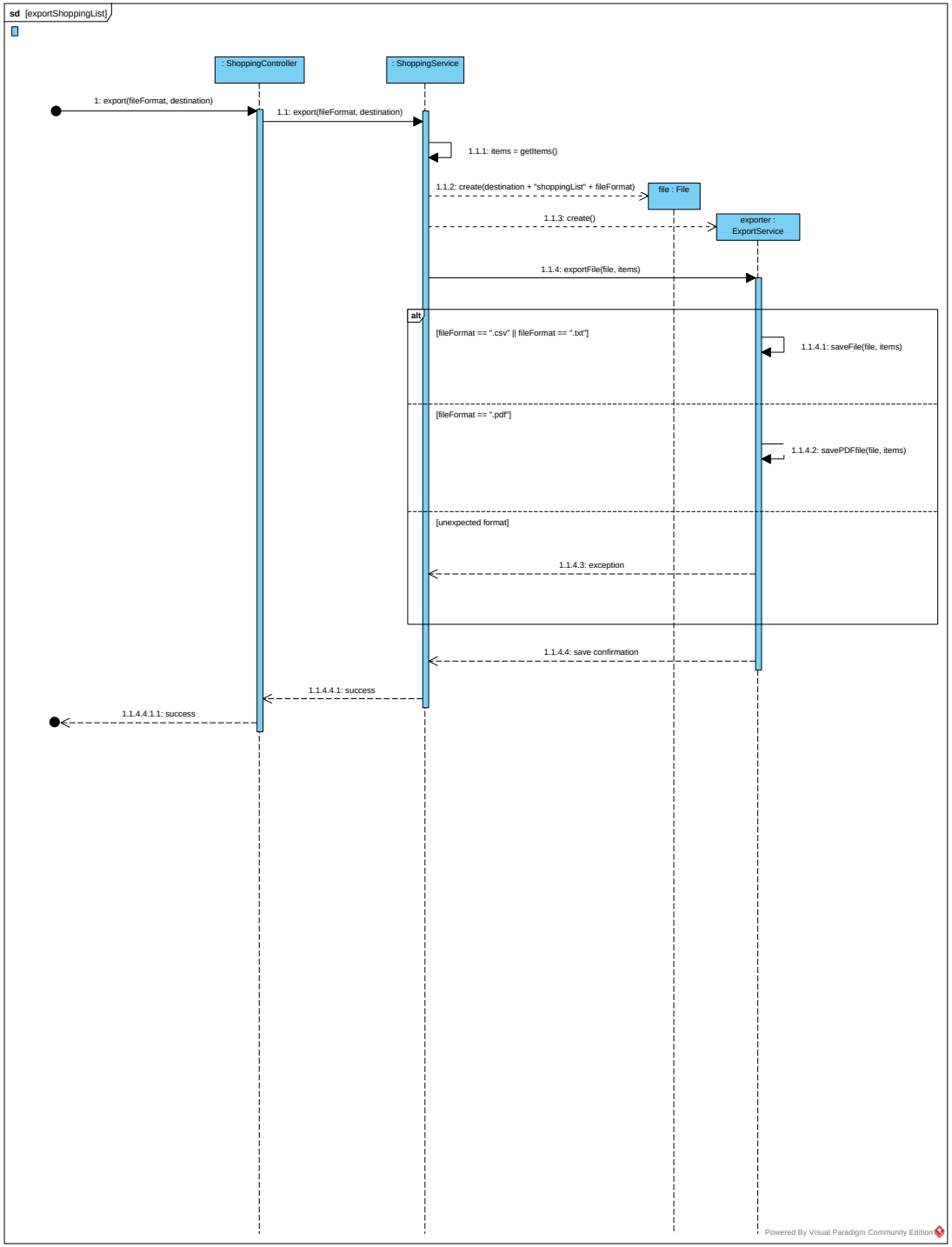
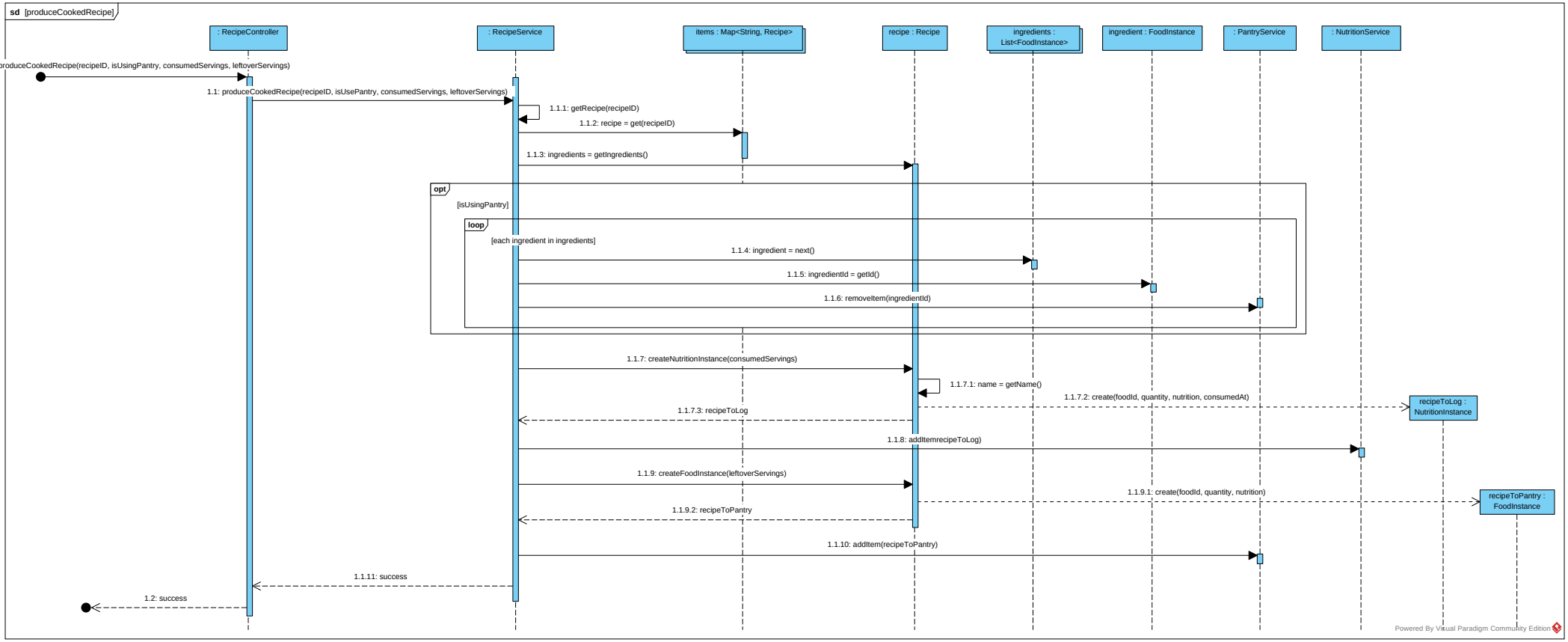
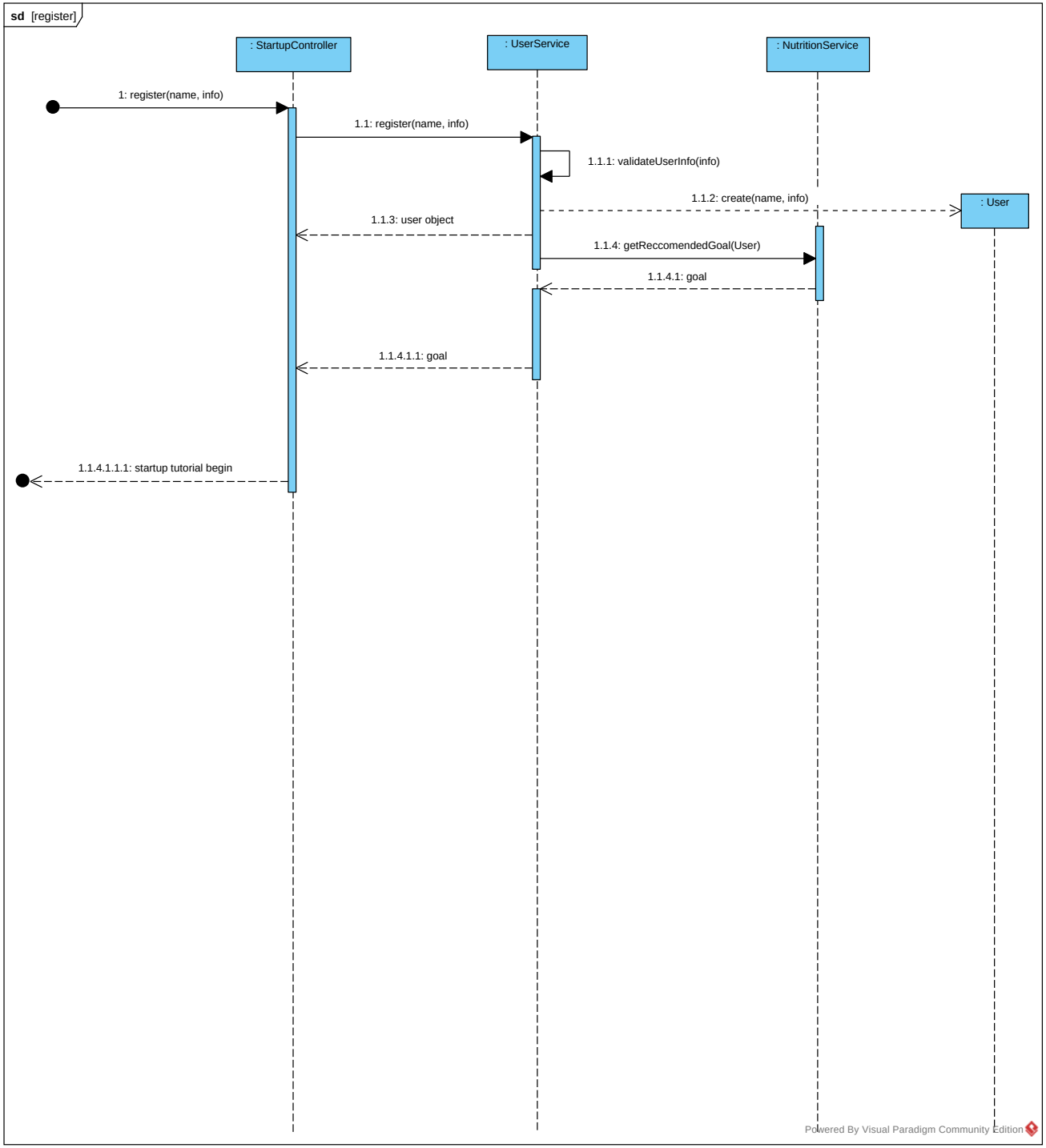
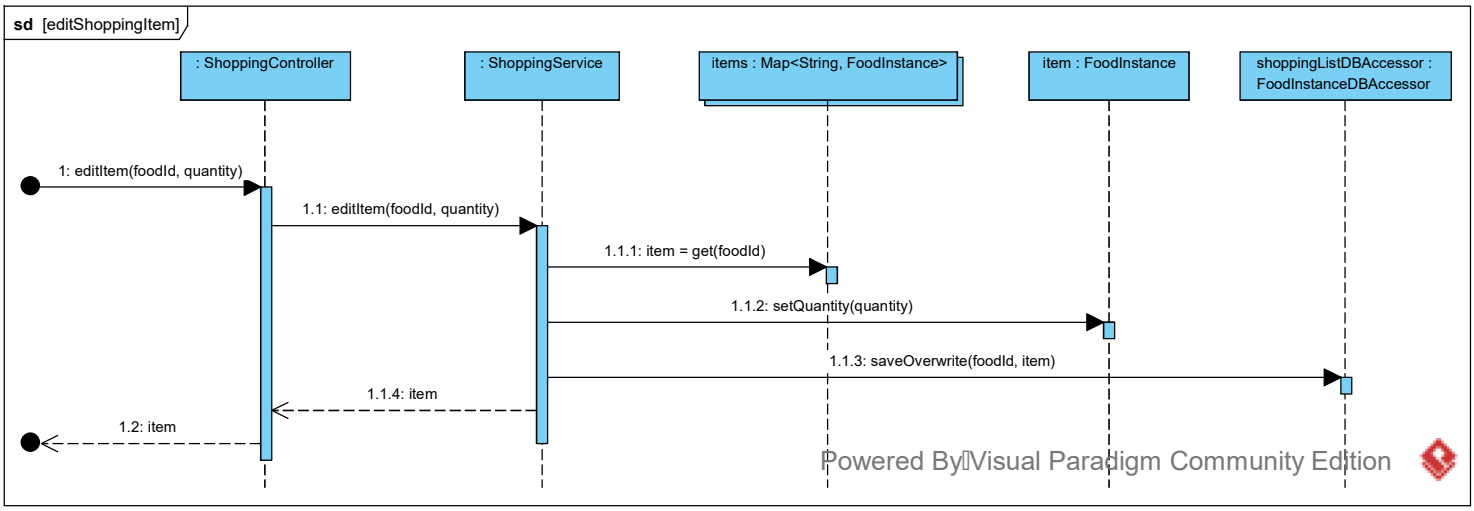
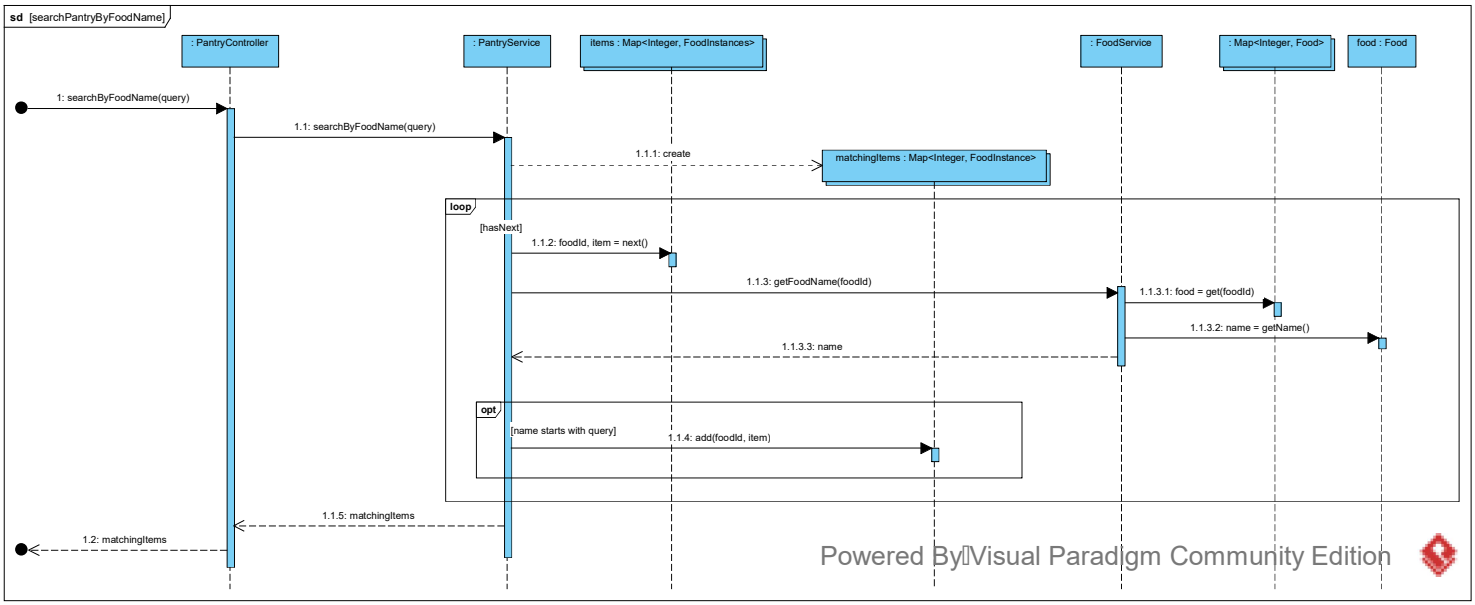
FoodPants
Iteration II

