Assignment 2

Team number: 99
Team members

Name	Student Nr.	Email
C.T.T. Le (Thành)	**2737387**	**lecungtuantranh@student.vu.nl**
Shivangi Kachole	**2634561**	**s.c.kachole@student.vu.nl**
Kyle Schippers	**2729563**	**k.a.schippers@student.vu.nl**
Quang Tu Le	**2723610**	**q.t.le@student.vu.nl**

Do NOT describe the obvious in the report (e.g., we know what a `name` or `id` attribute means), focus more on the **key design decisions** and their "**why**", the pros and cons of possible **alternative designs**, etc.

Format: establish formatting conventions when describing your models in this document. For example, you style the name of each class in bold, whereas the attributes, operations, and associations as underlined text, objects are in italic, etc.

Diagrams.io link:
https://app.diagrams.net/#G1clDFlFqJk1aL9_QEQBhGq1hW0qoX5f2u

Summary of changes from Assignment 1

Author(s): Tu, Thanh, Shivangi, Kyle

Changes to Overview:
Added missing feature:
+ Add custom notes to any recipe

Removed features:
- Save your meals on your calendar
- Find recipes based on the time you have
- Vegan/Vegetarian meals

Added missing functional feature:
+ Annotation: The user can add/edit notes for each instruction of a recipe.

Changes to descriptions of functional features:
+ (CRUD) Create, Read, Update, Delete recipes. This is the core of the system.
+ Pantry & Shopping list: Mutable lists of Ingredients. This will help the users keep track of ingredients owned, and add any missing ingredients to the shopping list.

+ Lookup & Filters: Recipes can have tags (i.e. vegan, beef, halal, etc.) and can be filtered based on the tags. This feature will make it easier for the users to look up for a recipe they really want.
+ Execution: This feature will let the user walk through the recipe step-by-step and check whether the ingredients are available (handled by the pantry feature). This feature will allow users to follow the recipe.

Moved quality features to functional features:

+ Sorting: Sort recipes by name or time in minutes. This will make it easier for the users to follow the recipe list that is provided to them by the system.
+ Up-to-date: The function getRecipe() is regularly called to update recipes based on recent data collected from the user.

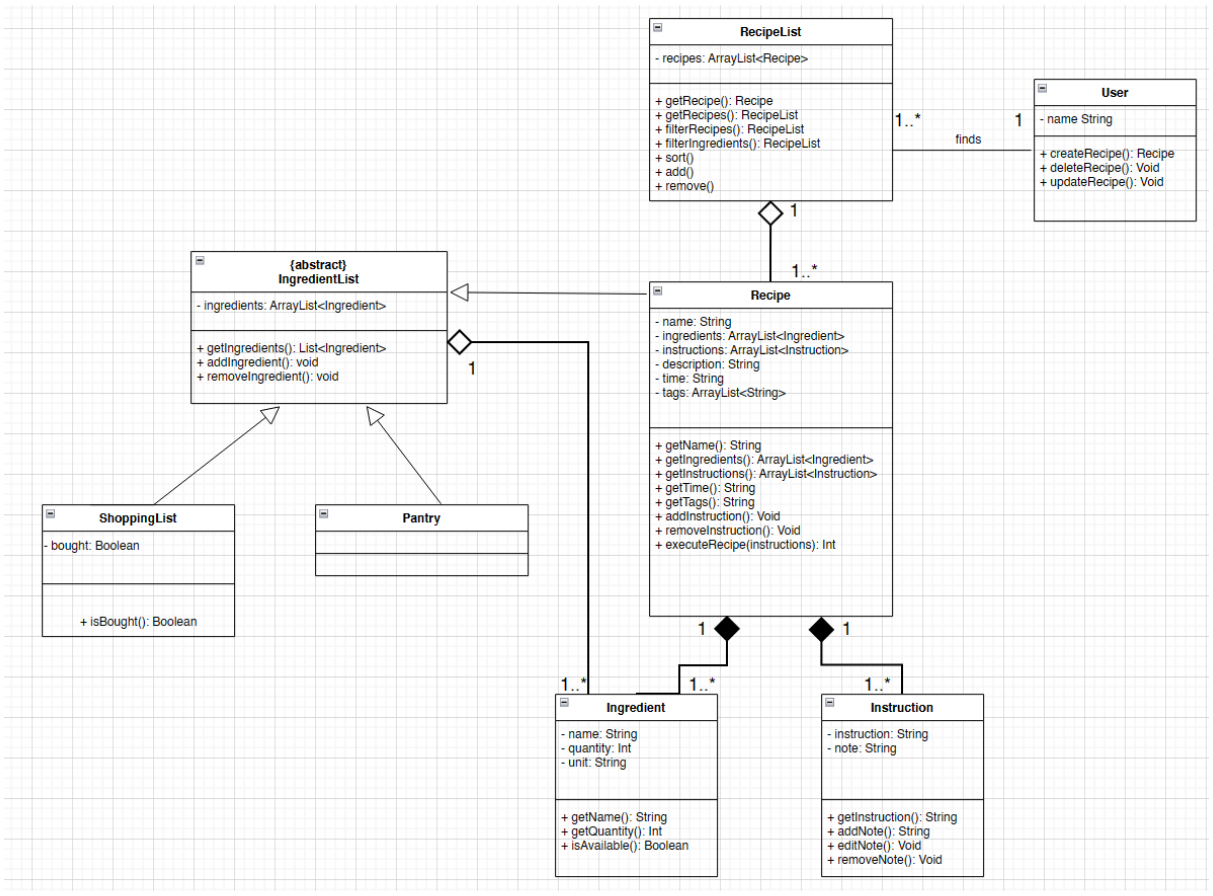
Added quality feature:

