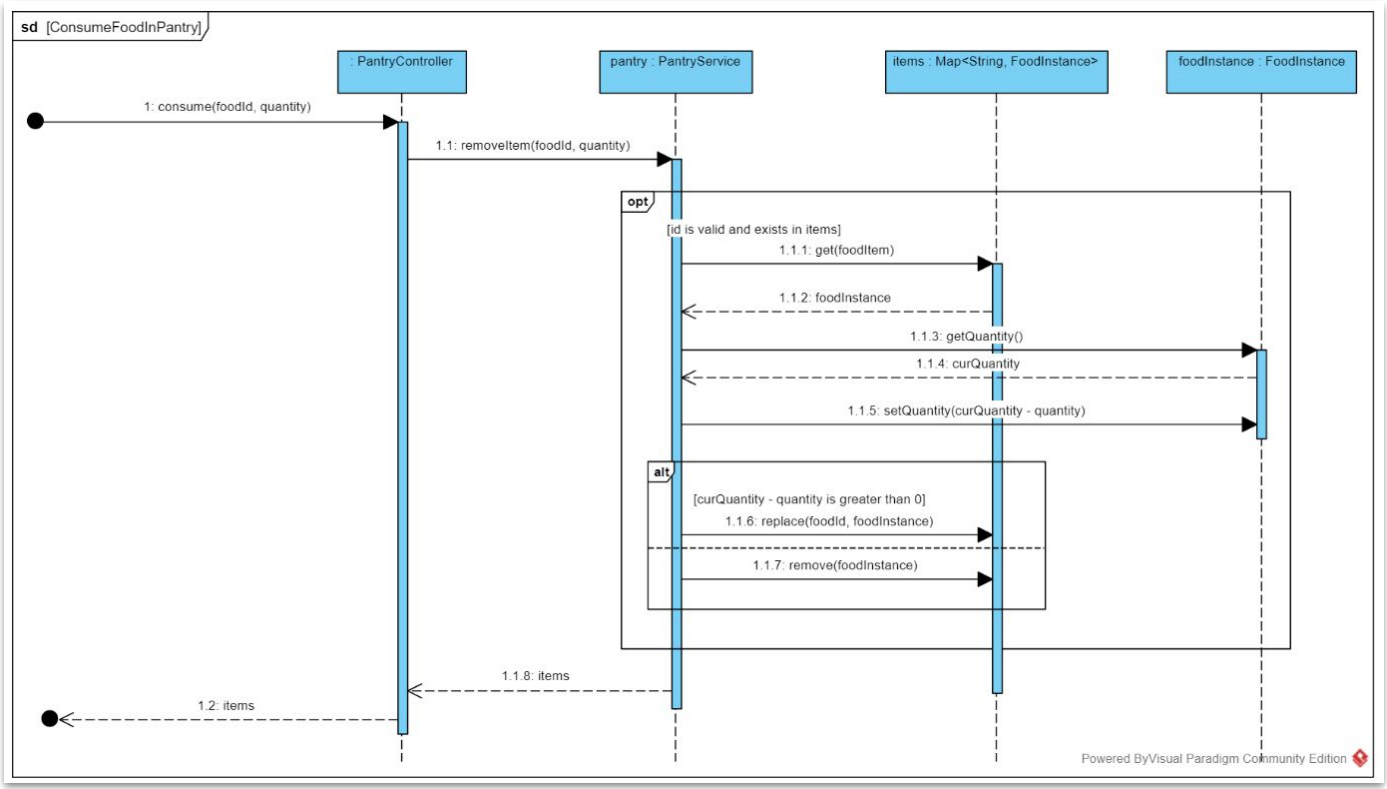
Class Name	RecipeDBAccessor
GRASP Pattern	Low Coupling
Description	This database accessor helps separate the service from the database and keeps the services separated from each other.

Example GRASP

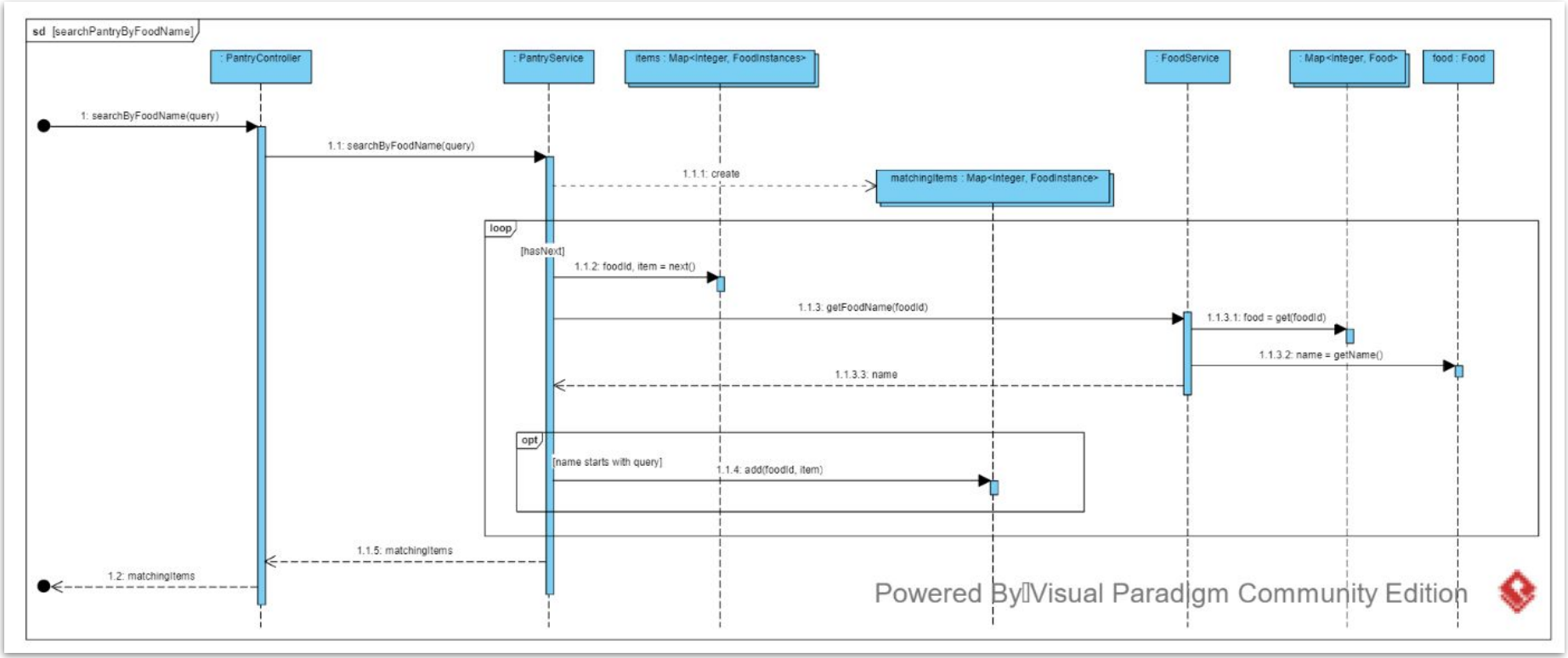
Sequence Diagrams



Example Sequence Diagram: Produce Cooked Recipe



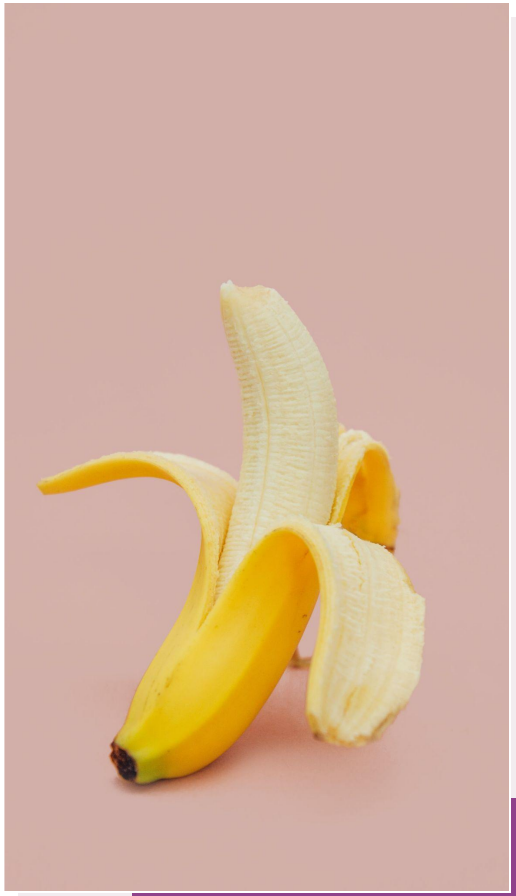
Example Sequence Diagram: Consume Food In Pantry