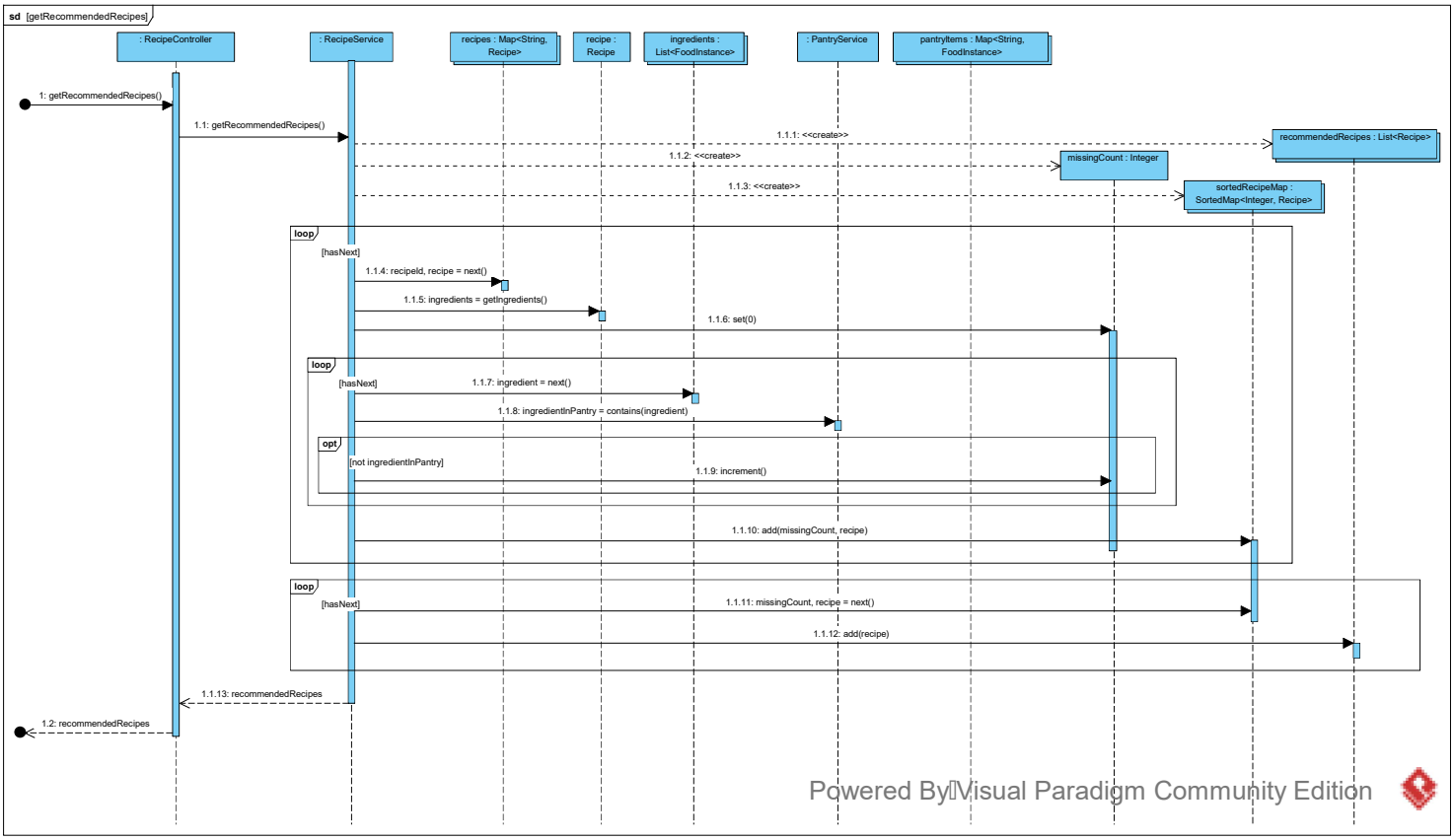

sd [exportShoppingList]
: ShoppingController
: ShoppingService
1: export(fileFormat, destination)
1.1: export(fileFormat, destination)
1.1.1: items = getItems()
1.1.2: create(destination + "shoppingList" + fileFormat)
file : File
1.1.3: create()
exporter : ExportService
1.1.4: exportFile(file, items)
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

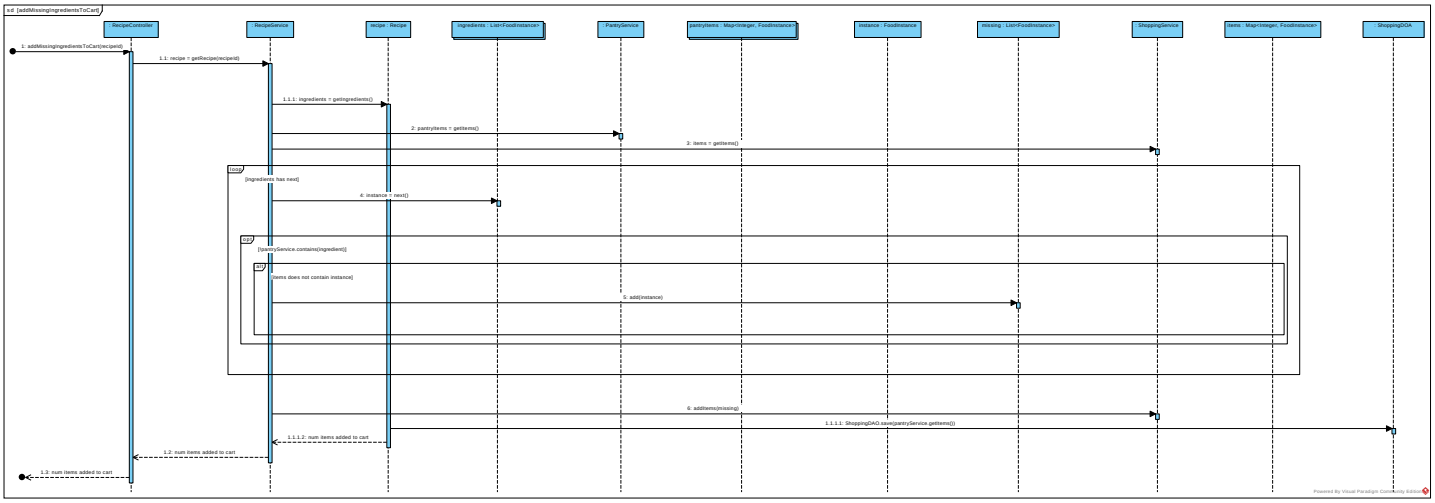

sd [produceCookedRecipe]
: RecipeController
: RecipeService
items : Map<String, Recipe>
recipe : Recipe
ingredients : List<FoodInstance>
ingredient : FoodInstance
: PantryService
: NutritionService
1: produceCookedRecipe(recipeID, isUsingPantry, consumedServings, leftoverServings)
1.1: produceCookedRecipe(recipeID, 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

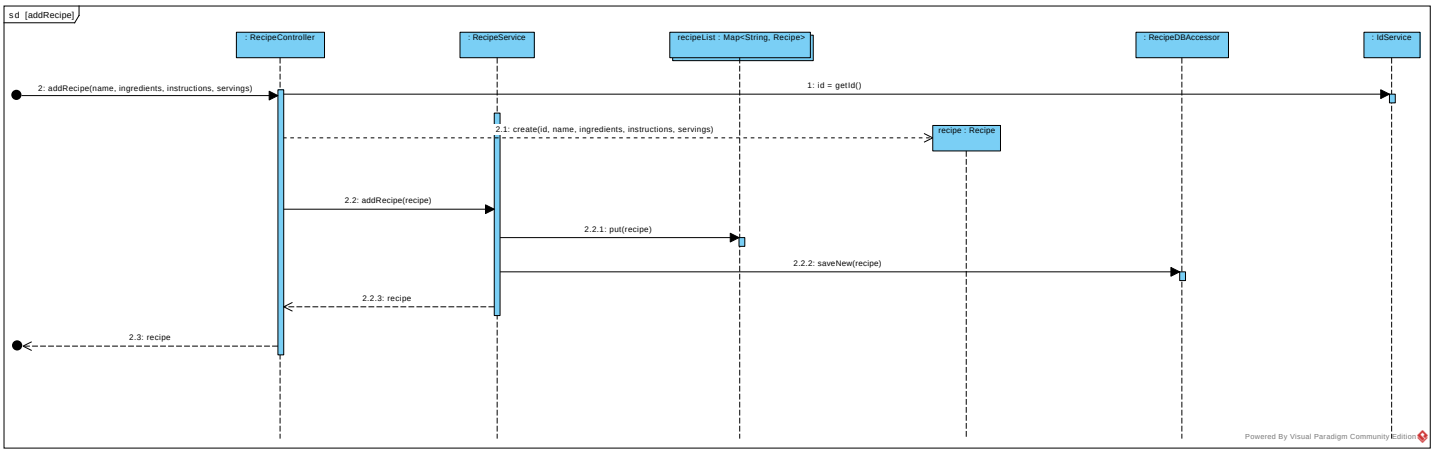

sd [register]
: StartupController
: UserService
: NutritionService
: User
1: register(name, info)
1.1: register(name, info)
1.1.1: validateUserInfo(info)
1.1.2: create(name, info)
1.1.3: user object
1.1.4: getReccomendedGoal(User)
1.1.4.1: goal
1.1.4.1.1: goal
1.1.4.1.1.1: startup tutorial begin
Powered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

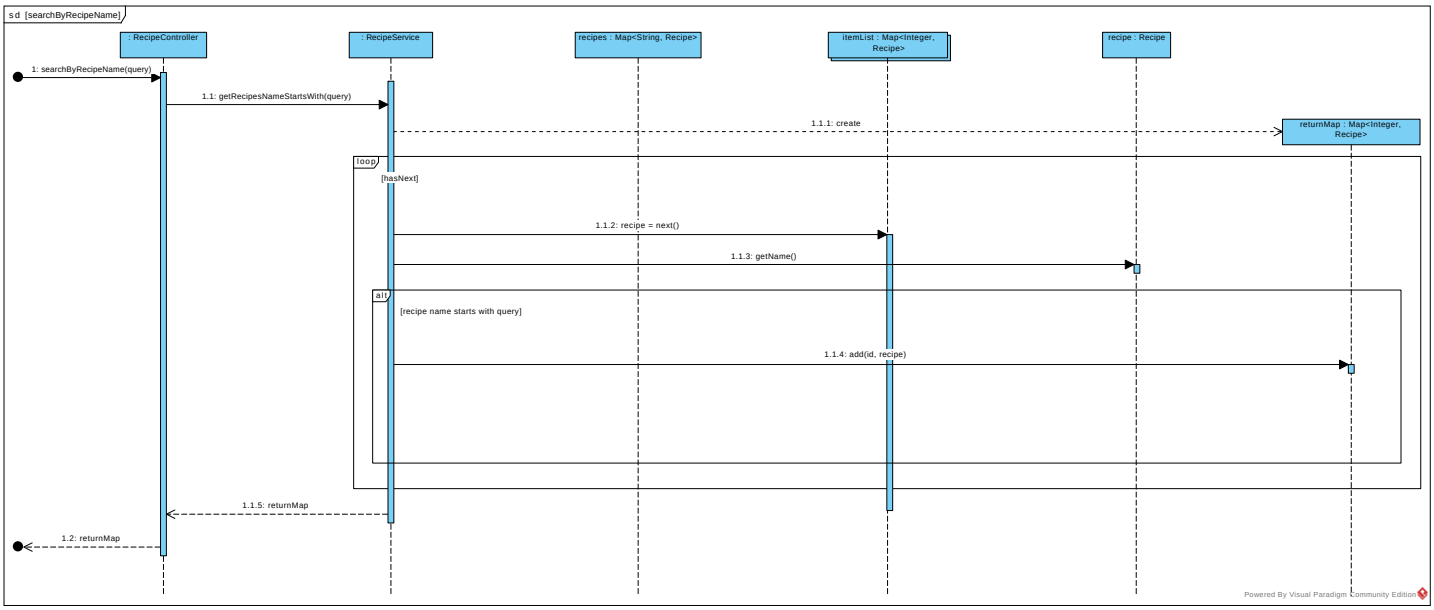

sd [editShoppingItem]
: ShoppingController
: ShoppingService
items : Map<String, FoodInstance>
item : FoodInstance
shoppingListDBAccessor : FoodInstanceDBAccessor
1: editItem(foodId, quantity)
1.1: editItem(foodId, quantity)
1.1.1: item = get(foodId)
1.1.2: setQuantity(quantity)
1.1.3: saveOverwrite(foodId,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