+ Fast Recipe Opening - Performance efficiency - Get the RecipeList and open a Recipe in under 3 seconds

Class diagram

Author(s): Tu, Thanh, Shivangi, Kyle

This chapter contains the specification of the UML class diagram of your system, together with a textual description of all its elements.



- User class:
    + Is used to represent the user, has a String for name.

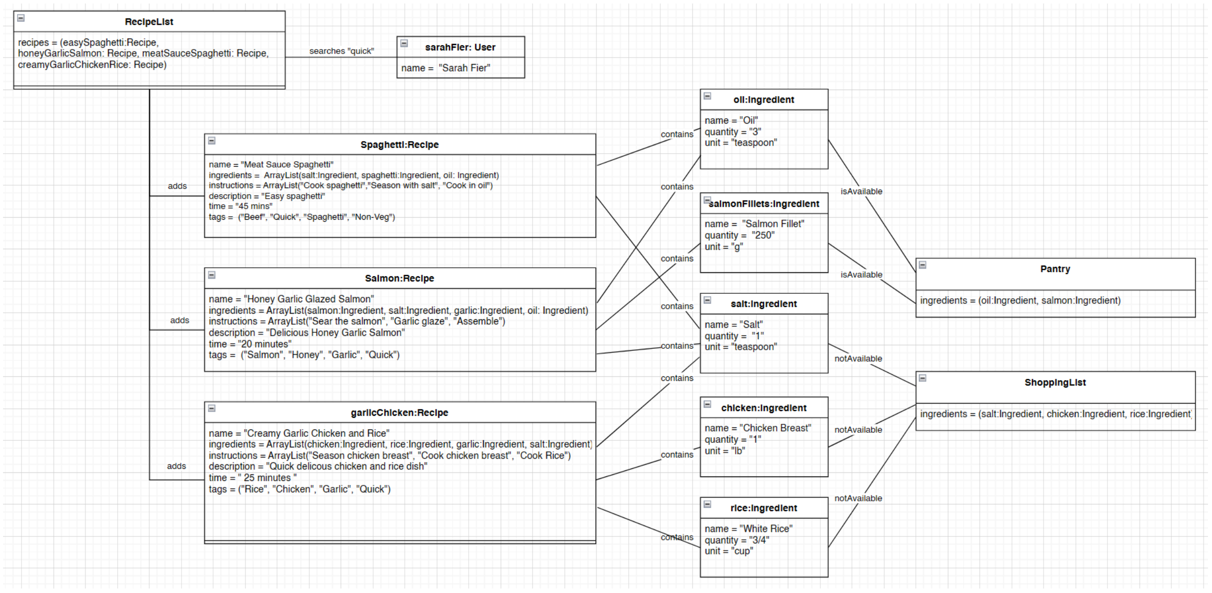
    + Users can create, delete, update a recipe, hence the functions createRecipe(), deleteRecipe(), updateRecipe().
    + The User class shares a binary association with the RecipeList class, where the user finds a recipe from the recipe list.
- Recipe class:
    + Includes the following attributes: name, ArrayList of Ingredients, ArrayList of Instructions, description, time taken as a string, ArrayList for tags (optional).
    + Operations with the prefix “get” or “remove” return or remove the recipe’s attribute according to the operation. getTime() returns the time taken to make the recipe, getTags() returns the tags for the recipe itself (not mandatory to have tags).
    + Users can also add and remove instructions from the recipe.
    + executeRecipe executes the Recipe step-by-step and returns 0 on success.
    + There must be at least one (>=1) ingredient and one instruction for each recipe.
- RecipeList class:
    + Has an ArrayList to represent all the recipes in the recipe list.
    + getRecipes() returns the recipe list in the app’s storage and getRecipe() returns a single recipe for the user to view it.
    + filterIngredients() and filterRecipes() returns smaller lists based on the user’s search.
    + The recipes in the list can also be sorted, removed, or added by the user, hence sort(), add(), remove()
    + There’s a shared aggregation between recipeList and recipe class, as the recipe class is part of the recipeList class. A recipe list can have multiple recipes and an user can have multiple recipe lists.
- Ingredient class:
    + Has a string for name, quantity as an integer, and for units we have a string for it.
    + getName() returns the Ingredient name, getQuantity() returns an int for amount of the Ingredient itself, and isAvailable() is an operation to check if the user have enough ingredient in the pantry (later for the abstract IngredientList class).
    + Ingredient class has a shared aggregation with the abstract class IngredientList, and an IngredientList can have multiple Ingredients.
- Instruction class:
    + Also has a string for the name, and a string for its notes.
    + User can edit the instruction’s note, we have operations like addNote(), editNote(), and removeNote() for that.
    + Instruction class shares a composition with the Recipe. A recipe MUST has at least one instruction.
- Abstract IngredientList class:
    + Shares a generalization relationship with subclass Pantry and Shopping List.
    + This class is used for the Ingredients in the user pantry, or the user shopping list. When a recipe is executed, the system checks the ingredient list in the user pantry, to see if they have enough ingredients to cook. If they don’t, missing ingredients will be added to the shopping list.