Example Sequence Diagram: Search Pantry

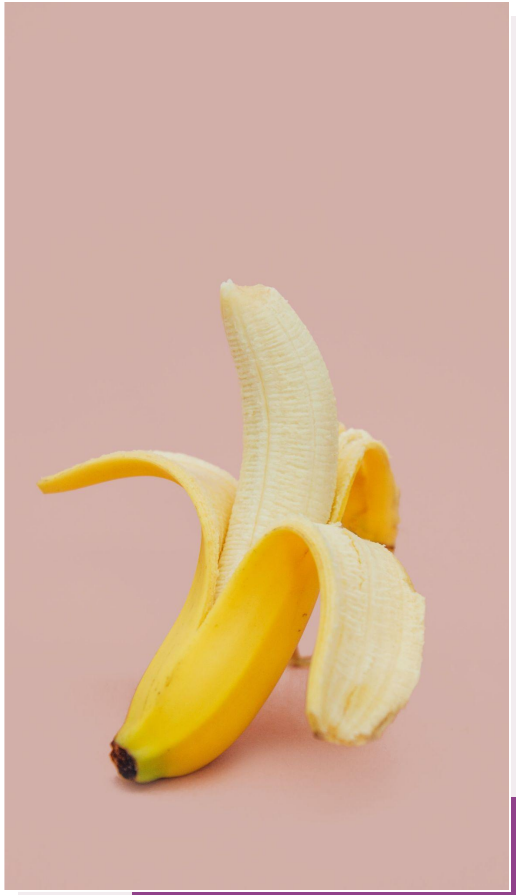
Test Coverage Plan

Our Test Coverage Plan is designed to prescribe the scope, approach, and high-level overview of all testing activities of the FoodPants project. The plan identifies the items to be tested, the features to be tested, and the types of testing to be performed. Unit testing, integration testing, and system testing will be performed using JUnit to ensure code is implemented correctly and robustly and analyzing the UI to ensure that the interfaces among the subsystems operate correctly and all requirements are met for the user.

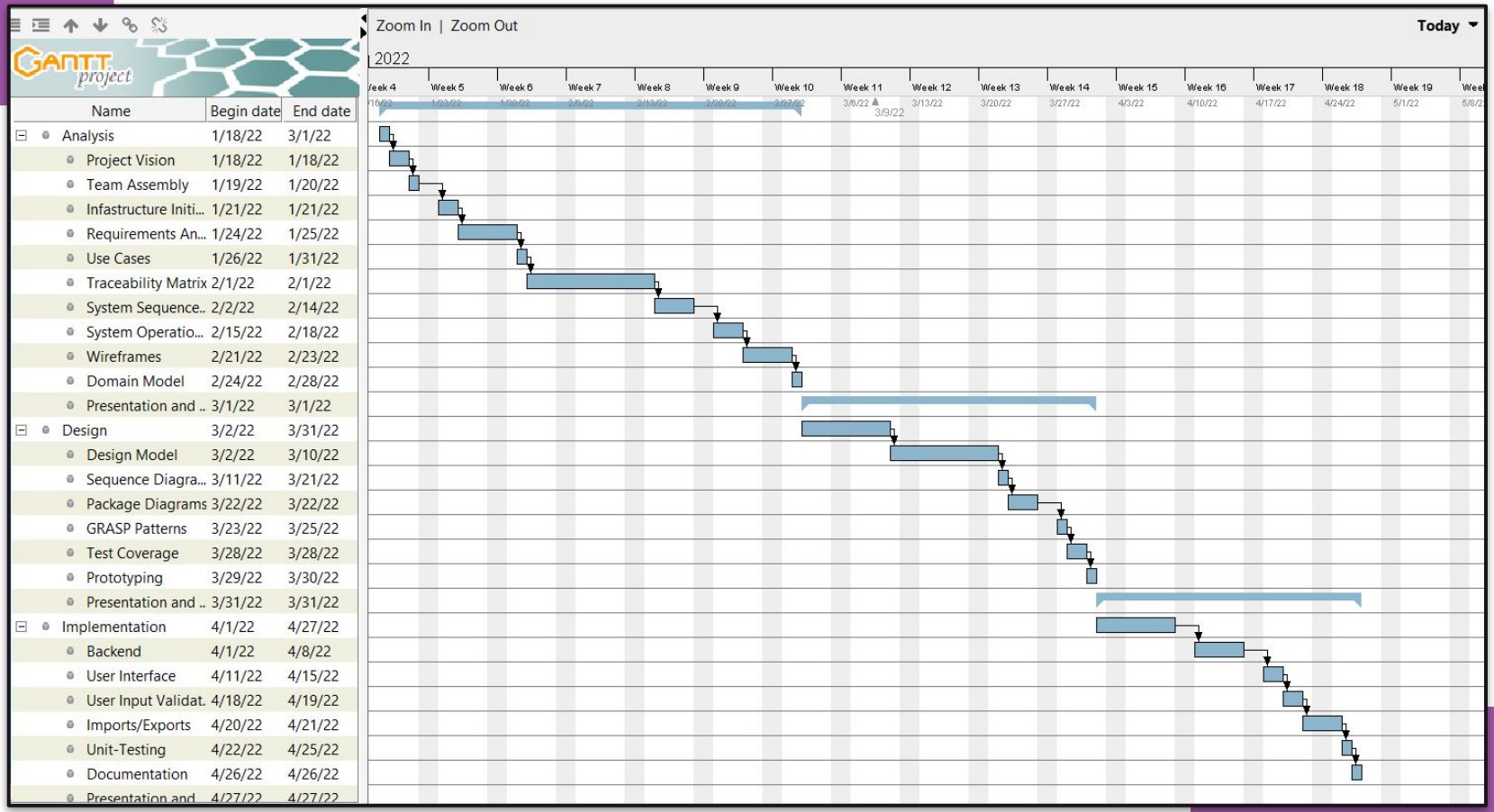


Test Coverage Plan (Cont.)