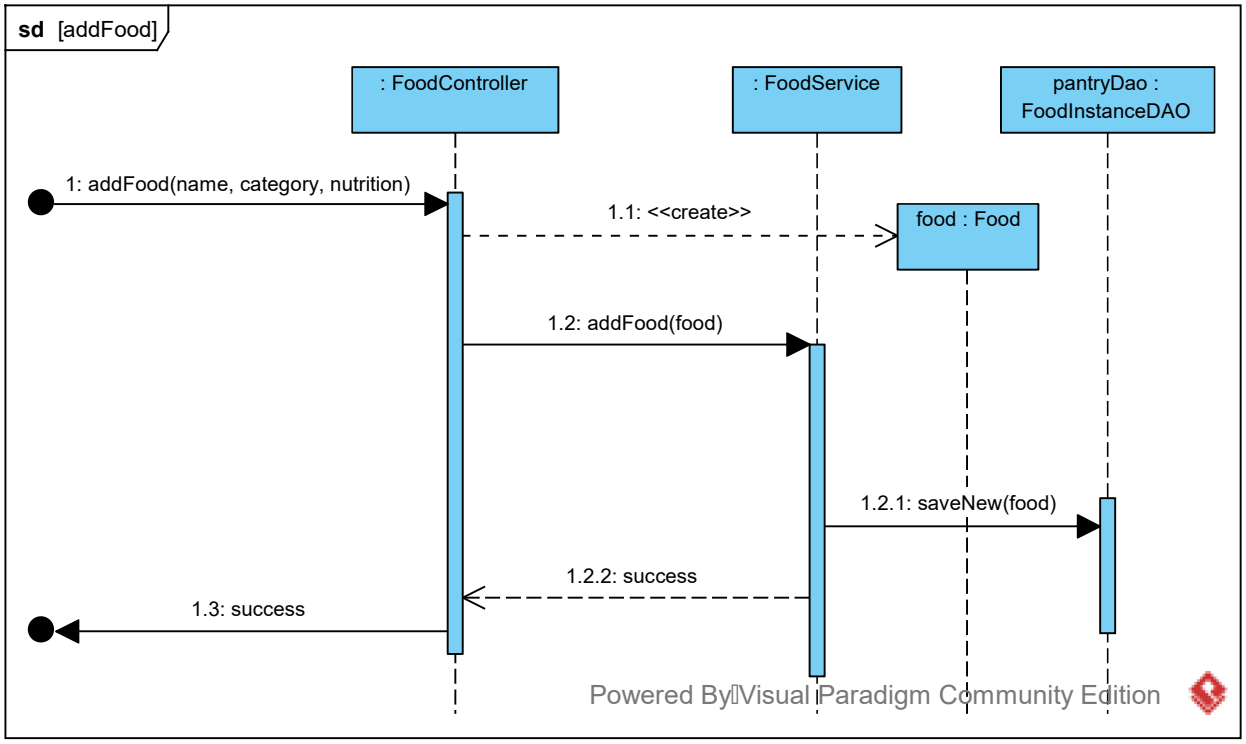

sd [searchPantryByFoodName]
: PantryController
: PantryService
items : Map<Integer, FoodInstances>
: FoodService
: Map<Integer, Food>
food : Food
1: searchByFoodName(query)
1.1: searchByFoodName(query)
1.1.1: create
matchingItems : Map<Integer, FoodInstance>
loop
[hasNext]
1.1.2: foodId, item = next()
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

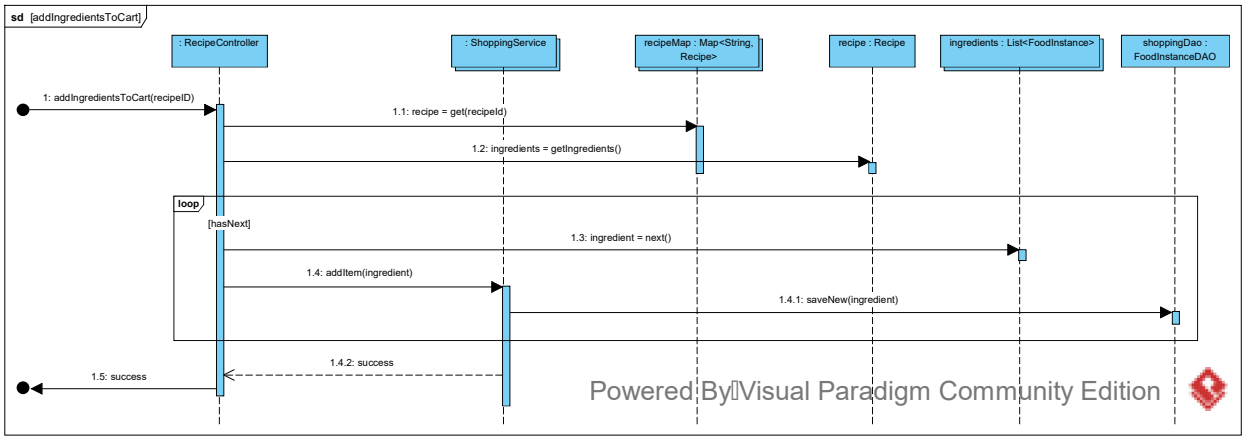

sd [getRecommendedRecipes]
: RecipeController
: RecipeService
recipes : Map<String, Recipe>
recipe : Recipe
ingredients : List<FoodInstance>
: PantryService
pantryItems : Map<String, FoodInstance>
1: getRecommendedRecipes()
1.1: getRecommendedRecipes()
1.1.1: <<create>>
recommendedRecipes : List<Recipe>
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

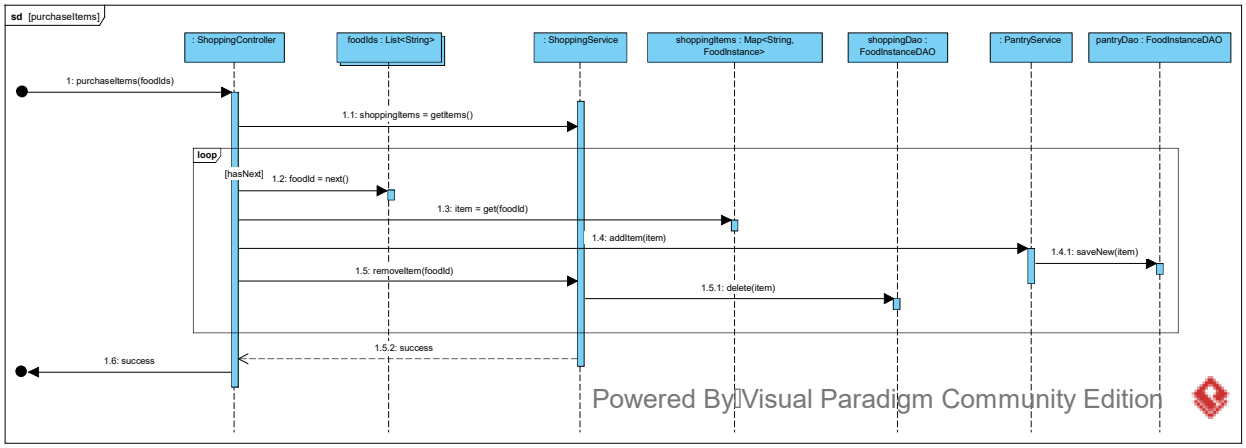

sd [addMissingIngredientsToCart()]
: RecipeController
: RecipeService
recipe : Recipe
ingredients : List<FoodInstance>
: PantryService
pantryItems : Map<Integer, FoodInstance>
instance : FoodInstance
missing : List<FoodInstance>
: ShoppingService
items : Map<Integer, FoodInstance>
: ShoppingDOA
1: addMissingIngredientsToCart(recipeId)
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

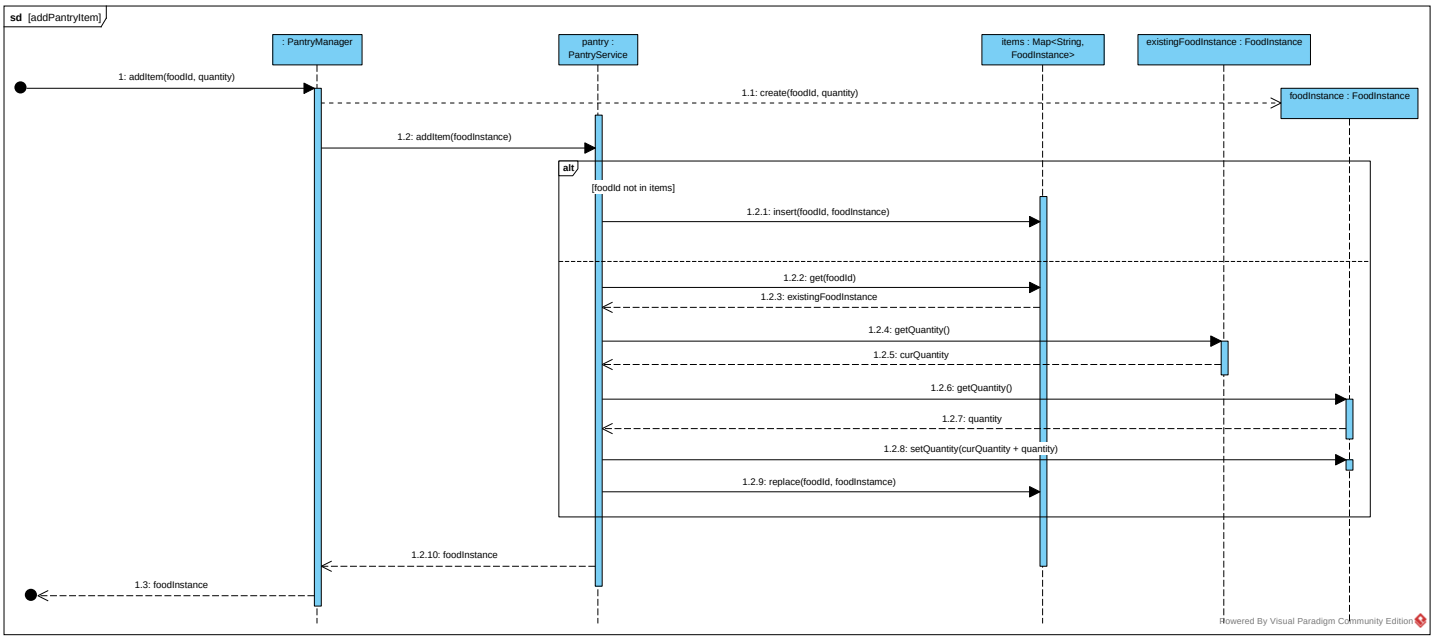

sd [addRecipe]
: RecipeController
: RecipeService
recipeList : Map<String, Recipe>
: RecipeDBAccessor
: IdService
2: addRecipe(name, ingredients, instructions, servings)
1: id = getId()
2.1: create(id, name, ingredients, instructions, servings)
recipe : Recipe
2.2: addRecipe(recipe)
2.2.1: put(recipe)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

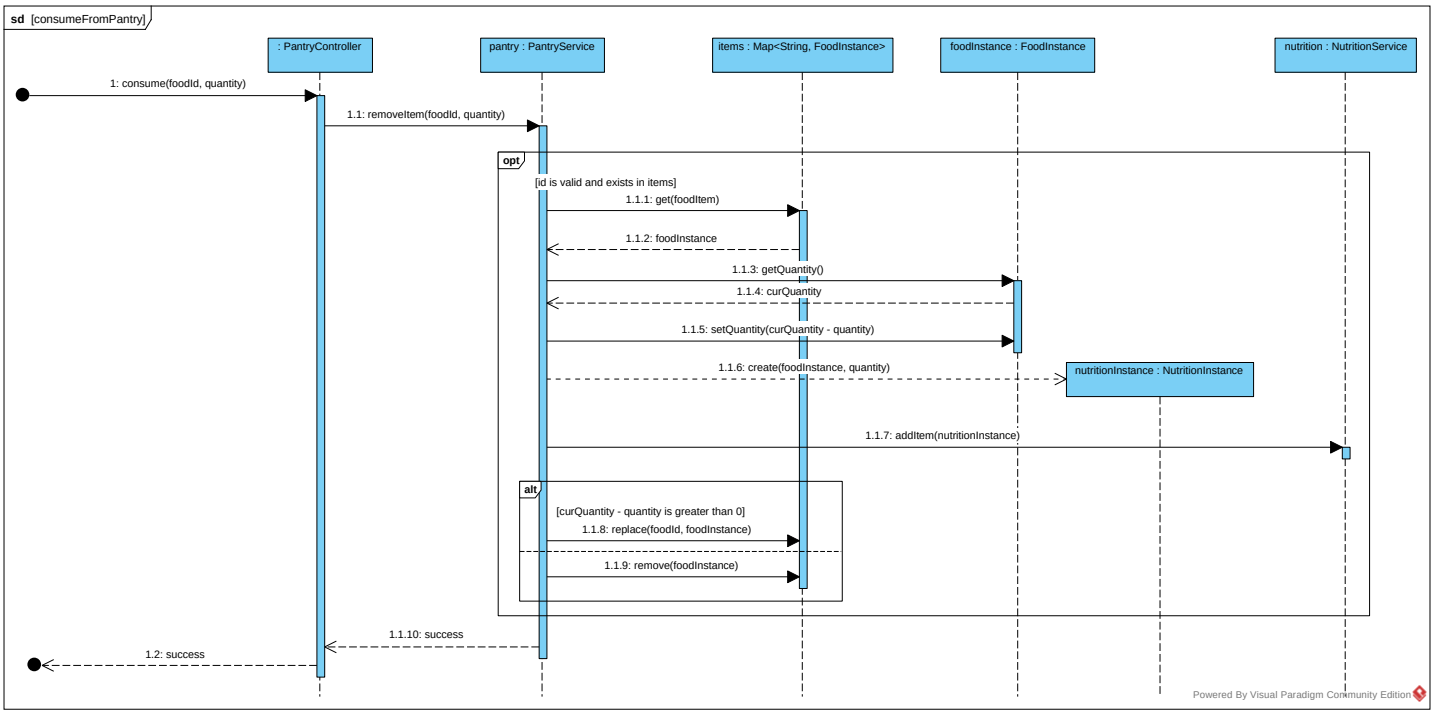

sd [searchByRecipeName]
: RecipeController
: RecipeService
recipes : Map<String, Recipe>
itemList : Map<Integer, Recipe>
recipe : Recipe
returnMap : Map<Integer, Recipe>
1: searchByRecipeName(query)
1.1: getRecipesNameStartsWith(query)
1.1.1: create
loop
[hasNext]
1.1.2: recipe = next()
1.1.3: getName()
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