+ It has an ArrayList of ingredients, and getIngredients() returns a list of ingredients. The user can also add and remove certain ingredients.
+ The ShoppingList class has a private boolean to mark which ingredient is bought, and a boolean operation to check if it has been bought.

Object diagram

Author(s): Shivangi, Kyle, Thanh, Tu

This chapter contains the description of a "snapshot" of the status of your system during its execution. This chapter is composed of a UML object diagram of your system, together with a textual description of its key elements.

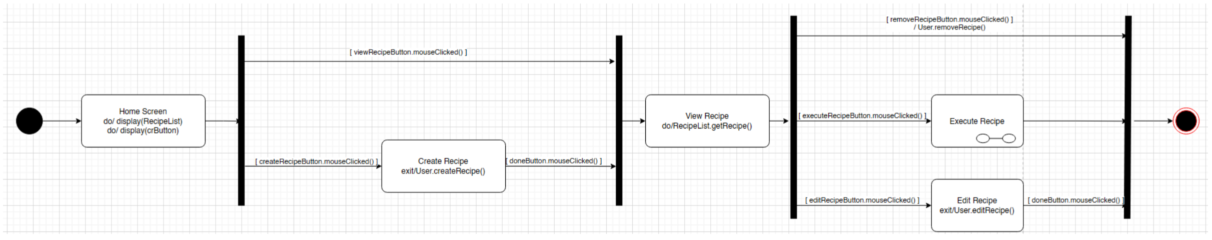

RecipeList
recipes = (easySpaghetti:Recipe, honeyGarlicSalmon: Recipe, meatSauceSpaghetti: Recipe, creamyGarlicChickenRice: Recipe)
searches "quick"
sarahFier: User
name = "Sarah Fier"
adds
Spaghetti:Recipe
name = "Meat Sauce Spaghetti"
ingredients = ArrayList(salt:Ingredient, spaghetti:Ingredient, oil: Ingredient)
instructions = ArrayList("Cook spaghetti","Season with salt", "Cook in oil")
description = "Easy spaghetti"
time = "45 mins"
tags = ("Beef", "Quick", "Spaghetti", "Non-Veg")
Salmon:Recipe
name = "Honey Garlic Glazed Salmon"
ingredients = ArrayList(salmon:Ingredient, salt:Ingredient, garlic:Ingredient, oil: Ingredient)
instructions = ArrayList("Sear the salmon", "Garlic glaze", "Assemble")
description = "Delicious Honey Garlic Salmon"
time = "20 minutes"
tags = ("Salmon", "Honey", "Garlic", "Quick")
garlicChicken:Recipe
name = "Creamy Garlic Chicken and Rice"
ingredients = ArrayList(chicken:Ingredient, rice:Ingredient, garlic:Ingredient, salt:Ingredient)
instructions = ArrayList("Season chicken breast", "Cook chicken breast", "Cook Rice")
description = "Quick delicous chicken and rice dish"
time = " 25 minutes "
tags = ("Rice", "Chicken", "Garlic", "Quick")
contains
oil:Ingredient
name = "Oil"
quantity = "3"
unit = "teaspoon"
salmonFillets:Ingredient
name = "Salmon Fillet"
quantity = "250"
unit = "g"
salt:Ingredient
name = "Salt"
quantity = "1"
unit = "teaspoon"
chicken:Ingredient
name = "Chicken Breast"
quantity = "1"
unit = "lb"
rice:Ingredient
name = "White Rice"
quantity = "3/4"
unit = "cup"
isAvailable
notAvailable
Pantry
ingredients = (oil:Ingredient, salmon:Ingredient)
ShoppingList
ingredients = (salt:Ingredient, chicken:Ingredient, rice:Ingredient