(1) PANTRY USE CASES (Scope: Pantry system):
- 1.1 Manage pantry (add/edit/remove items manually)
    - Test successful adding of food item (Expected Result: food item should appear in pantry with correct quantity and name)
    - Test successful editing of food item details (Expected Result: food item name or quantity should appear updated to match new entered details)
    - Test successful removal of food items (Expected Result: food item should no longer appear in pantry)
    - Test adding of food item with missing details (Expected Result: program should wait until user enters all missing details or exits without adding)
    - Test adding of food item that already exists (Expected Result: program should add new quantity to already existing quantity of food item that is already in the pantry)
    - Test adding of food item that has new food type (Expected Result: program should prompt use to add a new food type)
- 1.2 Search pantry (by name of food type)
    - Test searching pantry for specific food item (Expected Result: food item with name containing user's query should appear in results)
    - Test searching pantry for query that does not match any food item (Expected Result: program should tell user no match exists)
    - Test searching pantry with query that matches multiple food items (Expected Result: all food items with name containing user's query should appear in results)
- 1.3 Consume specific item (add to nutrition log)
    - Test successful consumption of 1 of food item in pantry (Expected Result: food item should be updated with decremented quantity)
    - Test successful consumption of more than 1 of a food item in pantry (Expected Result: food item should be updated with previous quantity – amount consumed)
    - Test consumption of quantity amount of food item in pantry (Expected Result: quantity left is 0 meaning food item should be removed from pantry)
    - Test consumption of more than quantity amount of food item in pantry (Expected Result: negative quantity left meaning food item should be removed from pantry)



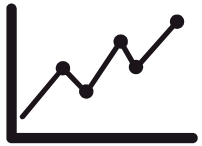
Updated Gantt Project Plan



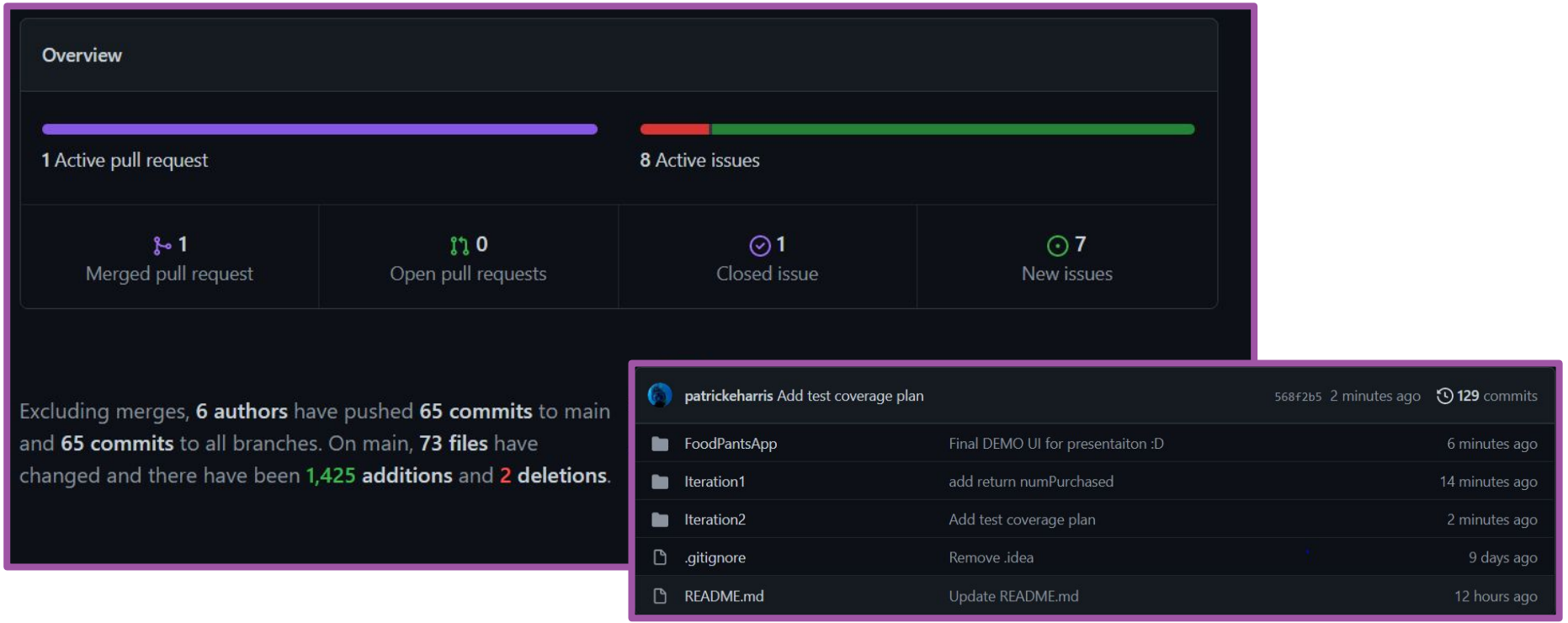
Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/20/22
Infastructure Initi...	1/21/22	1/21/22
Requirements An...	1/24/22	1/25/22
Use Cases	1/26/22	1/31/22
Traceability Matrix	2/1/22	2/1/22
System Sequence..	2/2/22	2/14/22
System Operatio...	2/15/22	2/18/22
Wireframes	2/21/22	2/23/22
Domain Model	2/24/22	2/28/22
Presentation and ..	3/1/22	3/1/22
Design	3/2/22	3/31/22
Design Model	3/2/22	3/10/22
Sequence Diagra...	3/11/22	3/21/22
Package Diagrams	3/22/22	3/22/22
GRASP Patterns	3/23/22	3/25/22
Test Coverage	3/28/22	3/28/22
Prototyping	3/29/22	3/30/22
Presentation and ..	3/31/22	3/31/22
Implementation	4/1/22	4/27/22
Backend	4/1/22	4/8/22
User Interface	4/11/22	4/15/22
User Input Validat.	4/18/22	4/19/22
Imports/Exports	4/20/22	4/21/22
Unit-Testing	4/22/22	4/25/22
Documentation	4/26/22	4/26/22
Presentation and	4/27/22	4/27/22

160+ hours

Tracked via Chronos in Trello



Users	Σ	09 Wed	10 Thu	11 Fri	12 Sat	13 Sun	14 Mon	15 Tue	16 Wed	17 Thu	18 Fri	19 Sat	20 Sun	21 Mon	22 Tue	23 Wed	24 Thu	25 Fri	26 Sat	27 Sun	28 Mon	29 Tue	30 Wec
Austin_Huizinga1	36h 34m							4h 0m						1h 0m							4h 20m		8h 30m
Daniel Luper	33h 8m				1h 15m								21m						1h 20m			6h 50m	11h 3m
Kurt_Wokoek1	22h 34m																		1h 0m	48m	1h 30m	2h 15m	3h 49m
Patrick_Harris3	25h 30m																		2h 0m			2h 30m	7h 30m
PJ_Wallace1	25h 0m															1h 30m			1h 30m		2h 0m	3h 30m	
Luka_Lelovic1	25h 0m														4h 0m				1h 30m	2h 30m	1h 15m	2h 30m	
		0m	0m	0m	1h 15m	0m	0m	4h 0m	0m	0m	0m	0m	21m	1h 0m	4h 0m	1h 30m	0m	0m	7h 20m	3h 18m	9h 5m	17h 35m	30h 5



Git Insights

Issue Tracking System

7 Open ✓ 5 Closed Author Label Projects

System LookAndFeel within swing interferes with images due to improper scaling on HighDPI resolutions bug
#13 opened 14 seconds ago by austinth127

Document Issues/Communication
#12 opened yesterday by kfwokoek

Update SD's
#11 opened yesterday by kfwokoek

Consistency help wanted
#10 opened 2 days ago by austinth127

Iteration 2
#9 opened 8 days ago by lukalelovic

Page Forms enhancement
#8 opened 8 days ago by lukalelovic

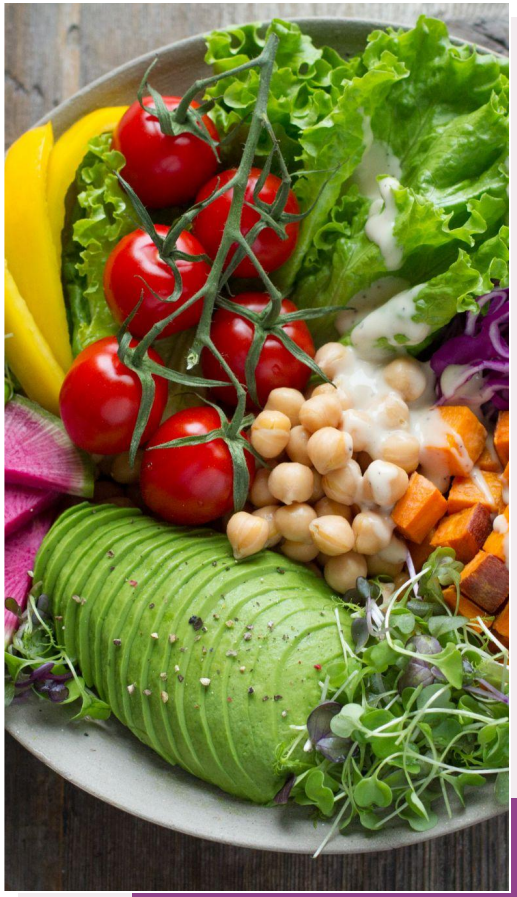
Controllers enhancement help wanted
#7 opened 8 days ago by lukalelovic