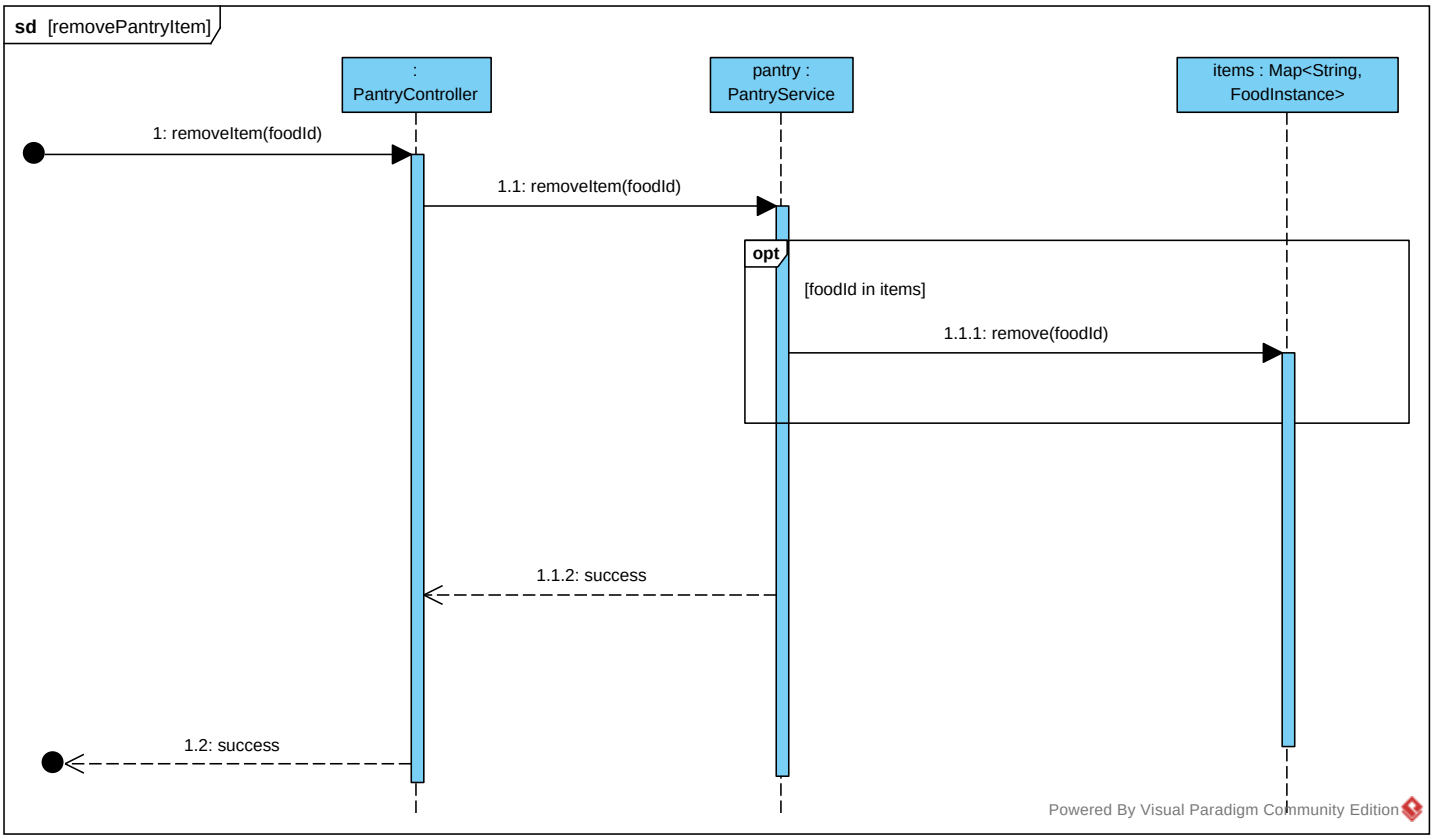

sd [addFood]
: FoodController
: FoodService
pantryDao : FoodInstanceDAO
1: addFood(name, category, nutrition)
1.1: <<create>>
food : Food
1.2: addFood(food)
1.2.1: saveNew(food)
1.2.2: success
1.3: success
Powered By Visual Paradigm Community Edition

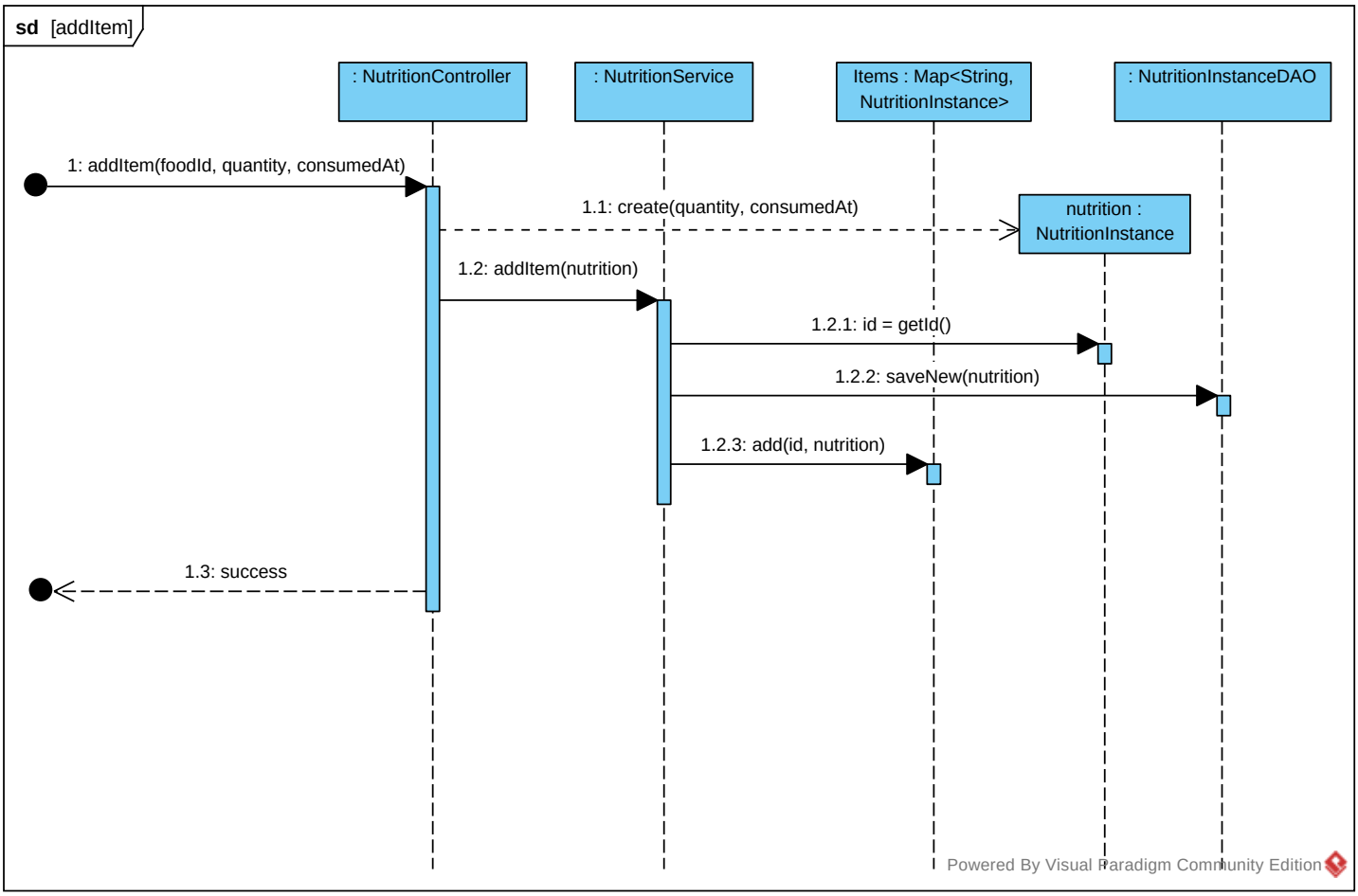

sd [addIngredientsToCart]
: RecipeController
: ShoppingService
recipeMap : Map<String, Recipe>
recipe : Recipe
ingredients : List<FoodInstance>
shoppingDao : FoodInstanceDAO
1: addIngredientsToCart(recipeID)
1.1: recipe = get(recipeId)
1.2: ingredients = getIngredients()
loop
[hasNext]
1.3: ingredient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

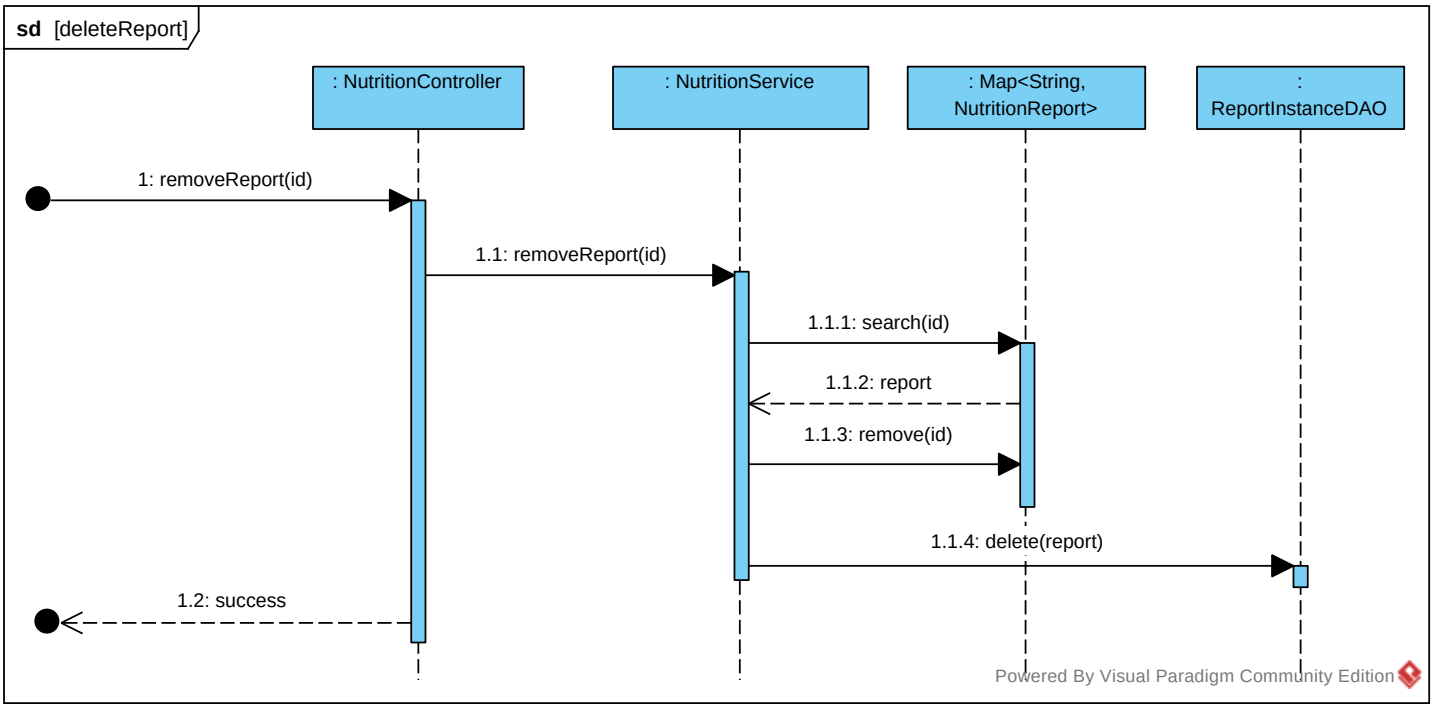

sd [purchaseItems]
: ShoppingController
foodIds : List<String>
: ShoppingService
shoppingItems : Map<String, FoodInstance>
shoppingDao : FoodInstanceDAO
: PantryService
pantryDao : FoodInstanceDAO
1: purchaseItems(foodIds)
1.1: shoppingItems = getItems()
loop
[hasNext]
1.2: foodId = next()
1.3: item = 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