The diagram shows the system containing:

- The User searches for the tag "Quick" in RecipeList.The RecipeList contains 3 different Recipes with the matching "Quick" tag.
- Each recipe contains several Ingredients, recipes can share Ingredients.
- Ingredients that the User owns will be isAvailable: in the Pantry.
- Ingredients that the User does not own will be notAvailable: in the ShoppingList.

State machine diagrams

Author(s): Thanh, Shivangi, Kyle, Tu


Home Screen
do/ display(RecipeList)
do/ display(crButton)
[viewRecipeButton.mouseClicked()]
[createRecipeButton.mouseClicked()]
Create Recipe
exit/User.createRecipe()
[doneButton.mouseClicked()]
View Recipe
do/RecipeList.getRecipe()
[removeRecipeButton.mouseClicked()]
/ User.removeRecipe()
[executeRecipeButton.mouseClicked()]
Execute Recipe
[editRecipeButton.mouseClicked()]
Edit Recipe
exit/User.editRecipe()
[doneButton.mouseClicked()]

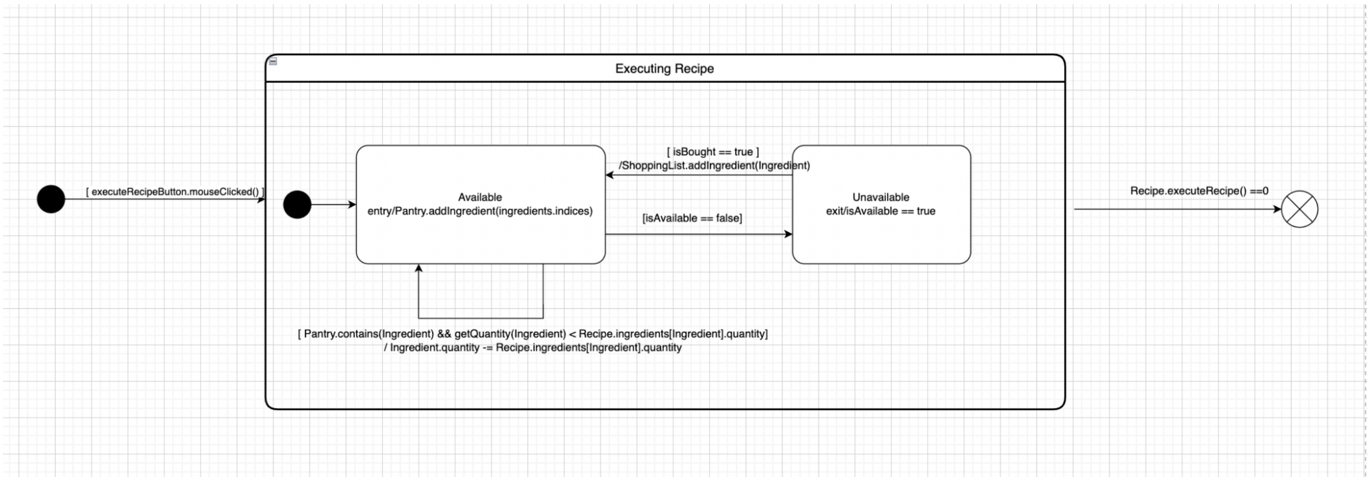
Represented classes: RecipeList

The initial state is the Home Screen, containing the RecipeList and the crButton (Create Recipe Button). The user can choose to do one of the following things:

1. Create a recipe by clicking on crButton. The system transitions to the next state: Create Recipe Screen. When exiting this state, the system adds a new Recipe to the RecipeList by doing User.createRecipe(). After the User clicks on doneButton, the machine transitions to the View Recipe state: viewing the newly created recipe and the already existing recipes.
2. View a certain recipe by clicking it directly from the home screen. The system transitions to the View Recipe state: viewing the chosen recipe. When entering this state, the system will get the chosen recipe by calling the RecipeList.getRecipe() function.