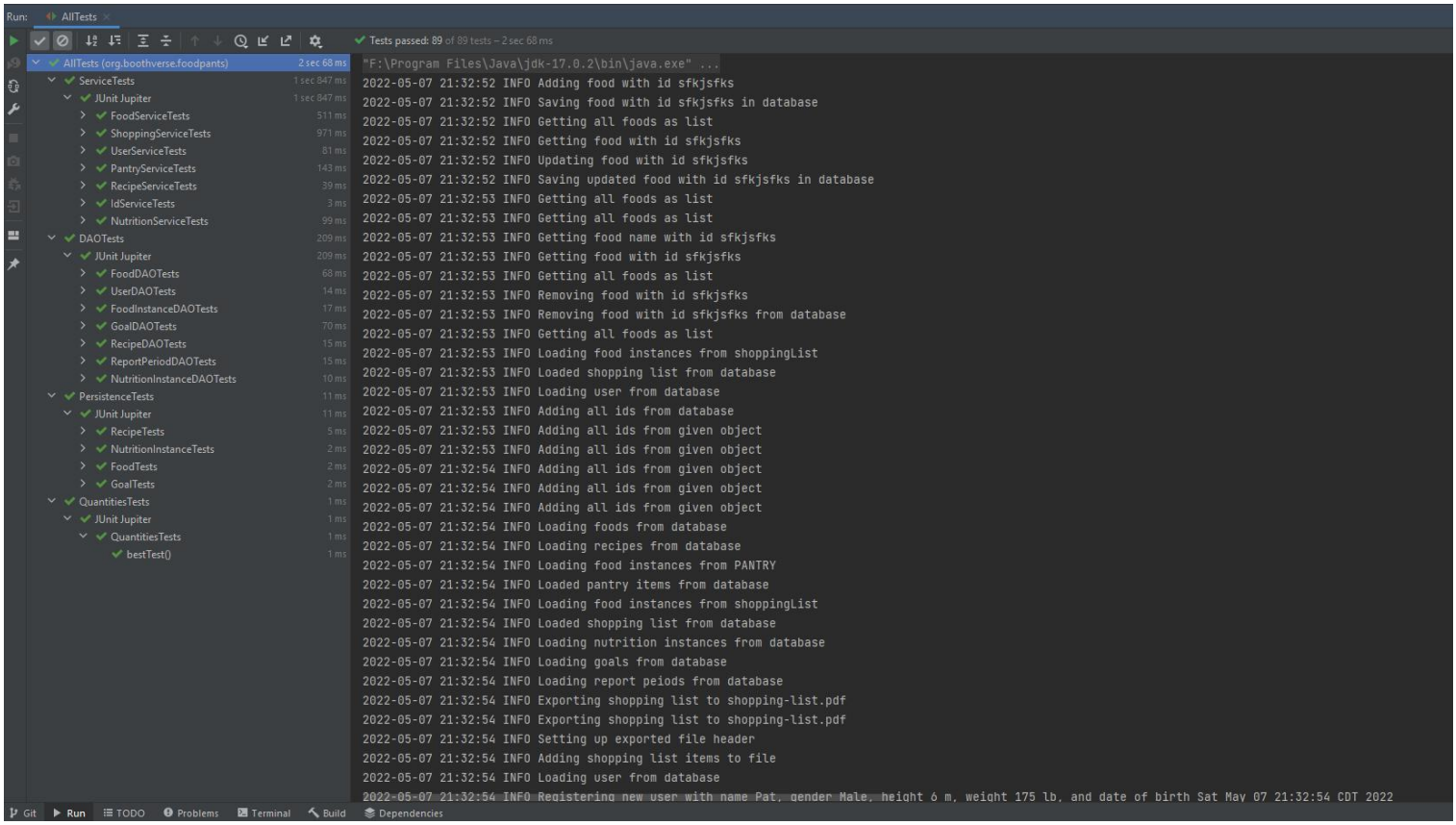
Q/A

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash




Run: AllTests
Tests passed: 89 of 89 tests – 2 sec 68 ms
AllTests (org.boothverse.foodpants) 2 sec 68 ms
ServiceTests 1 sec 847 ms
JUnit Jupiter 1 sec 847 ms
FoodServiceTests 511 ms
ShoppingServiceTests 971 ms
UserServiceTests 81 ms
PantryServiceTests 143 ms
RecipeServiceTests 39 ms
IdServiceTests 3 ms
NutritionServiceTests 99 ms
DAOTests 209 ms
JUnit Jupiter 209 ms
FoodDAOTests 68 ms
UserDAOTests 14 ms
FoodInstanceDAOTests 17 ms
GoalDAOTests 70 ms
RecipeDAOTests 15 ms
ReportPeriodDAOTests 15 ms
NutritionInstanceDAOTests 10 ms
PersistenceTests 11 ms
JUnit Jupiter 11 ms
RecipeTests 5 ms
NutritionInstanceTests 2 ms
FoodTests 2 ms
GoalTests 2 ms
QuantitiesTests 1 ms
JUnit Jupiter 1 ms
QuantitiesTests 1 ms
bestTest() 1 ms
"F:\Program Files\Java\jdk-17.0.2\bin\java.exe" ...
2022-05-07 21:32:52 INFO Adding food with id sfkjsfks
2022-05-07 21:32:52 INFO Saving food with id sfkjsfks in database
2022-05-07 21:32:52 INFO Getting all foods as list
2022-05-07 21:32:52 INFO Getting food with id sfkjsfks
2022-05-07 21:32:52 INFO Updating food with id sfkjsfks
2022-05-07 21:32:52 INFO Saving updated food with id sfkjsfks in database
2022-05-07 21:32:53 INFO Getting all foods as list
2022-05-07 21:32:53 INFO Getting all foods as list
2022-05-07 21:32:53 INFO Getting food name with id sfkjsfks
2022-05-07 21:32:53 INFO Getting food with id sfkjsfks
2022-05-07 21:32:53 INFO Getting all foods as list
2022-05-07 21:32:53 INFO Removing food with id sfkjsfks
2022-05-07 21:32:53 INFO Removing food with id sfkjsfks from database
2022-05-07 21:32:53 INFO Getting all foods as list
2022-05-07 21:32:53 INFO Loading food instances from shoppingList
2022-05-07 21:32:53 INFO Loaded shopping list from database
2022-05-07 21:32:53 INFO Loading user from database
2022-05-07 21:32:53 INFO Adding all ids from database
2022-05-07 21:32:53 INFO Adding all ids from given object
2022-05-07 21:32:53 INFO Adding all ids from given object
2022-05-07 21:32:54 INFO Adding all ids from given object
2022-05-07 21:32:54 INFO Adding all ids from given object
2022-05-07 21:32:54 INFO Adding all ids from given object
2022-05-07 21:32:54 INFO Loading foods from database
2022-05-07 21:32:54 INFO Loading recipes from database
2022-05-07 21:32:54 INFO Loading food instances from PANTRY
2022-05-07 21:32:54 INFO Loaded pantry items from database
2022-05-07 21:32:54 INFO Loading food instances from shoppingList
2022-05-07 21:32:54 INFO Loaded shopping list from database
2022-05-07 21:32:54 INFO Loading nutrition instances from database
2022-05-07 21:32:54 INFO Loading goals from database
2022-05-07 21:32:54 INFO Loading report peiods from database
2022-05-07 21:32:54 INFO Exporting shopping list to shopping-list.pdf
2022-05-07 21:32:54 INFO Exporting shopping list to shopping-list.pdf
2022-05-07 21:32:54 INFO Setting up exported file header
2022-05-07 21:32:54 INFO Adding shopping list items to file
2022-05-07 21:32:54 INFO Loading user from database
2022-05-07 21:32:54 INFO Registering new user with name Pat, gender Male, height 6 m, weight 175 lb, and date of birth Sat May 07 21:32:54 CDT 2022
Git
Run
TODO
Problems
Terminal
Build
Dependencies