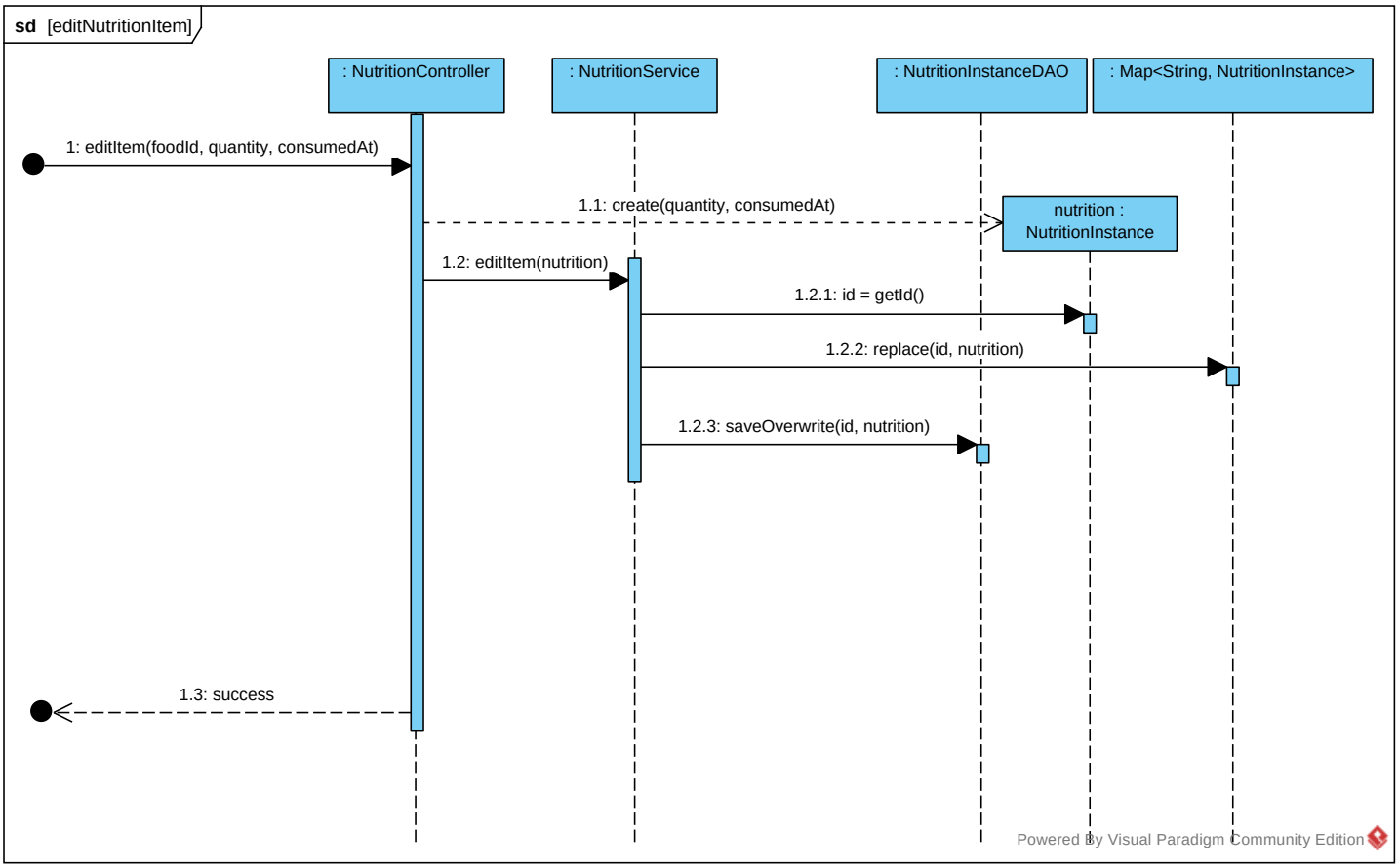

sd [addPantryItem]
: PantryManager
pantry : PantryService
items : Map<String, FoodInstance>
existingFoodInstance : FoodInstance
foodInstance : FoodInstance
1: addItem(foodId, quantity)
1.1: create(foodId, quantity)
1.2: addItem(foodInstance)
alt
[foodId not in items]
1.2.1: insert(foodId, foodInstance)
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

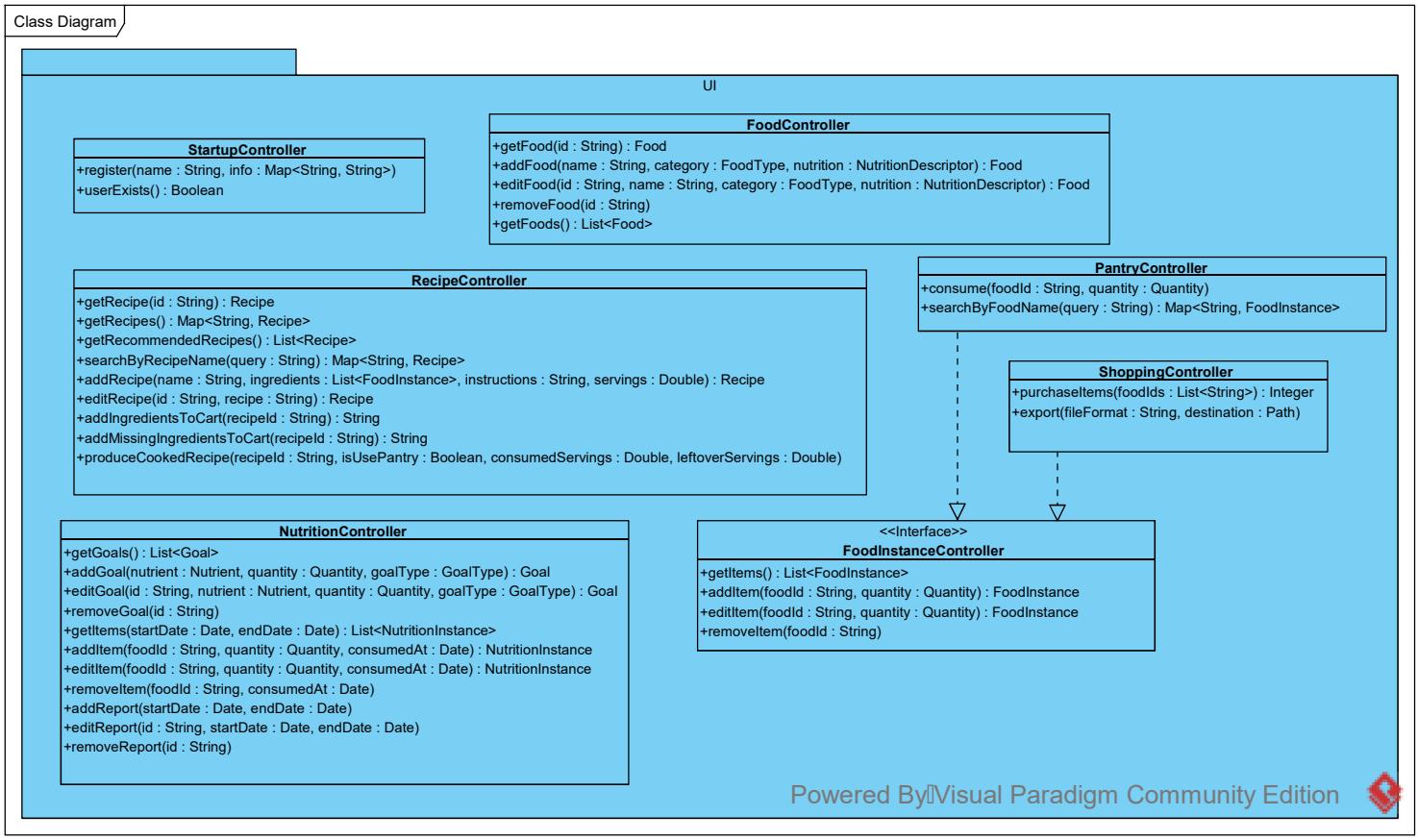

sd [consumeFromPantry]
: PantryController
pantry : PantryService
items : Map<String, FoodInstance>
foodInstance : FoodInstance
nutrition : NutritionService
1: consume(foodId, quantity)
1.1: removeItem(foodId, quantity)
opt
[id is valid and exists in items]
1.1.1: get(foodItem)
1.1.2: foodInstance
1.1.3: 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

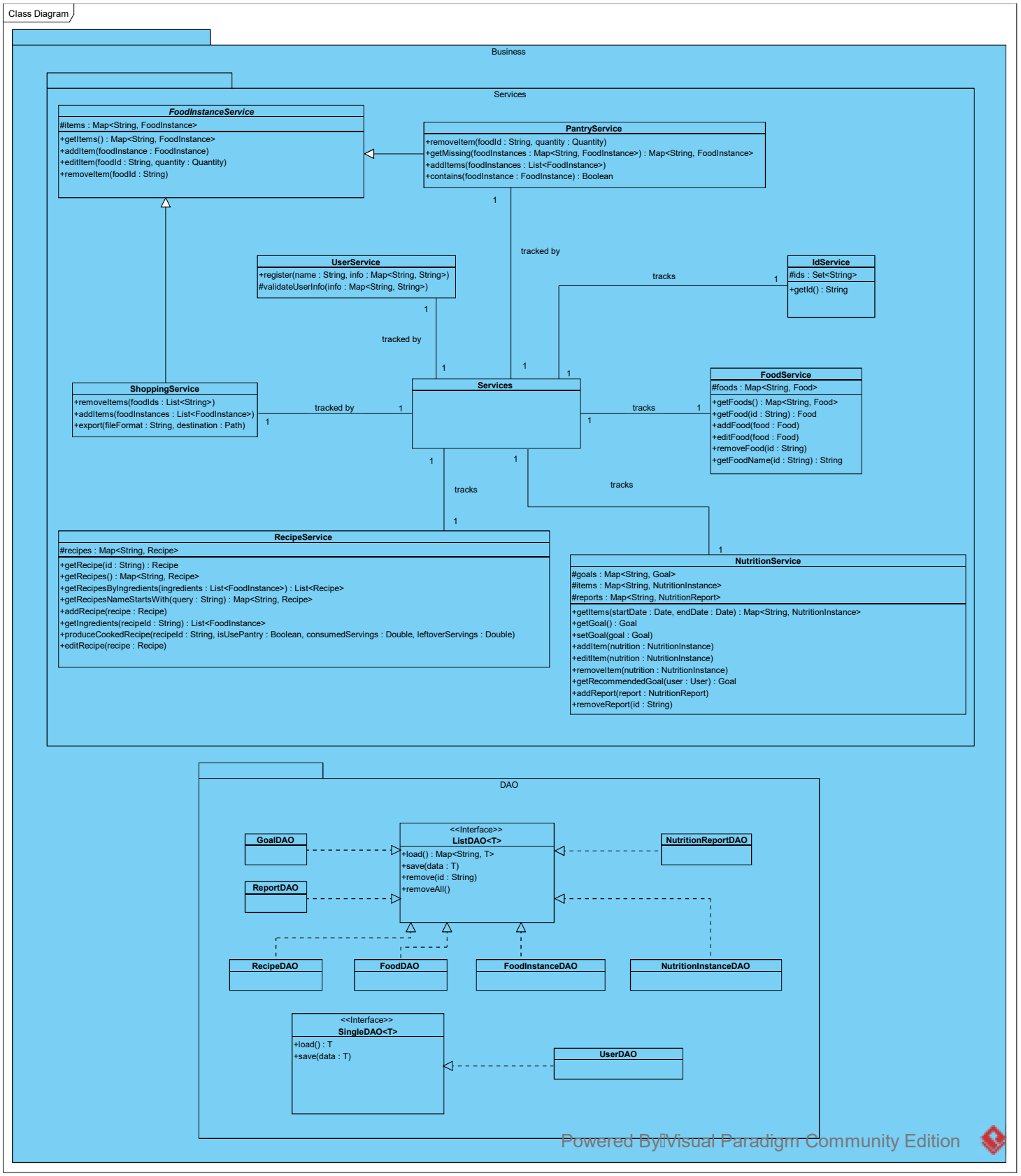

sd [removePantryItem]
: PantryController
pantry : PantryService
items : Map<String, FoodInstance>
1: removeItem(foodId)
1.1: removeItem(foodId)
opt
[foodId in items]
1.1.1: remove(foodId)
1.1.2: success
1.2: success
Powered By Visual Paradigm Community Edition