After this, the user has 3 options:

1. Remove the recipe by clicking on removeRecipeButton. The system will do User.removeRecipe() then transitions back to the Home screen.
2. Execute the Recipe by clicking on executeRecipeButton, the system will transition to the Execute Recipe state: step by step execution of the recipe. The system will go back to the Home screen after execution is finished. Here we are referencing one of the other state machine diagrams we implemented (Execute Recipe)
3. Edit the Recipe by clicking on editRecipeButton, this will bring the system to the Edit Recipe state. The system will apply the changes made by the user when exiting this state by doing User.editRecipe(). The system will go back to the Home screen when the User clicks on doneButton to finish editing.


Executing Recipe
[executeRecipeButton.mouseClicked()]
Available
entry/Pantry.addIngredient(ingredients.indices)
[isBought == true]
/ShoppingList.addIngredient(Ingredient)
[isAvailable == false]
Unavailable
exit/isAvailable == true
Recipe.executeRecipe() ==0
[Pantry.contains(Ingredient) && getQuantity(Ingredient) < Recipe.ingredients[Ingredient].quantity]
/ Ingredient.quantity -= Recipe.ingredients[Ingredient].quantity

Represented classes: Ingredient

After the initial state is triggered, when the executeRecipe button is clicked and we enter the main state which is Executing Recipe.

1. This state machine has two internal states, Available and Unavailable. These two internal states represent the availability of the ingredients from the Ingredient class. As soon as we enter the Available state we perform the internal activity where we add ingredients to the pantry with the method call Pantry.addIngredient(Ingredients.indices).
2. We also keep having external transitions to the Available state itself, where our guard keeps checking if the pantry contains the ingredients and the the quantity of the ingredients present in the pantry is less than the required quantity (done by the

method calls Pantry.contains(Ingredient) && getQuantity(Ingredient) < Recipe.ingredients[Ingredient].quantity) . The activity we perform is present while entering the state itself is we decrement the quantity available in the pantry from the required quantity from the recipe, in order to remain in the available state (done the following way Ingredient.quantity -= Recipe.ingredients[Ingredient].quantity).

In order to get to the next state, which is the Unavailable state the following happens:

1. the guard isAvailable is checked. If it is false, then we enter the unavailable state. If we want to exit the Unavailable state, then the exit activity is executed where we check if the isAvailable is true.
2. However, to enter the Available state again, the guard isBought is checked. If it is true, then the activity ShoppingList.addIngredient(Ingredient) is added to the list and we enter the Available state again. Then once all the ingredients are added, the terminate state is triggered when Recipe.executeRecipe() == 0.

For each state machine you will provide:

- the name of the class for which you are representing the internal behavior;
- a figure representing the state machine;
- a textual description of all its states, transitions, activities, etc. in a narrative manner (you do not need to structure your description into tables). We expect 3-4 lines of text for describing trivial or very simple state machines (e.g., those with one to three states), whereas you will provide longer descriptions (e.g., ~500 words) when describing more complex state machines.

The goal of your state machine diagrams is both descriptive and prescriptive, so put the needed level of detail here, finding the right trade-off between understandability of the models and their precision.

Maximum number of pages for this section: 4

Sequence diagrams

Author(s): Kyle, Shivangi, Thanh, Tu

This chapter contains the specification of at least 2 UML sequence diagrams of your system, together with a textual description of their elements. Here you have to focus on specific situations you want to describe. For example, you can describe the interaction of the player when performing a key part of the videogame, during a typical execution scenario, in a special case that may happen (e.g., an error situation), when finalizing a match, etc.

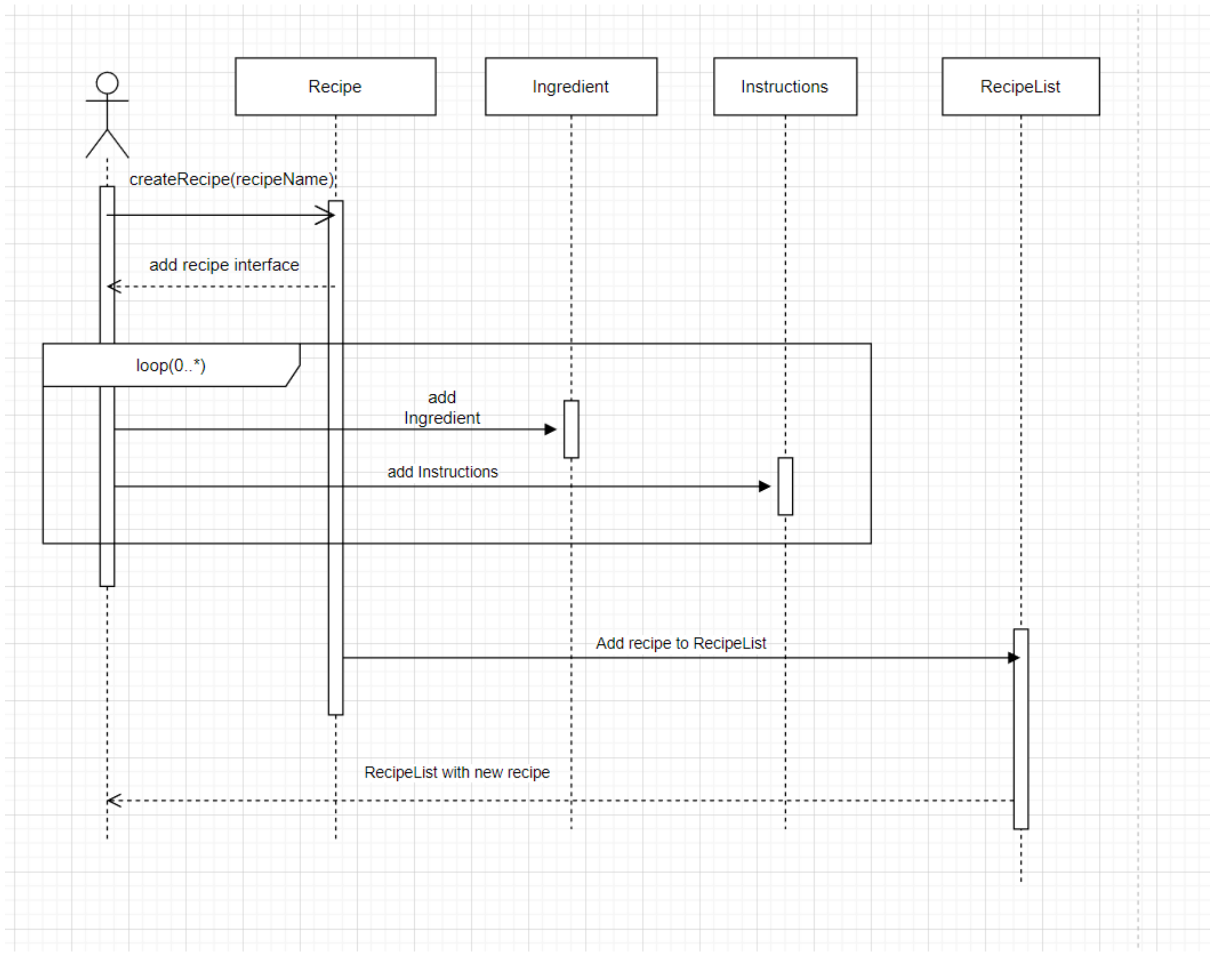
For each sequence diagram you will provide:

- a title representing the specific situation you want to describe;
- a figure representing the sequence diagram;
- a textual description of its main elements in a narrative manner (you do not need to structure your description into tables). We expect a detailed description of all the interaction partners, their exchanged messages, and the fragments of interaction where they are involved. For each sequence diagram we expect a description of about 200-500 words.

- The goal of your sequence diagrams is both descriptive and prescriptive, so put the needed level of detail here, finding the right trade-off between understandability of the models and their precision.

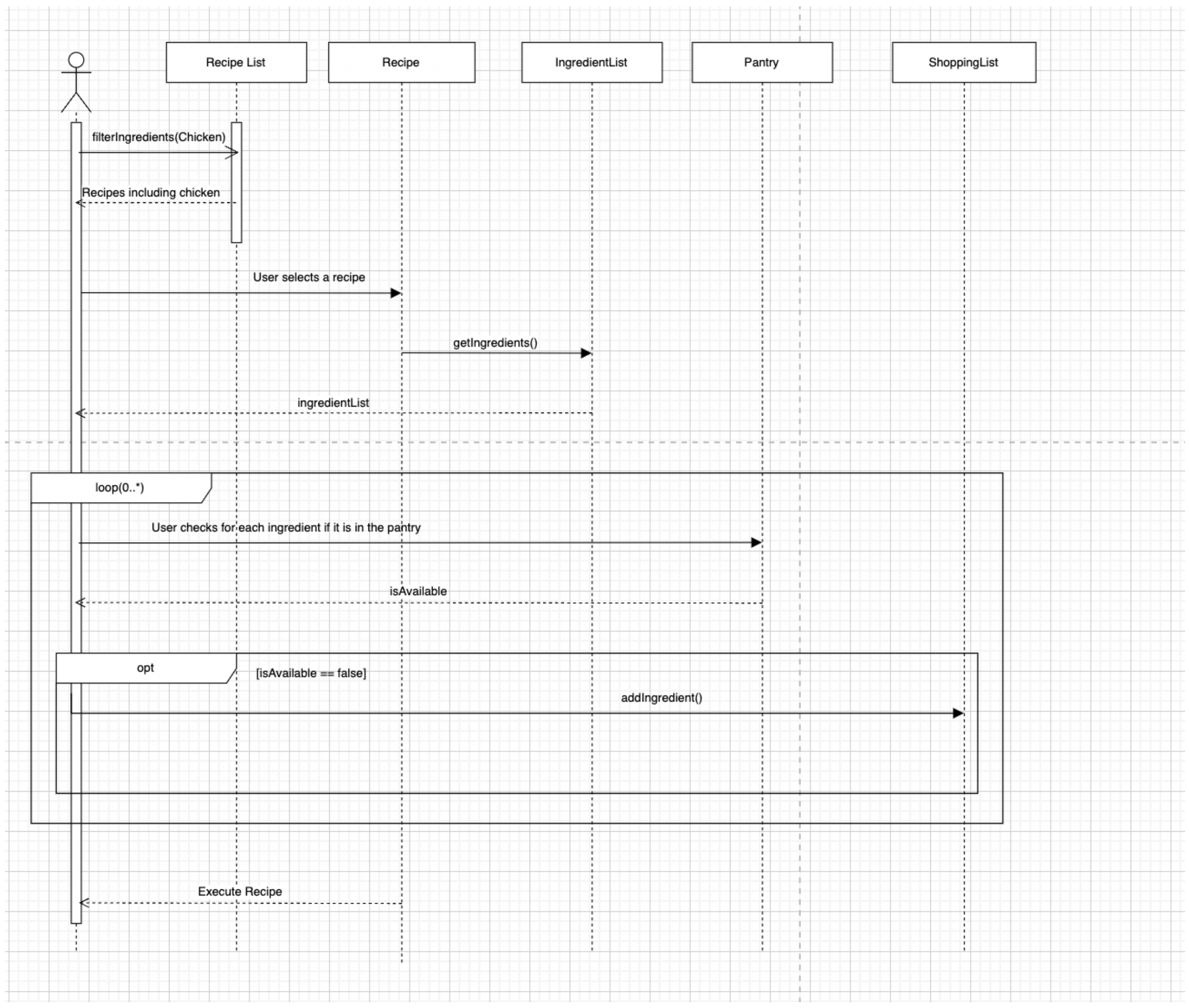
Maximum number of pages for this section: 4

Title: Creating a recipe



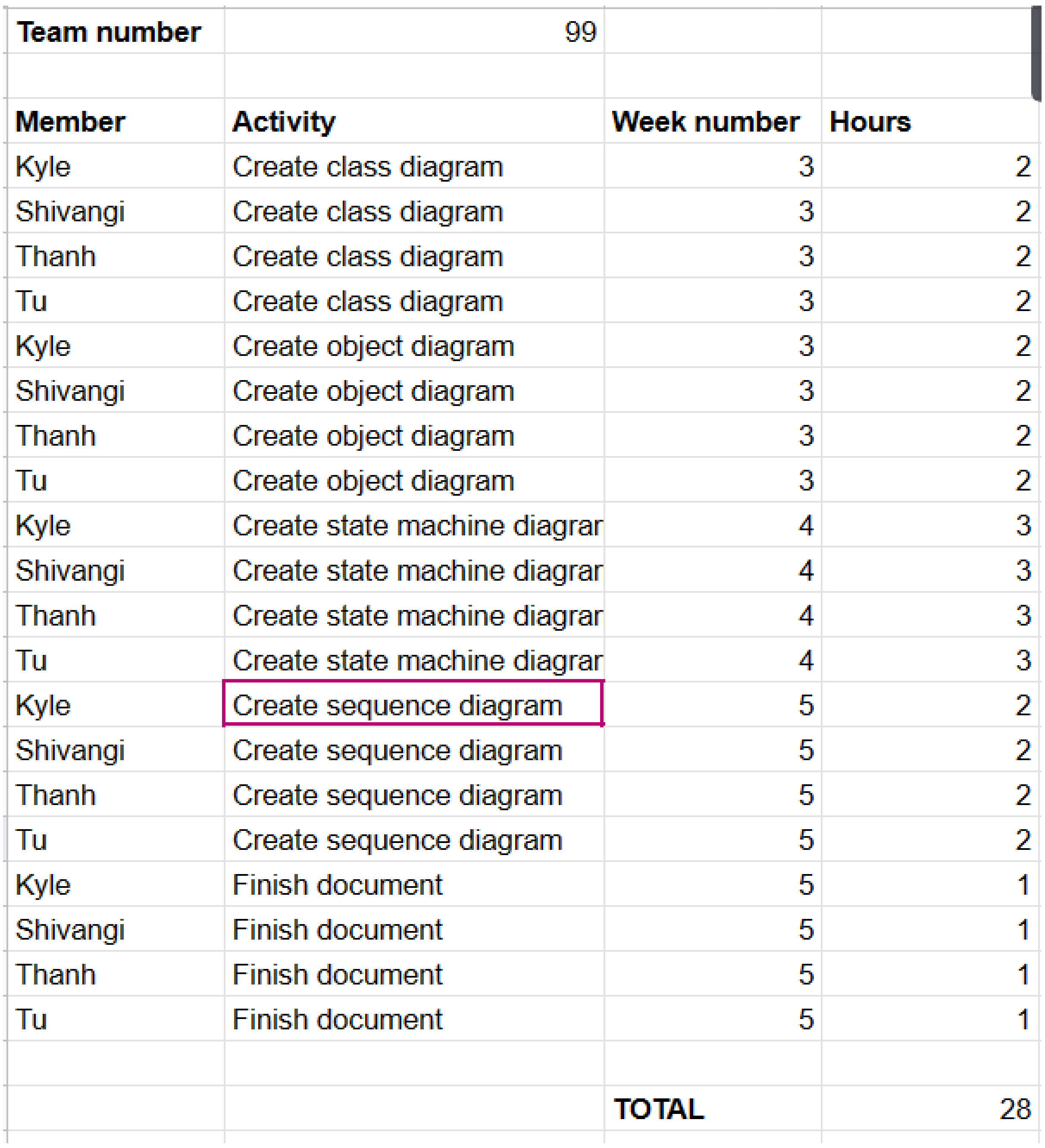
In this situation, the user wishes to create a new recipe and add it to the list of recipes. First, the user is able to use the createRecipe function, which will interact with the Recipe class, and then the Recipe class would respond with an user interface to the user where the user needs to fill in all the ingredients and the step by step instructions, that are necessary for the new recipe. In order for the user to fill in all the necessary ingredients and instructions, the interface will keep on looping and prompting the user to add a new ingredients and/or instructions one at a time until the user has entered all the ingredients and instructions that are required for the new recipe. When the user is finished adding all the necessary ingredients and instructions to the new recipe, the classes Recipe and RecipeList will be interacting with each other with the add() function provided by the RecipeList class where the user will then be able to add the new recipe to the recipe list where, and finally the RecipeList class will return the updated recipe list that now includes the new recipe that was added by the user.

Title: Finding a recipe



In the figure above, we are describing a situation where the user is finding a recipe based on a specific ingredient, in this example the user is looking for a recipe which consists of chicken. The first class the user interacts with is the recipeList class by calling the filterIngredients function (the function call is filterIngredients(chicken)). In response to this request, the recipeList class provides a list of recipes that includes the ingredient chicken. Once that is done the user selects a particular recipe which interacts with the class Recipe. From there, the Recipe class interacts with the ingredientList class by calling the getIngredients() function. After the interaction between the Recipe and ingredientList classes, the ingredientList class in response, provides all the ingredients in the ingredient list to the user on their interface. Later on, in order to check if each and every ingredient from the ingredient list is available in the pantry, we loop through each ingredient in the ingredient list and return for each ingredient if it is available or not using the isAvailable function. Within the loop, we also have an opt fragment which checks the guard, where the guard is isAvailable == false. When the condition isAvailable == false is taking place, then the user interacts with the shoppingList class using the addIngredients() function, which adds ingredients to the shopping list. Once this loop ends, the user continues to execute the recipe where the interaction takes place between the Recipe class and user.

Time logs



Team number	99		
Member	**Activity**	**Week number**	**Hours**
Kyle	Create class diagram	3	2
Shivangi	Create class diagram	3	2
Thanh	Create class diagram	3	2
Tu	Create class diagram	3	2
Kyle	Create object diagram	3	2
Shivangi	Create object diagram	3	2
Thanh	Create object diagram	3	2
Tu	Create object diagram	3	2
Kyle	Create state machine diagrar	4	3
Shivangi	Create state machine diagrar	4	3
Thanh	Create state machine diagrar	4	3
Tu	Create state machine diagrar	4	3
Kyle	Create sequence diagram	5	2
Shivangi	Create sequence diagram	5	2
Thanh	Create sequence diagram	5	2
Tu	Create sequence diagram	5	2
Kyle	Finish document	5	1
Shivangi	Finish document	5	1
Thanh	Finish document	5	1
Tu	Finish document	5	1
		TOTAL	28

Time log SD - Google Sheets