18 January 2022 – 31 May 2022

Users	Σ
Austin_Huizinga1	62h 34m
Daniel Luper	69h 41m
K Kurt_Wokoek1	52h 22m
Patrick_Harris3	58h 25m
PJ_Wallace1	48h
Luka_Lelovic1	57h 40m

FoodPantsUpdated

May 7, 2022

http://

Project manager	
Project dates	Jan 18, 2022 - May 4, 2022
Completion	0%
Tasks	28
Resources	4

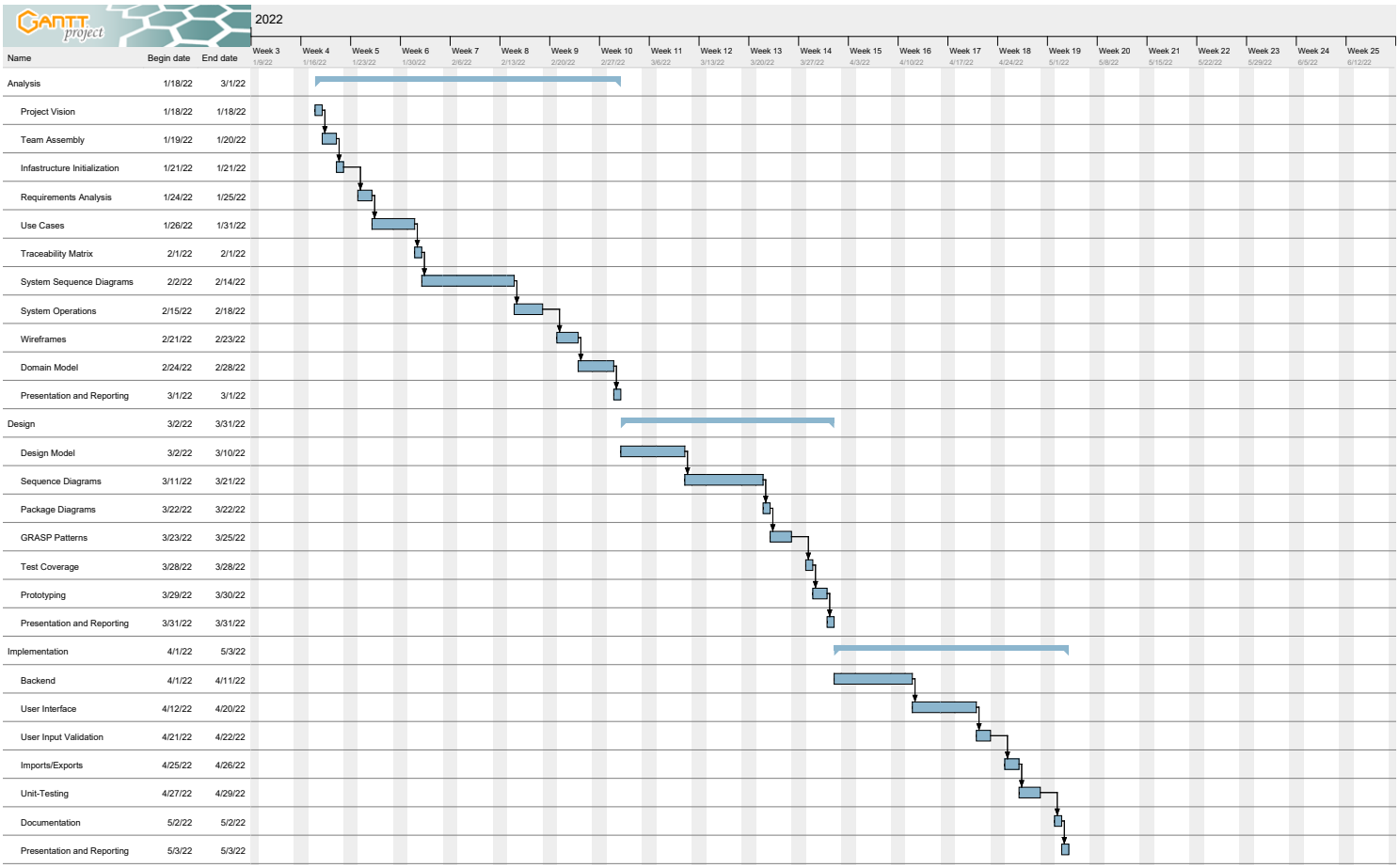
Tasks

Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/20/22
Infastructure Initialization	1/21/22	1/21/22
Requirements Analysis	1/24/22	1/25/22
Use Cases	1/26/22	1/31/22
Traceability Matrix	2/1/22	2/1/22
System Sequence Diagrams	2/2/22	2/14/22
System Operations	2/15/22	2/18/22
Wireframes	2/21/22	2/23/22
Domain Model	2/24/22	2/28/22
Presentation and Reporting	3/1/22	3/1/22
Design	3/2/22	3/31/22
Design Model	3/2/22	3/10/22
Sequence Diagrams	3/11/22	3/21/22
Package Diagrams	3/22/22	3/22/22
GRASP Patterns	3/23/22	3/25/22
Test Coverage	3/28/22	3/28/22
Prototyping	3/29/22	3/30/22
Presentation and Reporting	3/31/22	3/31/22
Implementation	4/1/22	5/3/22
Backend	4/1/22	4/11/22
User Interface	4/12/22	4/20/22
User Input Validation	4/21/22	4/22/22
Imports/Exports	4/25/22	4/26/22
Unit-Testing	4/27/22	4/29/22
Documentation	5/2/22	5/2/22
Presentation and Reporting	5/3/22	5/3/22

Resources

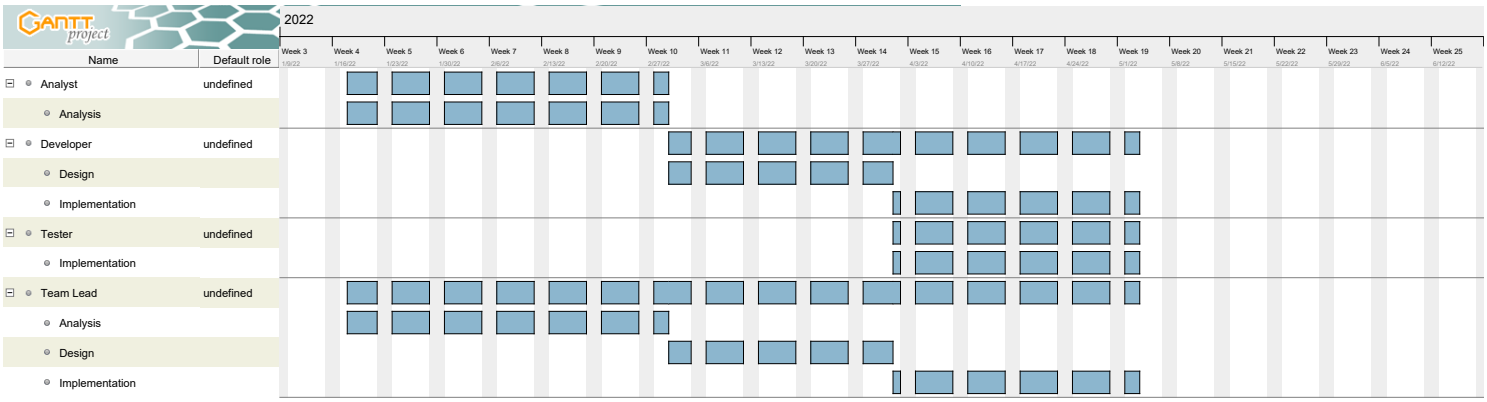
Name	Default role
Analyst	undefined
Developer	undefined
Tester	undefined
Team Lead	undefined