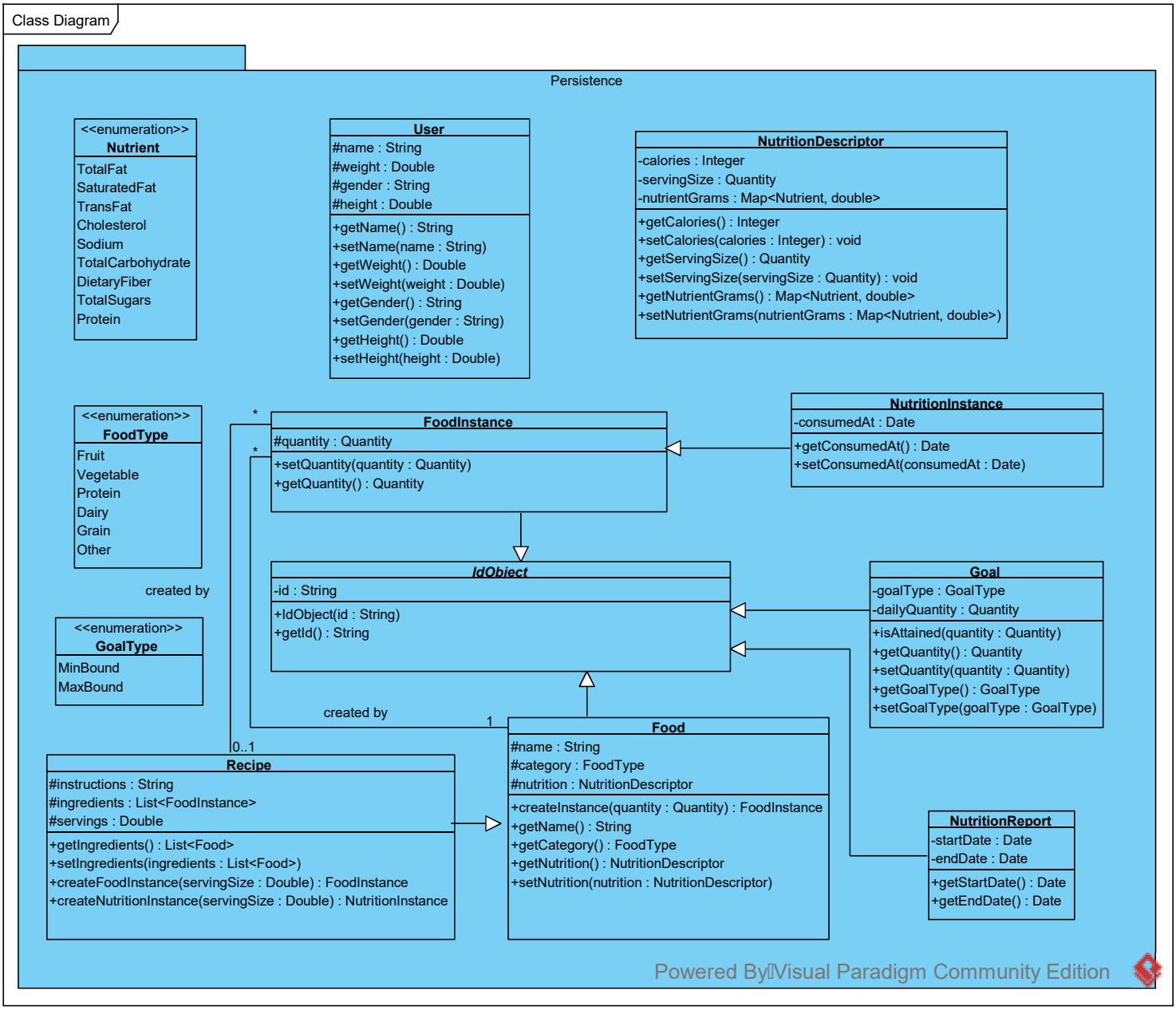

sd [addItem]
: NutritionController
: NutritionService
Items : Map<String, NutritionInstance>
: NutritionInstanceDAO
1: addItem(foodId, quantity, consumedAt)
1.1: create(quantity, consumedAt)
nutrition : NutritionInstance
1.2: addItem(nutrition)
1.2.1: id = getId()
1.2.2: saveNew(nutrition)
1.2.3: add(id,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

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 which 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 which 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 which 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 which 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

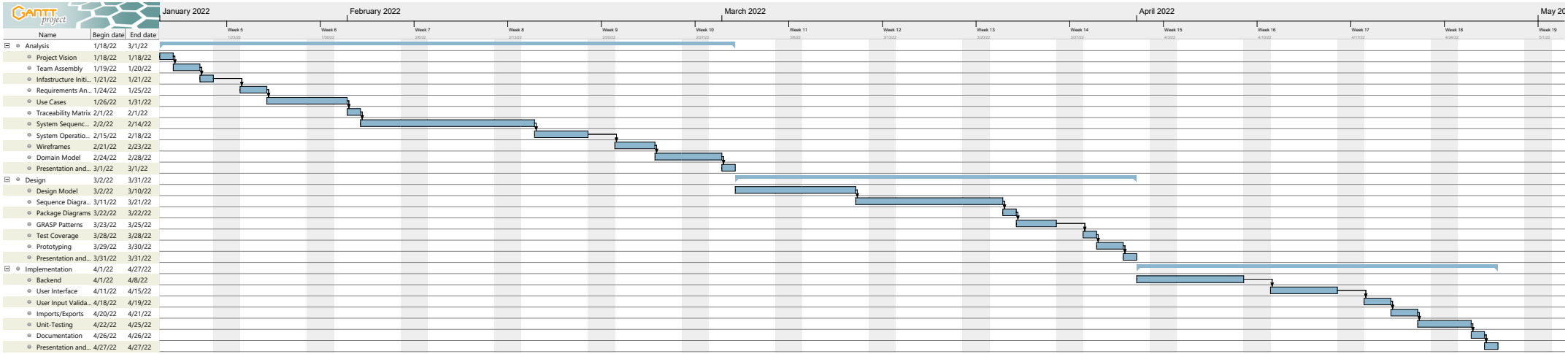
Tasks

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

Resources

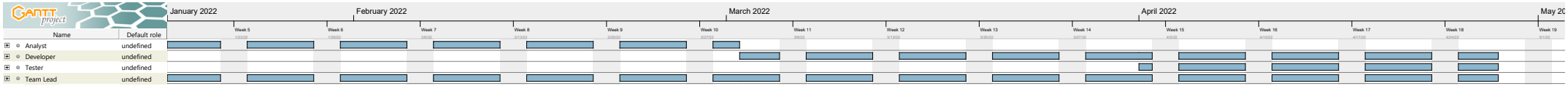
Name	Default role
Analyst	undefined
Developer	undefined
Tester	undefined
Team Lead	undefined

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

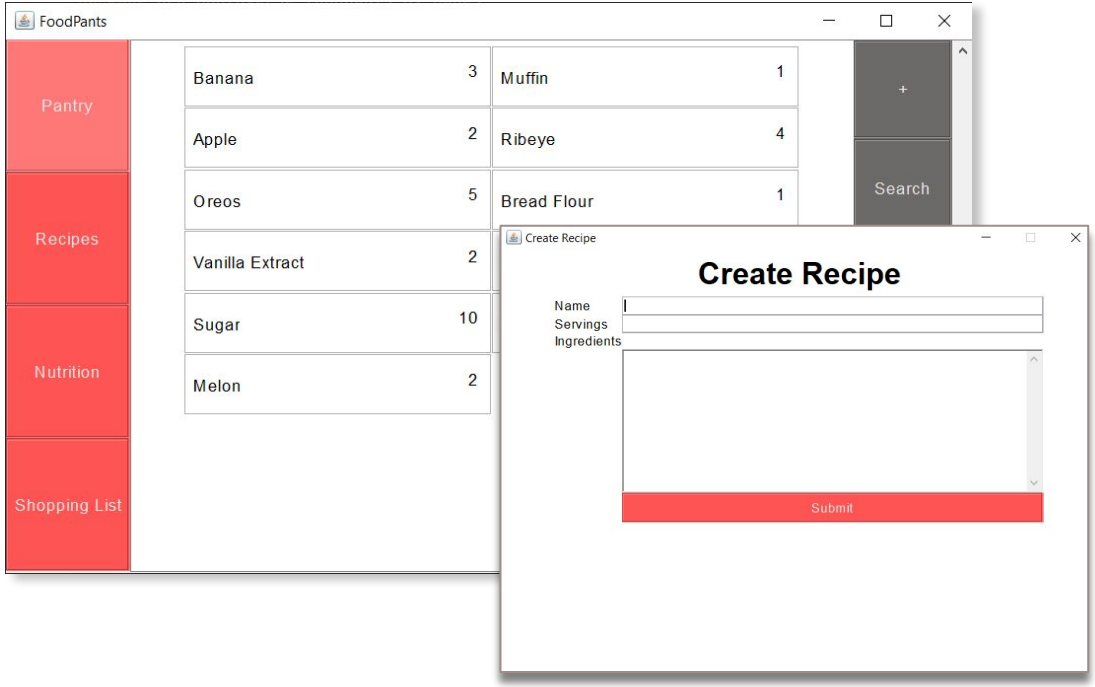
Iteration II

“

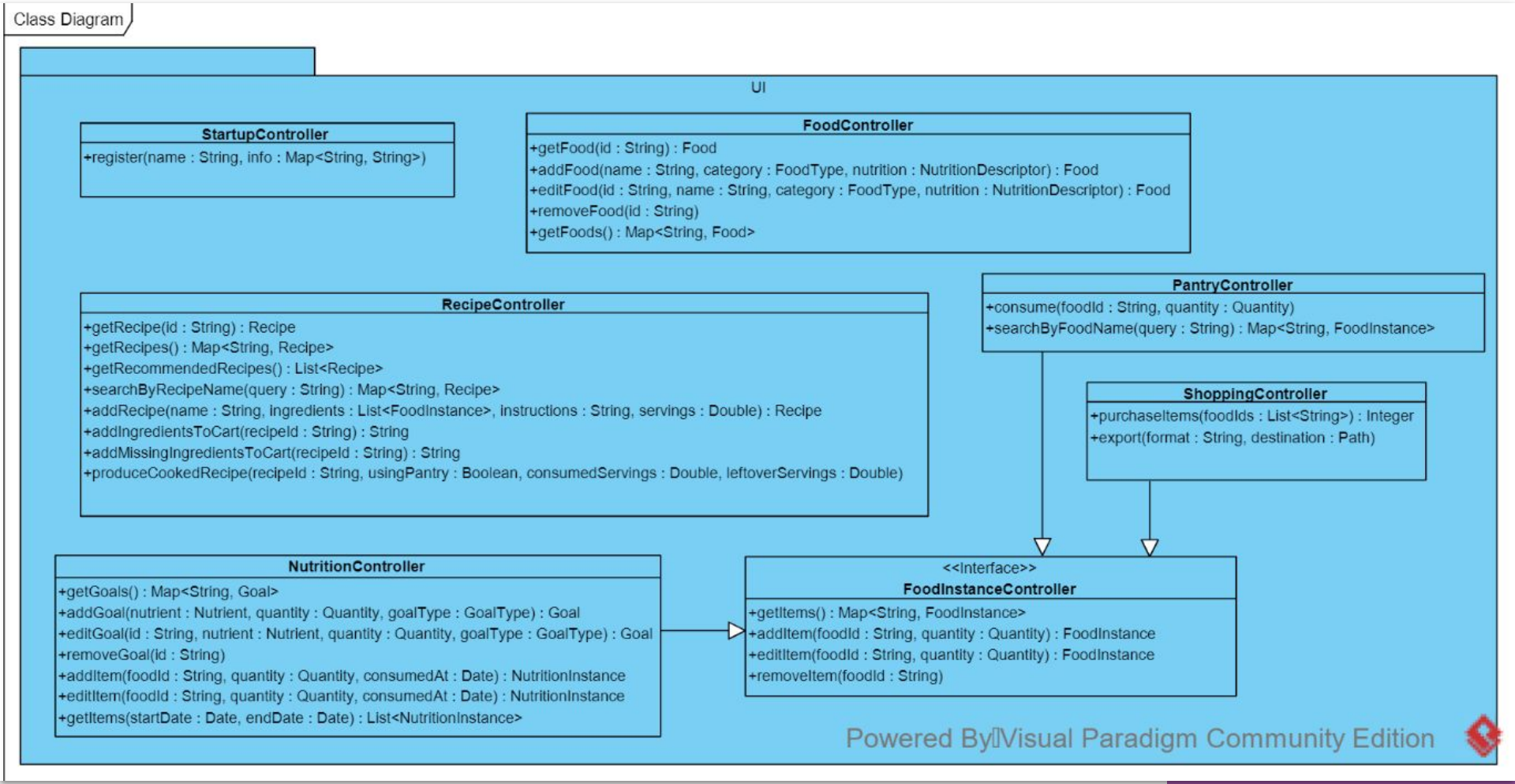
Food pants is the greatest new app coming to the market. I wish I found it sooner.”