Gantt Chart



Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/20/22
Infastructure Initialization	1/21/22	1/21/22
Requirements Analysis	1/24/22	1/25/22
Use Cases	1/26/22	1/31/22
Traceability Matrix	2/1/22	2/1/22
System Sequence Diagrams	2/2/22	2/14/22
System Operations	2/15/22	2/18/22
Wireframes	2/21/22	2/23/22
Domain Model	2/24/22	2/28/22
Presentation and Reporting	3/1/22	3/1/22
Design	3/2/22	3/31/22
Design Model	3/2/22	3/10/22
Sequence Diagrams	3/11/22	3/21/22
Package Diagrams	3/22/22	3/22/22
GRASP Patterns	3/23/22	3/25/22
Test Coverage	3/28/22	3/28/22
Prototyping	3/29/22	3/30/22
Presentation and Reporting	3/31/22	3/31/22
Implementation	4/1/22	5/3/22
Backend	4/1/22	4/11/22
User Interface	4/12/22	4/20/22
User Input Validation	4/21/22	4/22/22
Imports/Exports	4/25/22	4/26/22
Unit-Testing	4/27/22	4/29/22
Documentation	5/2/22	5/2/22
Presentation and Reporting	5/3/22	5/3/22

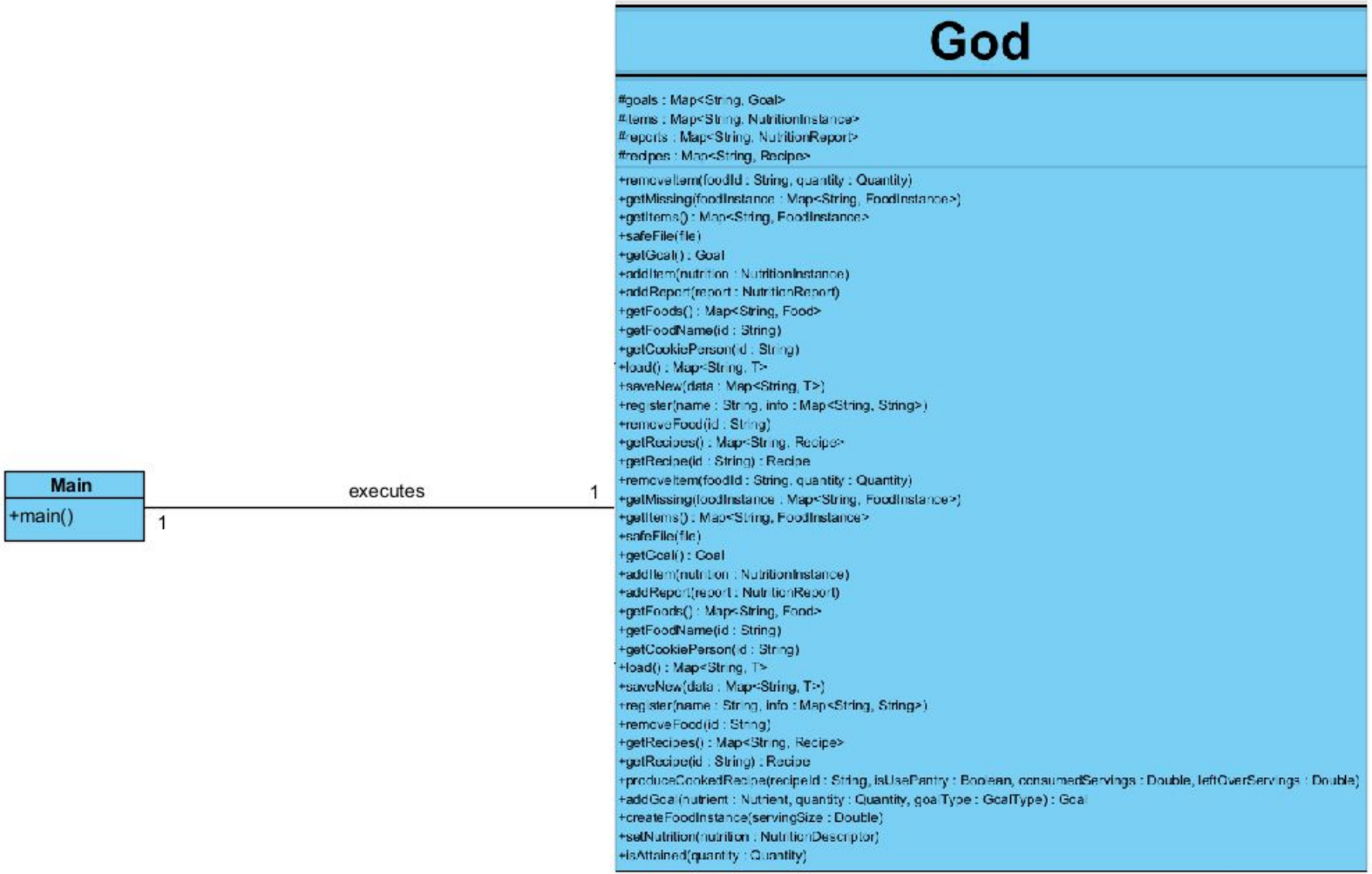
Resources Chart



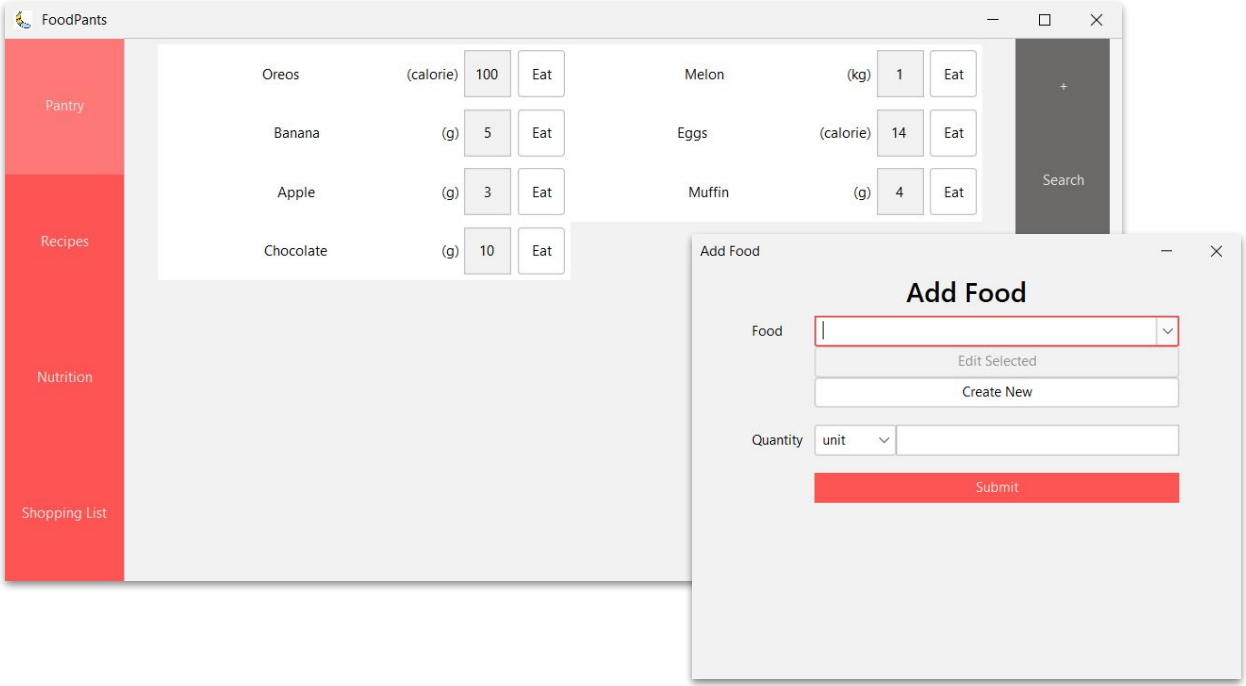
FoodPants

Iteration III

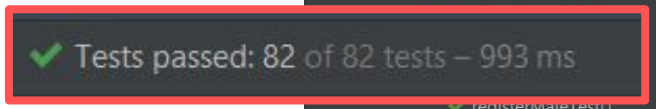
Class Diagram


Main
+main()
executes
1
1
God
#goals : Map<String, Goal>
#items : Map<String, NutritionInstance>
#reports : Map<String, NutritionReport>
#recipes : Map<String, Recipe>
+removeItem(foodId : String, quantity : Quantity)
+getMissing(foodInstance : Map<String, FoodInstance>)
+getItems() : Map<String, FoodInstance>
+safeFile(file)
+getGoal() : Goal
+addItem(nutrition : NutritionInstance)
+addReport(report : NutritionReport)
+getFoods() : Map<String, Food>
+getFoodName(id : String)
+getCookiePerson(id : String)
+load() : Map<String, T>
+saveNew(data : Map<String, T>)
+register(name : String, info : Map<String, String>)
+removeFood(id : String)
+getRecipes() : Map<String, Recipe>
+getRecipe(id : String) : Recipe
+removeItem(foodId : String, quantity : Quantity)
+getMissing(foodInstance : Map<String, FoodInstance>)
+getItems() : Map<String, FoodInstance>
+safeFile(file)
+getGoal() : Goal
+addItem(nutrition : NutritionInstance)
+addReport(report : NutritionReport)
+getFoods() : Map<String, Food>
+getFoodName(id : String)
+getCookiePerson(id : String)
+load() : Map<String, T>
+saveNew(data : Map<String, T>)
+register(name : String, info : Map<String, String>)
+removeFood(id : String)
+getRecipes() : Map<String, Recipe>
+getRecipe(id : String) : Recipe
+produceCookedRecipe(recipeId : String, isUsePantry : Boolean, consumedServings : Double, leftOverServings : Double)
+addGoal(nutrient : Nutrient, quantity : Quantity, goalType : GoalType) : Goal
+createFoodInstance(servingSize : Double)
+setNutrition(nutrition : NutritionDescriptor)
+isAttained(quantity : Quantity)

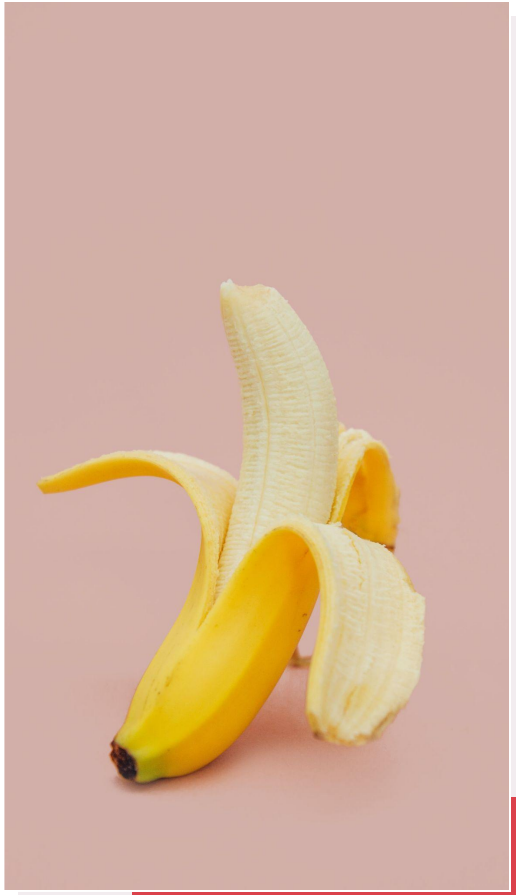
Live Demonstration!



JUnit


Tests passed: 82 of 82 tests – 993 ms

AllTests (org.boothverse.foodpants) 993 ms
ServiceTests 815 ms
JUnit Jupiter 815 ms
FoodServiceTests 173 ms
addFoodTest() 122 ms
getFoodTest() 6 ms
editFoodTest() 31 ms
getFoodNameTest() 1 ms
getFoodsTest() 0 ms
removeFoodTest() 13 ms
userIsFemaleFalseTest() 0 ms
overrideRegisterTest() 9 ms
userIsFemaleTrueTest() 0 ms
getBodyWeightKgTest() 2 ms
getHeightCmTest() 1 ms
PantryServiceTests 146 ms
getItemsEmptyTest() 1 ms
addItemTest() 54 ms
addItemsTest() 43 ms
editItemTest() 14 ms
removeItemTest() 14 ms
getMissingNoMissing() 1 ms
getMissingWrongQuantity() 1 ms
getMissingOverQuantity() 1 ms
containsTrueTest() 1 ms
containsFalseTest() 0 ms
containsFalseTest() 0 ms
searchByFoodNameTrueTes 2 ms
searchByFoodNameFalseTe 1 ms
RecipeServiceTests 45 ms
getRecipesEmptyTest() 1 ms
getRecipeThrowsTest() 2 ms
addRecipeTest() 18 ms
getRecipeTest() 1 ms
getIngredientsEmptyTest() 0 ms
editRecipeTest() 14 ms
getIngredientsTest() 1 ms
getRecipeByNameMatching 2 ms
getRecipeByNameStartsWit 0 ms
getRecipeByNameNoneTes 1 ms
getRecipeByIngredientsTrue 3 ms
getRecipeByIngredientsFals 1 ms
getRecipeByIngredientsWrc 1 ms
IdServiceTests 3 ms
NutritionServiceTests 93 ms
DAOTests 165 ms
PersistenceTests 13 ms
JUnit Jupiter 13 ms
RecipeTests 3 ms
NutritionInstanceTests 10 ms
FoodTests 0 ms
GoalTests 0 ms
checkGoalAttainedNullQuar 0 ms
checkGoalNotAttainedMaxi 0 ms
checkGoalAttainedMinimize 0 ms
checkGoalAttainedMaximiz 0 ms
checkGoalNotAttainedMinir 0 ms

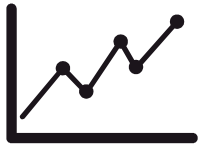


Testing

Use Case #	Use Case Description	Tests	Expected Result	Class Implemented In	Method
1.1	Manage pantry (add/edit/remove items manually)	Successful adding of food item	Food item should appear in pantry	PantryServiceTests	addItemTest()
		Successful editing of food item details	Food item name or quantity should appear updated	PantryServiceTests	editItemTest()
		Successful removal of food items	Food item should no longer appear in pantry	PantryServiceTests	removeItemTest()
		Adding of food item with missing details	Program should return an error to the user	PantryServiceTests	addIncompleteItemTest()
		Adding of food item that already exists	Program should increase quantity of existing food item	PantryServiceTests	addExistingItemTest()
		Adding of food item that has new food type	Program should prompt use to add a new food type	PantryServiceTests	addNewItemTest()
		Editing of food item to match name of another food item	Quantity of existing food item is updated	PantryServiceTests	editMatchingItemTest()

300+ hours!