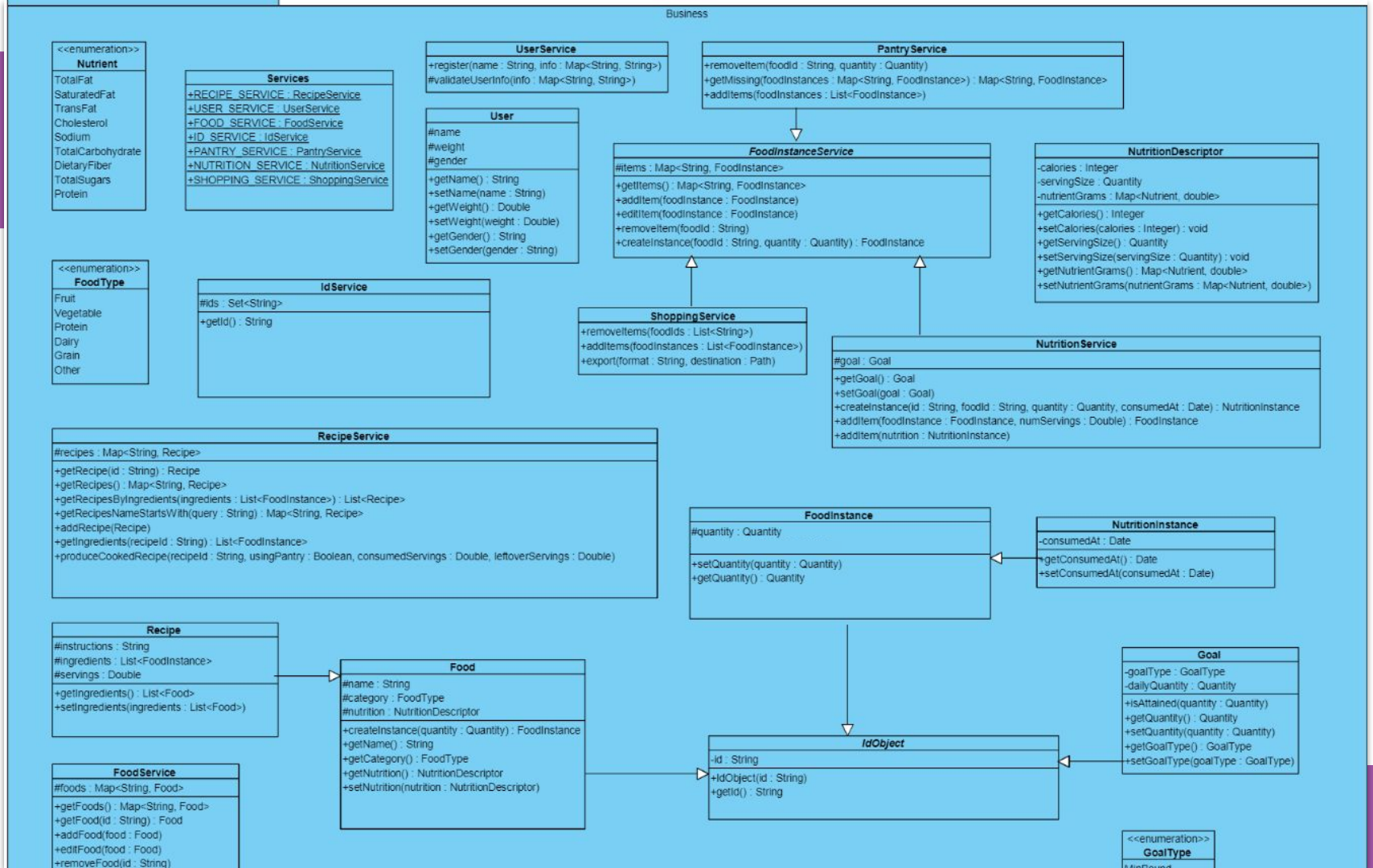
- Abe Lincolns

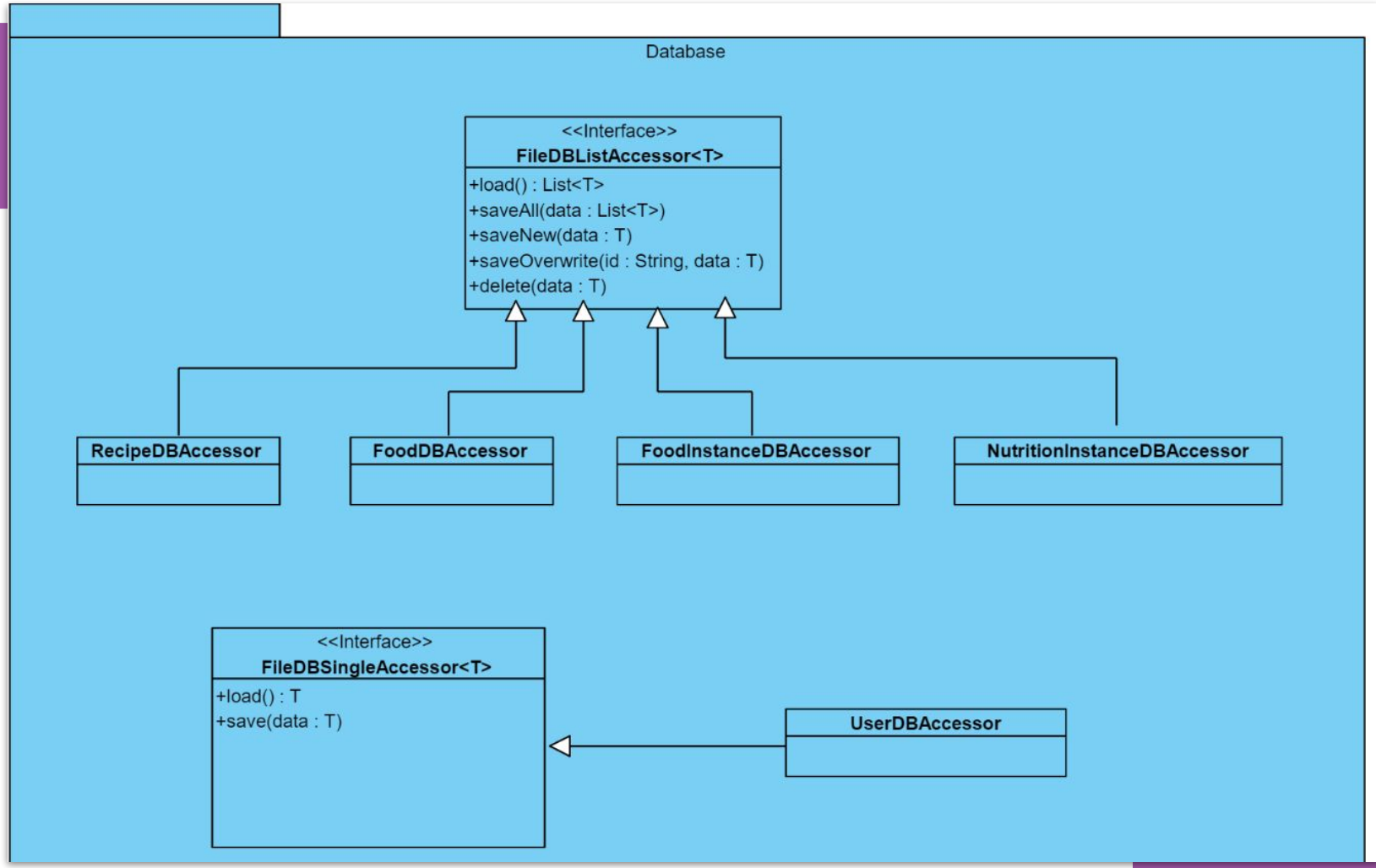
Demo User Interface



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.

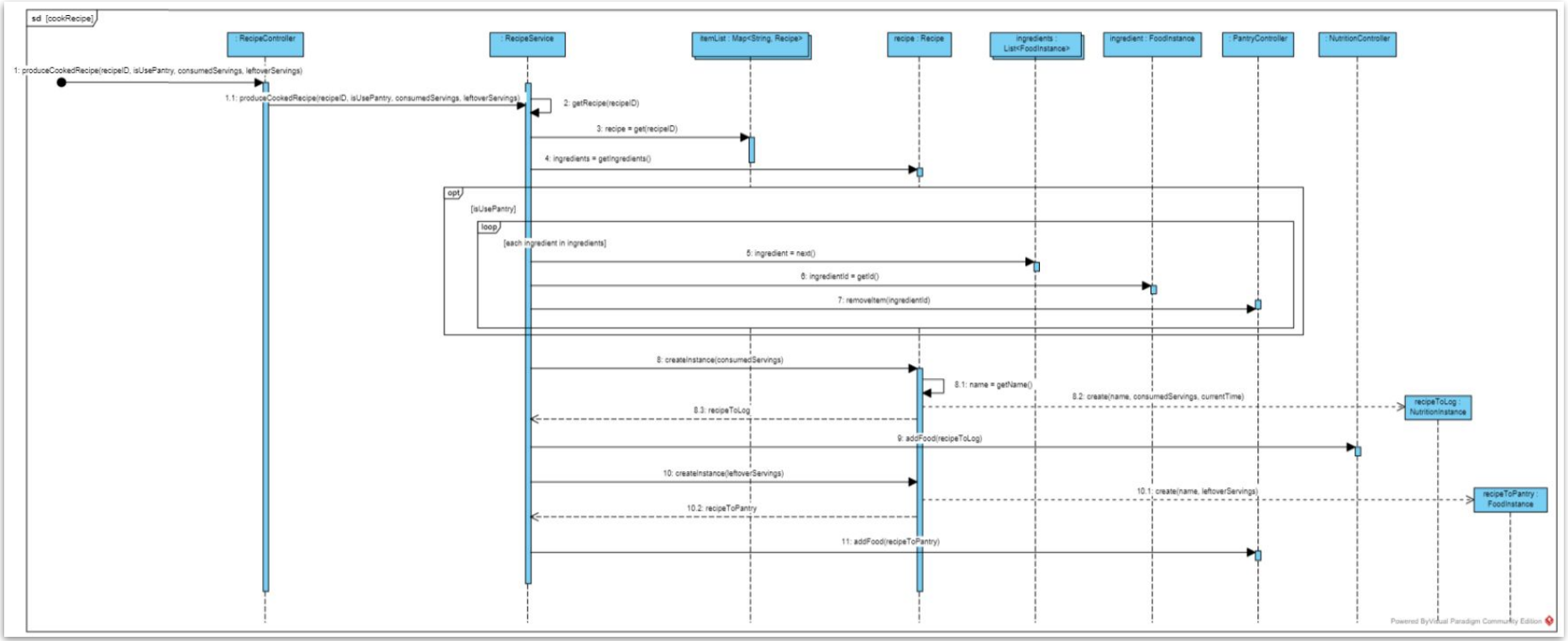
Class Name	Food
GRASP Pattern	Information Expert
Description	An information expert for food types. This class contains all necessary information to store a food type object.

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.

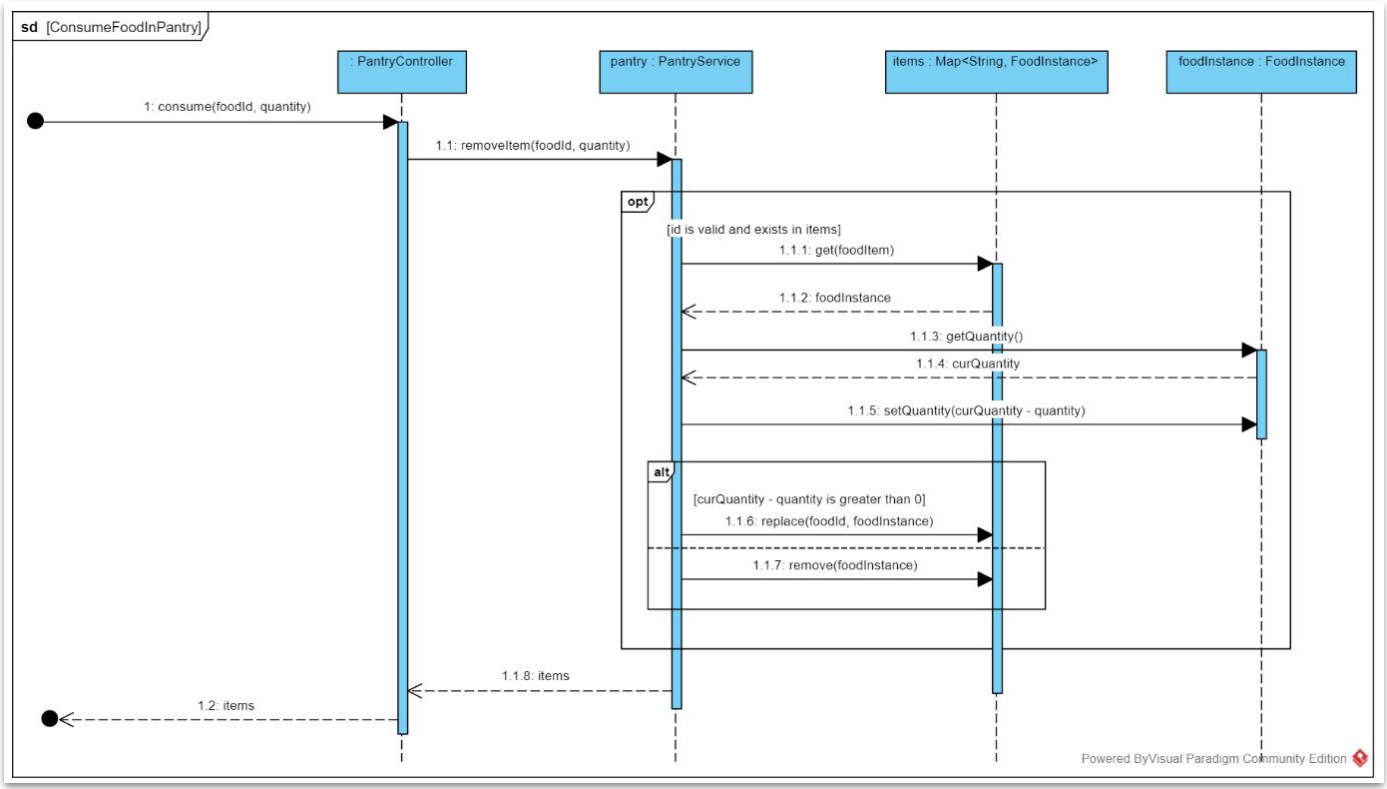
Class Name	RecipeDBAccessor
GRASP Pattern	Low Coupling
Description	This database accessor helps separate the service from the database and keeps the services separated from each other.

Example GRASP

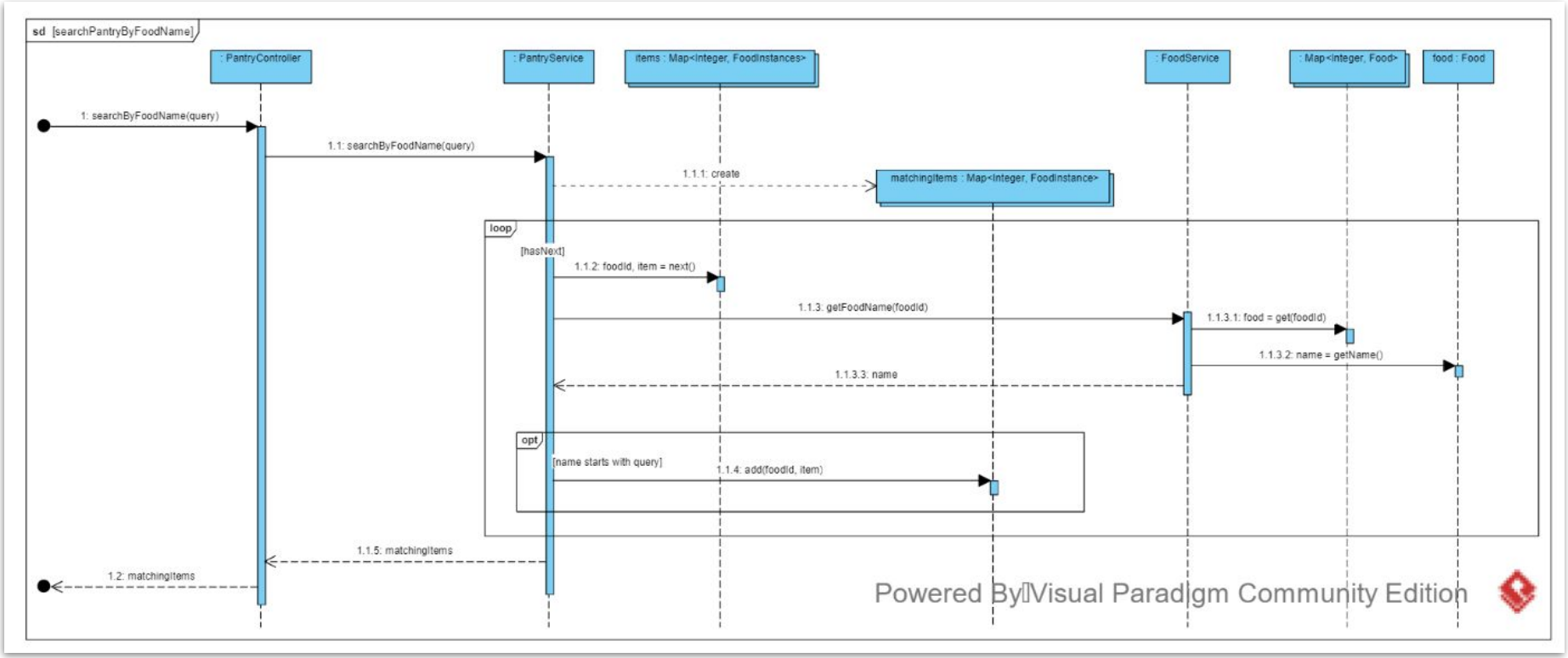
Sequence Diagrams



Example Sequence Diagram: Produce Cooked Recipe



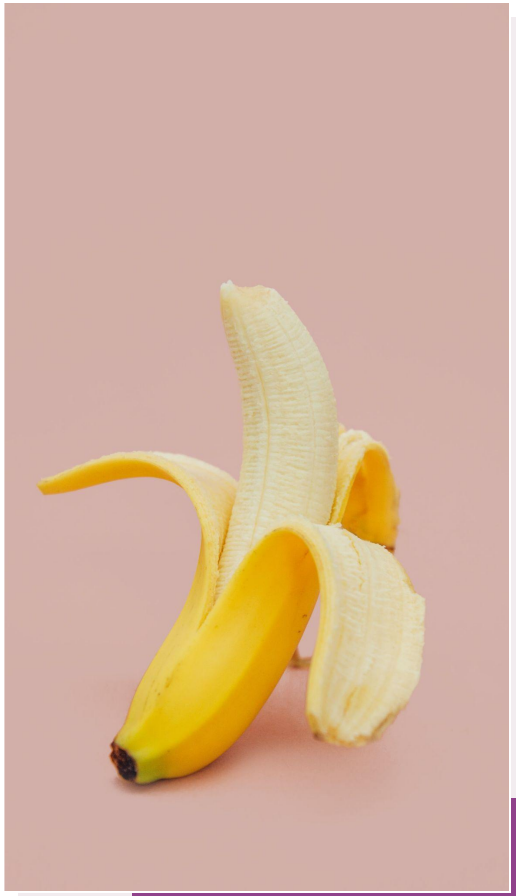
Example Sequence Diagram: Consume Food In Pantry



Example Sequence Diagram: Search Pantry

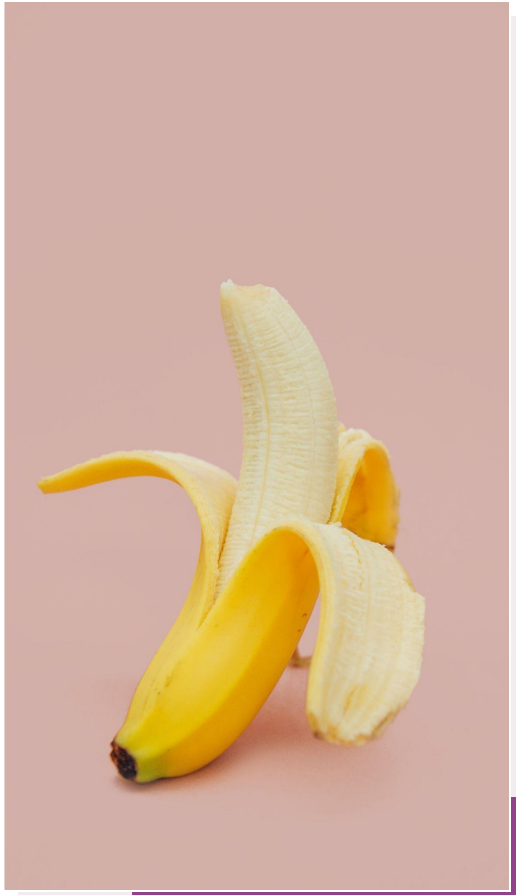
Test Coverage Plan

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.

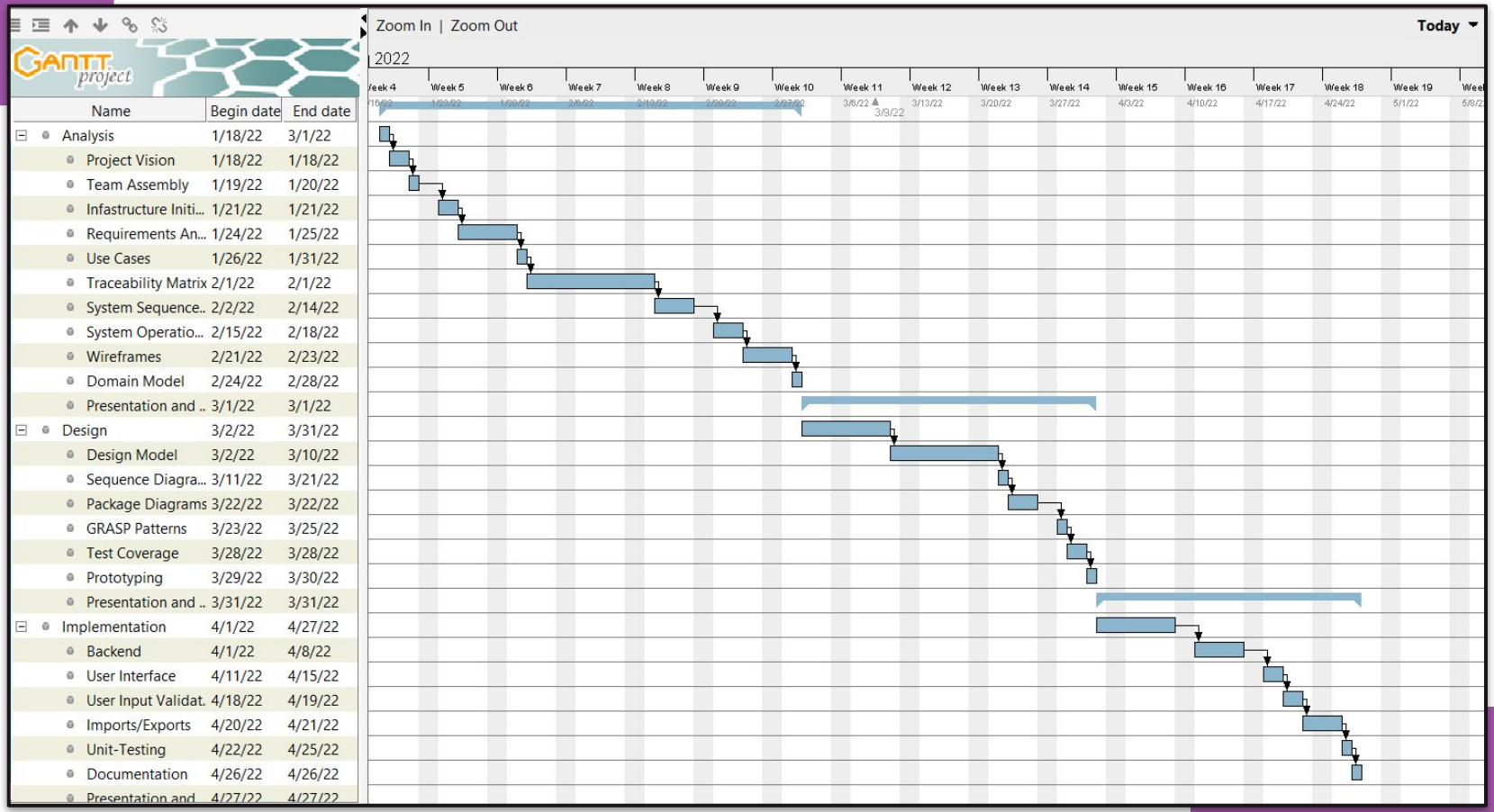


Test Coverage Plan (Cont.)

(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)



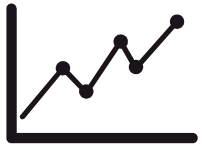
Updated Gantt Project Plan



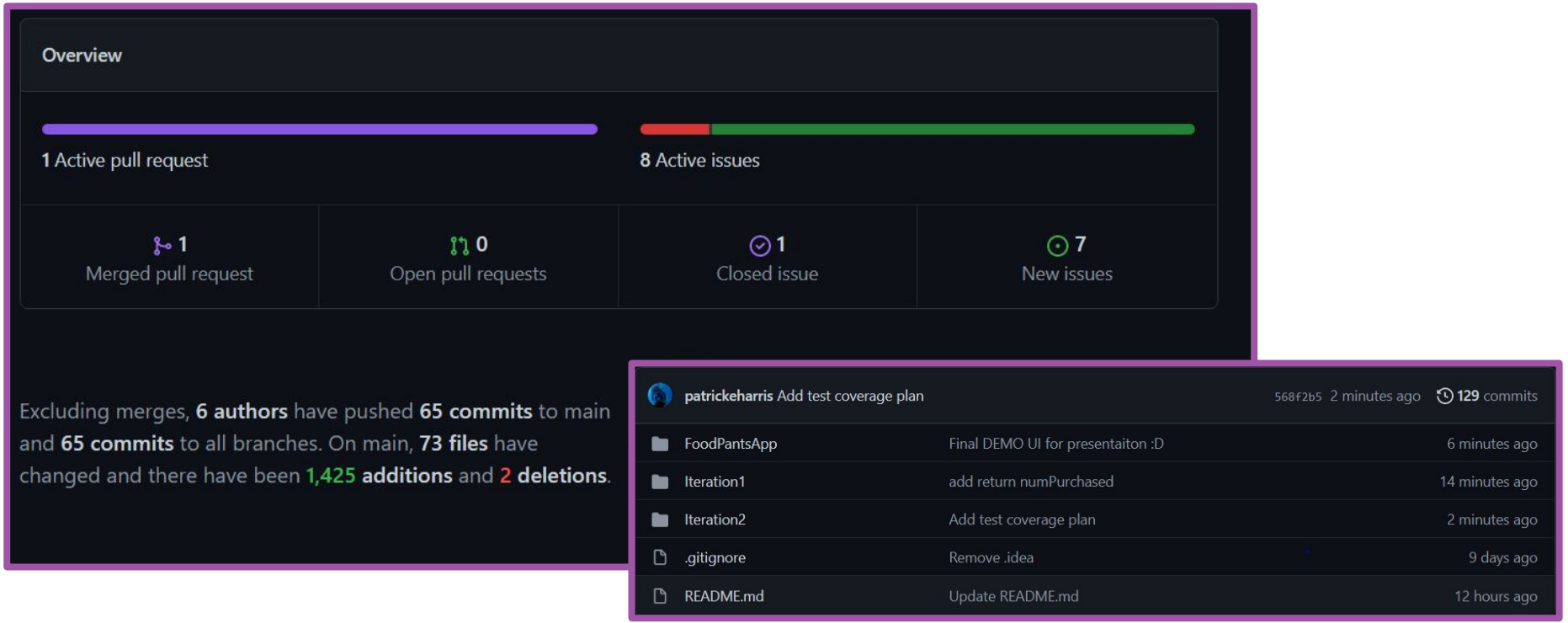
Name	Begin date	End date
Analysis	1/18/22	3/1/22
Project Vision	1/18/22	1/18/22
Team Assembly	1/19/22	1/20/22
Infastructure Initi...	1/21/22	1/21/22
Requirements An...	1/24/22	1/25/22
Use Cases	1/26/22	1/31/22
Traceability Matrix	2/1/22	2/1/22
System Sequence..	2/2/22	2/14/22
System Operatio...	2/15/22	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

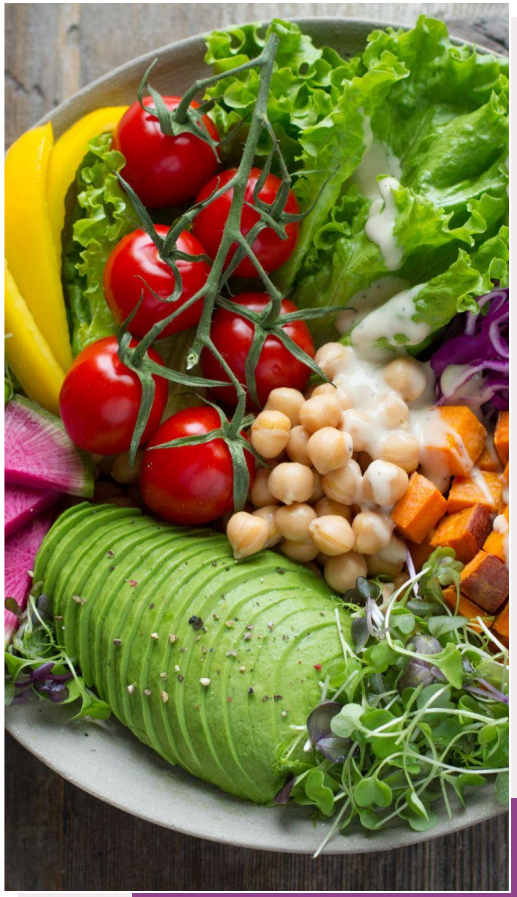
Controllers enhancement help wanted
#7 opened 8 days ago by lukalelovic

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



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

WEBSITE (with .jar of DEMO UI)
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