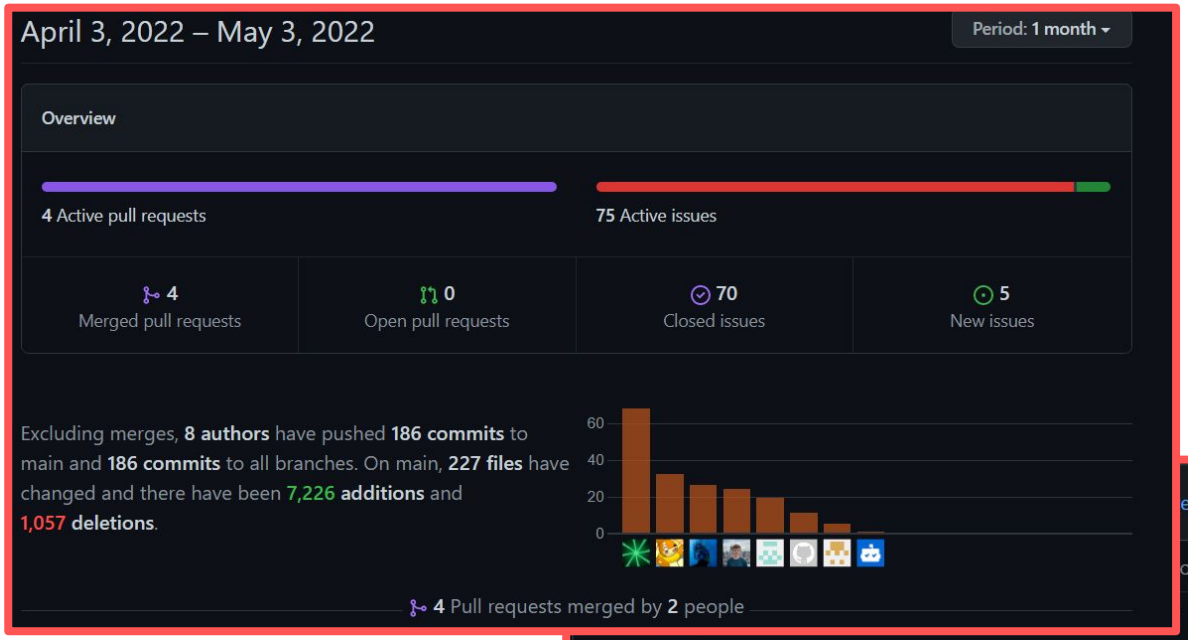
Tracked via Chronos in Trello

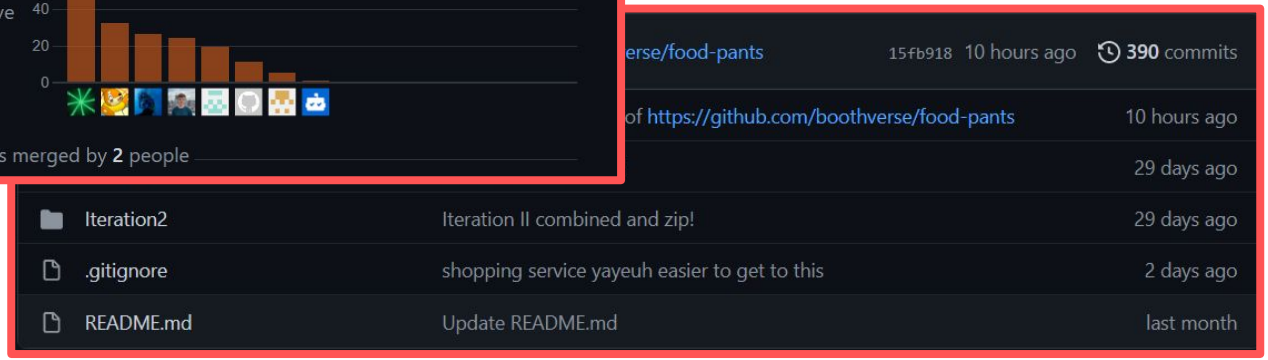


Users	Σ	19 Tue	20 Wed	21 Thu	22 Fri	23 Sat	24 Sun	25 Mon	26 Tue	27 Wed	28 Thu	29 Fri	30 Sat	01 Sun	02 Mon
Austin_Huizinga1	62h 34m												10h	2h	12h
Daniel Luper	67h 5m	5h											5h	6h 40m	5h
Kurt_Wokoek1	47h 7m									45m	3h 52m	1h 30m	1h 45m	2h	11h 51m
Patrick_Harris3	50h 25m											3h	5h	5h	11h
PJ_Wallace1	48h			5h			8h				3h	3h		3h	1h
Luka_Lelovic1	50h 30m										2h				7h
		5h	0m	5h	0m	0m	8h	0m	0m	45m	8h 52m	7h 30m	21h 45m	18h 40m	47h 51m

total time spent 325h 42m

Local(America/Chicago) Export Settings





Git Insights

Issue Tracking System

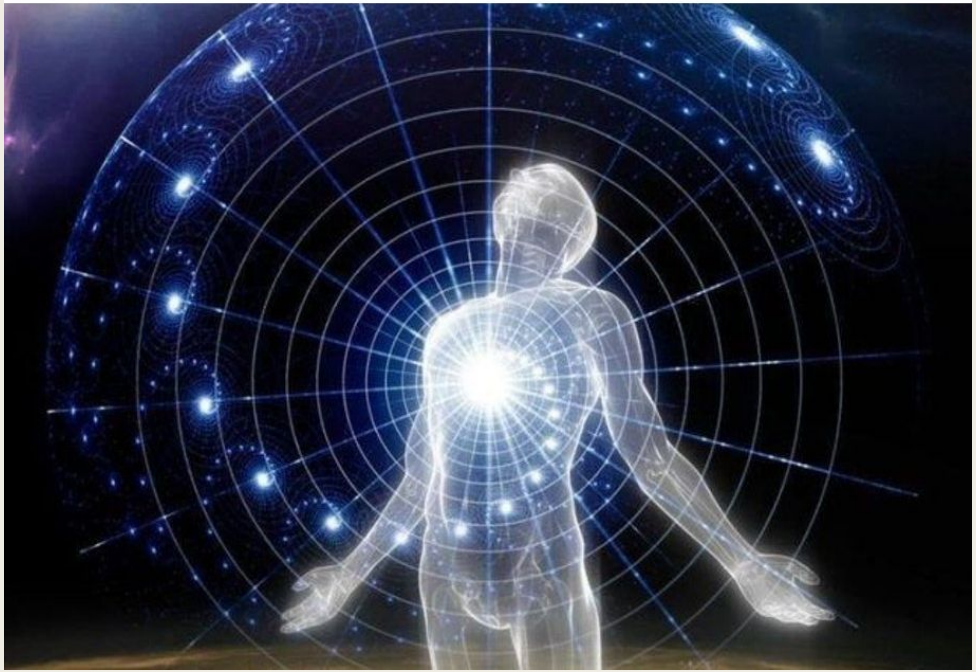
5 Open ✓ 79 Closed Author

- implement search recipes functionality
  #88 opened 13 hours ago by kfwokoek
- implement startup sequence
  #83 opened 13 hours ago by kfwokoek
- Revised Iteration II documentation
  #42 opened 7 days ago by austinth127
- Frontend combine components and cleanup frontend
  #39 opened 12 days ago by austinth127
- Logging
  #34 opened 24 days ago by daniel-luper

Q/A

...??

Me when FoodPants iteration III releases:



Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash