SMART REFRIGERATOR SYSTEM

Software Design Specification

2022. 05. 15

Introduction to Software Engineering

Team 10

Team Leader	Kwak Junyeong
Team Member	Kim Seyoung
	Kim Seongjun
	Park Suyeon
	Park Woohyeon
	Park Junyeong
	Amirah

Contents

list of figures

List of tables

1. Preface

This chapter contains the readership information, readership, scope, objective of this document and the document structure of this Software Design Document for Smart Refrigerator System.

1.1. Readership

This Software Design Document is divided into 11 sections with various subsections. Each section contains the overall structure of the system, the structure at the front end, the structure at the back end, the structure at the AI, protocols, database design and testing plan, and development plan. The detailed structure of the Software Design Document can be found as listed below, in the Document Structure subsection of this SDD. In this document, Team 10 is the main reader. Additionally, professors, TAs, and team members in the Introduction to Software Engineering class can be the main readers.

1.2. Scope

This Design Specification is to be used by Software Engineering and Software Quality Engineering as a definition of the design to be used to implement smart refrigerator and control application.

1.3. Objective

The primary purpose of this Software Design Document is to provide a description of the technical design aspects for our Smart Refrigerator System. This document describes the software architecture and software design decisions for the implementation of embedded system in refrigerator and control application. It also provides an architectural overview of the system to depict different aspects of the system. It further specifies the structure and design of some of the modules discussed in the SRS document and in addition, displays some of the use cases that have been transformed into sequential and activity diagrams, including the class diagrams which show how the programming team would implement the specific module. The intended audience of this document is, but not limited to, the stakeholders, developers, designers, and software testers of Smart Refrigerator System.

1.4. Document Structure

1. **Preface:** this chapter describes readership, scope of this document, object of this system, and structure of this document

2. **Introduction:** this chapter describes several tools used for this document, several diagrams used in this document and the references, and object of this project.

3. **Overall System Architecture:** this chapter describes the overall architecture of the system using context diagram, sequence diagram, and use case diagram.
4. **System Architecture - Frontend:** this chapter describes architecture of the frontend system using class diagram and sequence diagram.
5. **System Architecture - Backend:** this chapter describes architecture of the backend system using class diagram and sequence diagram.
6. **System Architecture - AI:** this chapter describes architecture of AI using class diagram and sequence diagram.
7. **Protocol Design:** this chapter describes design of several protocols which are used for communication between client and server.
8. **Database Design:** this chapter describes database design using several diagrams.
9. **Testing Plan:** this chapter describes the testing plan for our system.
10. **Development Plan:** this chapter describes which tools to use to develop the system, constraints, assumption, and dependencies for developing this system.
11. **Supporting Information:** this chapter describes the baseline of this document.

2. Introduction

The purpose of this project is to create a Smart Refrigerator System, which consists of a refrigerator with embedded system, and control application, to provide smart cooking features to users. This system provides 4 smart functions of 'Ingredient cognition', 'Recipe Recommendation', 'Cooking guidance' and 'Auto Thawing'. Users can always check each ingredients' quantity and expiration date, choose a recipe with all the ingredients prepared, watch the guide video of the recipe, and thaw the ingredient in the freezer automatically. This document is a design document and contains information about the design used to implement the project. For design details, refer to the description of the requirements specification.

2.1. Objectives

The diagrams and tools applied to the project are explained in this part.

2.2. Applied Diagrams

2.2.1. UML

UML is a standardized modelling language to facilitate communication between developers in the process of system development such as requirements analysis, system design, and system implementation. UML has the advantage of being a language with a strong expressive power for modelling and a logical notation with relatively few contradictions. Therefore, communication between developers is easy, and it is easy to point out modelling structures that are omitted or inconsistent and can be applied to all systems regardless of the scale of the system to be developed. UML provides a wealth of analysis and design devices for developing object-oriented software based on diagrams such as use case diagrams and class diagrams, so it is expected to be used as an industry standard for a considerable period in the future.

2.2.2. Use Case Diagram

The use case of UML describes the functional unit provided by the system. The main purpose of the Use Case Diagram is for development teams to visualize the functional requirements of the system, including relationships between different use cases as well as the relationships between the system and interacting actors for the main process. Use Case Diagram is used to describe the advanced functions of the system and the scope of the system.

2.2.3. Sequence Diagram

UML's sequence diagram shows a detailed flow of a specific use case or even a part of a specific use case. Sequence Diagram shows the calling relationship between different objects in the sequence, and it can also show different calls to different objects in detail. The sequence diagram is expressed in two dimensions, the vertical shows the message/call sequence in the order of occurrence time, and the horizontal shows the object instance to which the message is transmitted. The sequence diagram is very simple, and class instances (objects) are classified by placing an instance of each class in a box at the top of the diagram.

2.2.4. Class Diagram

UML's class diagram represents how different entities (people, products, data) relate to each other. In other words, it can be said to be the static structure of the system. Class diagrams are mainly used to show implementation classes handled by programmers, and implementation class diagrams show classes like logical class diagrams.

2.2.5. Context Diagram

Context diagram in UML is a diagram that defines the boundary between the system, or part of a system, and its environment, showing the entities that interact with it. Context diagram represents a central system without any details of the internal structure surrounded by all the systems, environments, and activities with which it interacts. The purpose of the context diagram is to focus on external factors and events that must be considered when developing overall system requirements and constraints. Best context diagrams are used to show how systems interoperate at a very high level, or how systems work and interact logically. Context diagrams are the tools you need to develop basic interactions between systems and actors.

2.2.6. Entity Relationship Diagram

When modelling the database structure of a system, the entity relationship diagram is a diagram showing the properties of entities that have unique characteristics that constitute them and the relationships between them in a network-type structure. In the entity relationship diagram, the entity set is represented by a rectangle, the attribute is represented by an ellipse, the entity set and its attributes are connected by a line, the relationship set is represented by a rhombus, and the mapping form of the relationship set is represented by an arrow.

2.3. Applied Tools

2.3.1 Draw.io

Draw.io is a diagram editing tool for creating each diagram. Draw.io is web-based and has a wide range of scalability through Google Drive, One Drive, Drop Box, Github, and just a local device. By using this, you can quickly and easily create Entity Relationship Diagrams (ERDs), Class Diagrams, Sequence Diagrams, Use Case Diagrams, and Context Diagrams.

2.3.2. Adobe Photoshop

Adobe Photoshop is a raster graphics editor developed and published by Adobe Inc. for Windows and macOS. Photoshop can edit and compose raster images in multiple layers and supports masks, alpha compositing and several color models including RGB, CMYK, CIELAB, spot color, and duotone. Photoshop uses its own PSD and PSB file formats to support these features. In addition to raster graphics, Photoshop has limited abilities to edit or render text and vector graphics (especially through clipping path for the latter), as well as 3D graphics and video. Its feature set can be expanded by plug-ins; programs developed and distributed

independently of Photoshop that run inside it and offer new or enhanced features. In this paper, Photoshop is used for creating or editing some features.

2.4. Project Scope

Through this project, users will have an opportunity to easily control their refrigerator with several ingredients and get recommendation of recipes. With continuous updates, we will have a database server that supports detection model with higher accuracy, highly preferred and new recipe recommendation, and the stable auto thawing system. Users will be able to experience a smart and convenient cooking environment.

2.5. References

The user of this SDS may need the following documents for reference:

Team 7, 2021 Spring, Software Design Document, SKKU.

3. System Architecture - Overall

3.1. Objectives

The overall architecture of Frontend, Backend, Computer vision system will be described in this chapter of the document.

3.2. System Organization

The part that mainly deals with interactions between the device and the user such as ingredient checking, addition, deletion, modification, recipe checking, addition, recommendation is called frontend. It shows predetermined views such as buttons, or pages to the user to enable fast and convenient communication between the user and the system. In the backend part, the interaction information collected from frontend is processed by the controller, having it get requested data such as recipe information, ingredient information, thawing information. It can also recommend appropriate recipes referring to the ingredients that the user has in his/her refrigerator. The computer vision system is a system that cognizes ingredients being put into the refrigerator using a specific lens attached. Based on the given image, the system determines what food the given image is demonstrating, and based on that determination, the related information such as its name, quantity is stored in the database via the controller.

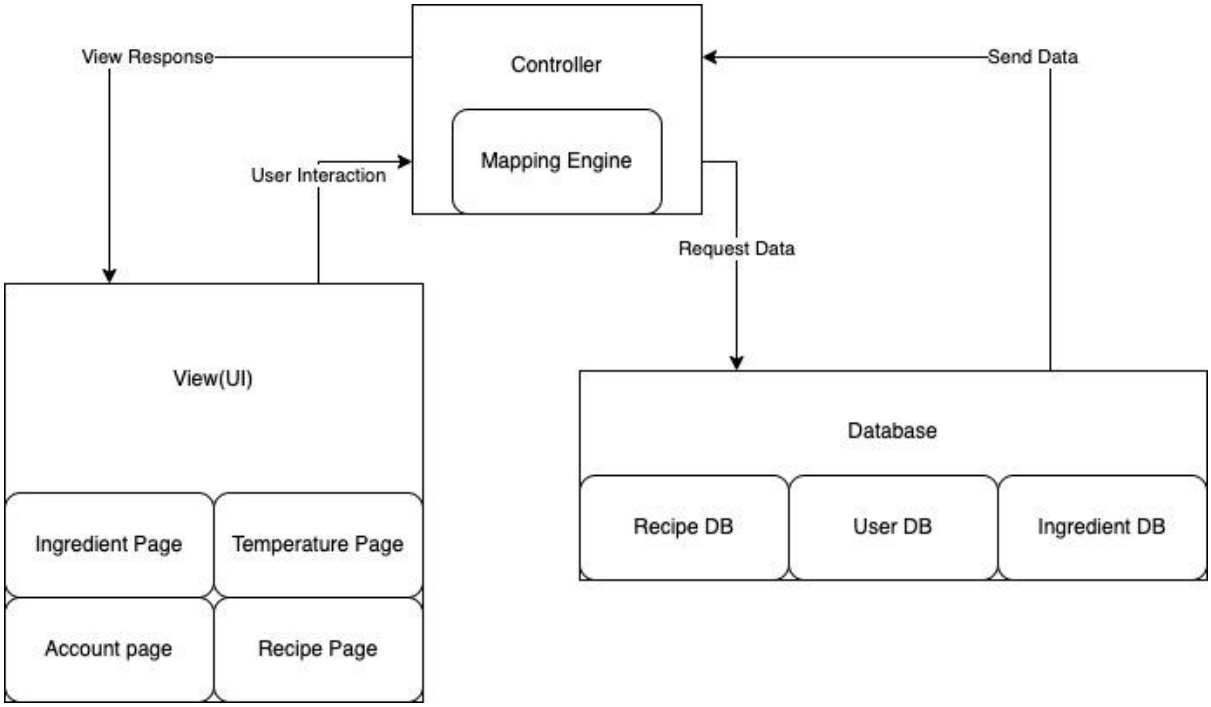

Figure 1 - Overall System Architecture

3.2.1. Context Diagram

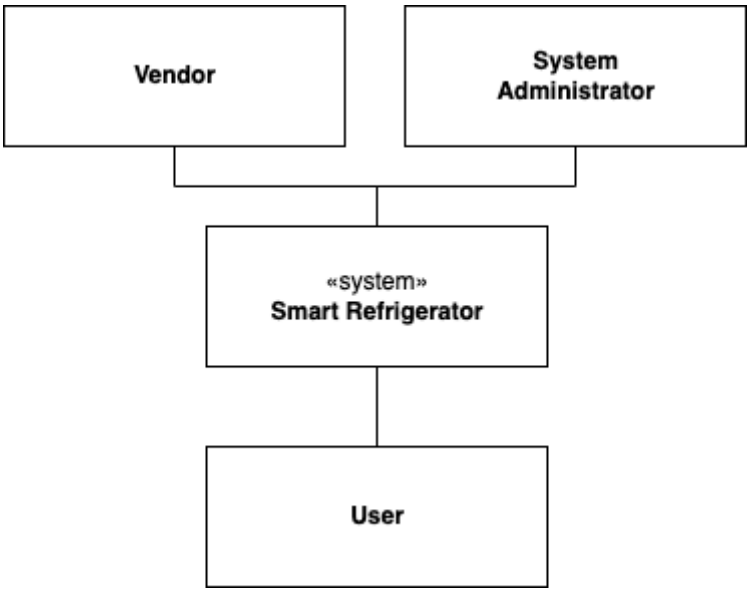

Figure 2 - Overall Context Diagram

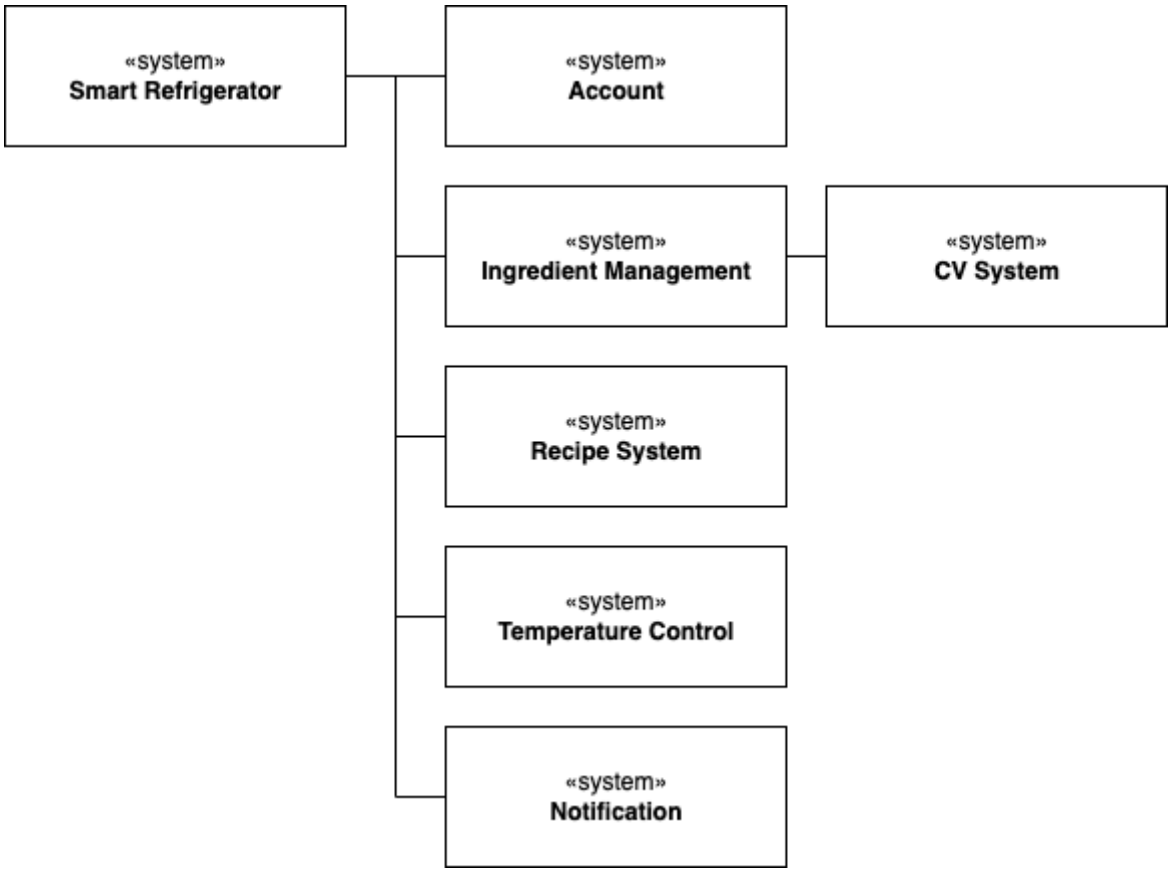

Figure 3 - Overall System Context Diagram

3.2.2. Sequence Diagram

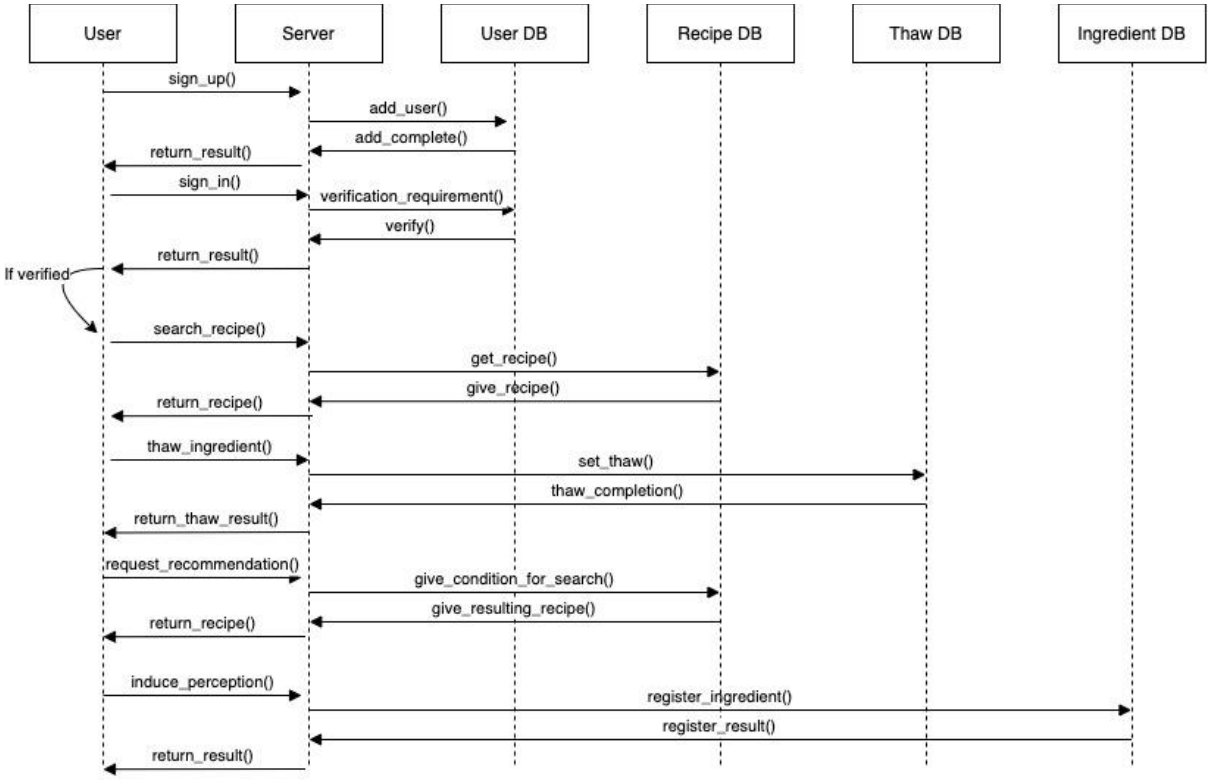

Figure 4 - Overall System Diagram

3.2.3. Use Case Diagram

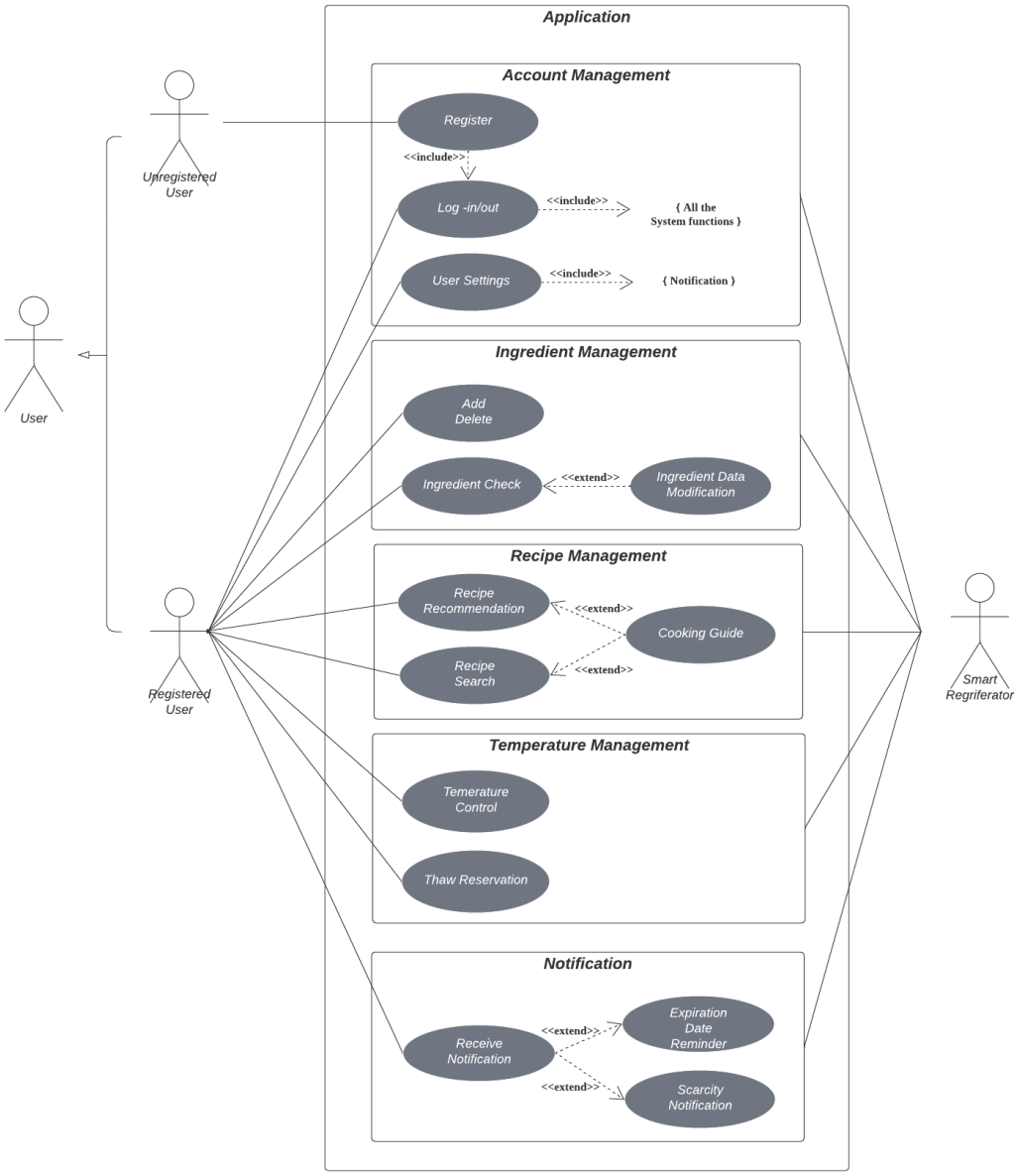

Figure 5 - Overall Use Case Diagram

4. System Architecture - Frontend

4.1. Objectives

This chapter describes the architecture of the frontend system, including the attributes and methods of each subcomponent and the relationship between components.

4.2. Subcomponents

4.2.1. Account Management

The account class manages operation related to the registration and log-in. When a new user wants to register to the system, the e-mail address as an ID, password, and refrigerator ID are required. The account is set to an active state after the e-mail verification. If a user with an account tries to log in to the system, this class checks if the ID and password input is valid and complete.

4.2.1.1. Attributes

These are the attributes that the account class has.

- user_id: user ID
- user_pw: user password, gets hashed when stored in the server
- refrigerator_id: unique ID that each refrigerator has

4.2.1.2. Methods

These are the methods that the account class has.

- set_account()
- get_account()
- validate_account()

4.2.1.3. Class Diagram

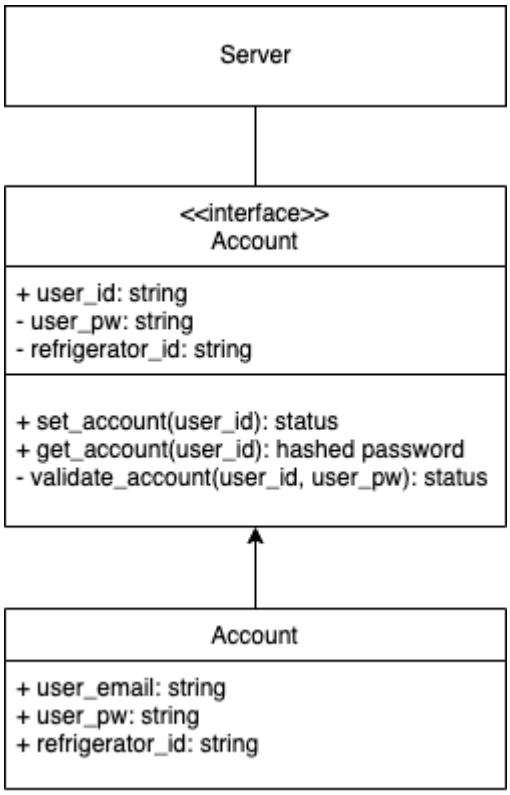

Figure 6 - Account Class Diagram

4.2.1.4. Sequence Diagram

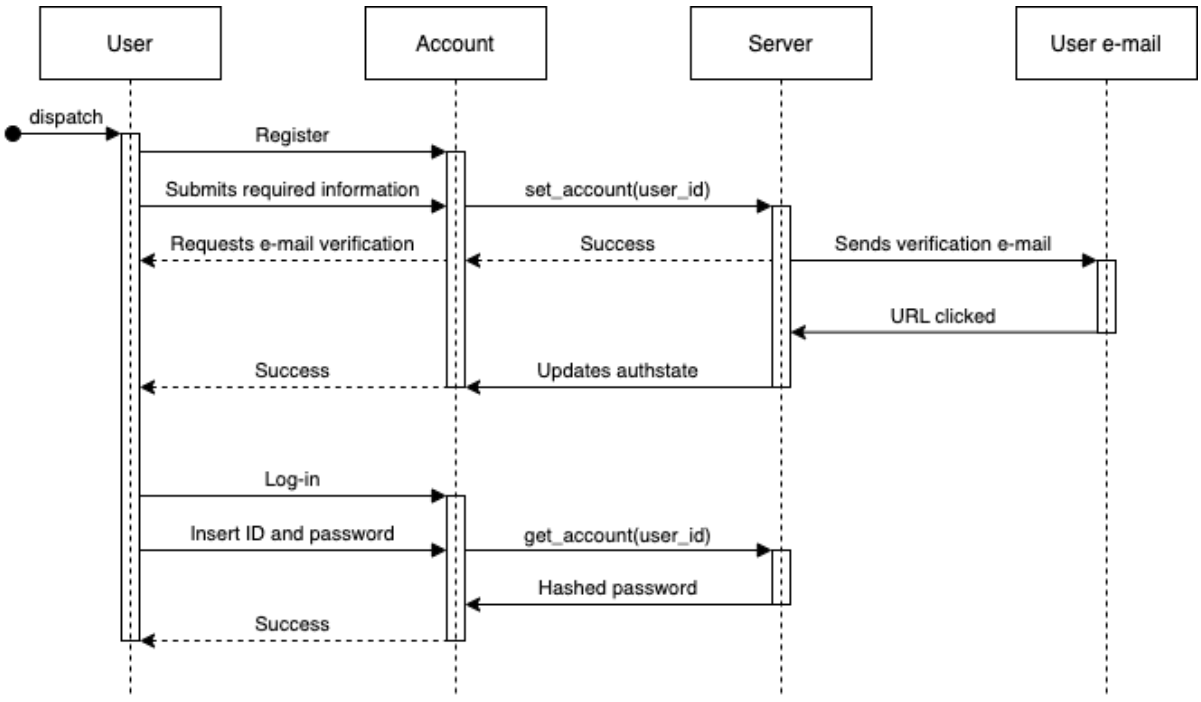

Figure 7 - Account Management Sequence Diagram

4.2.2. Add/Delete/Modify Ingredient

4.2.2.1. Attributes

These are the attributes that the ingredients management class has.

- ingredient_id: ID of cooking ingredient
- user_id: user ID

These are the attributes that the user_ingredient class has.

- ingredient_id: ID of cooking ingredient
- user_id: user ID
- ingredient_name: Name of cooking ingredient
- Img_url: URL where image of ingredient is stored
- Storage: Which storage ingredient stores
- expiration_date: Expiration date of ingredient
- quantity: Quantity of ingredient that is stored
- storage_date: Start date of storage

4.2.2.2. Methods

These are the methods that the ingredients management class has.

- addIngredient()
- deleteIngredient()
- modifyIngerdient()

4.2.2.3. Class Diagram

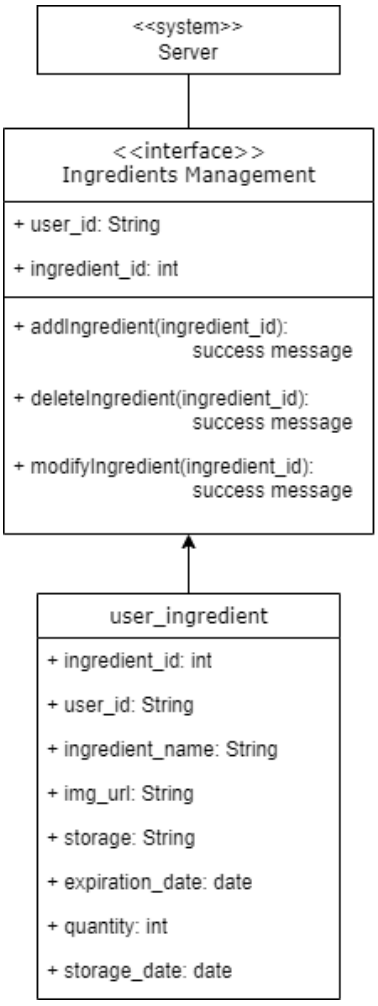

Figure 8 - Ingredients Management Class Diagram

4.2.2.4. Sequence Diagram

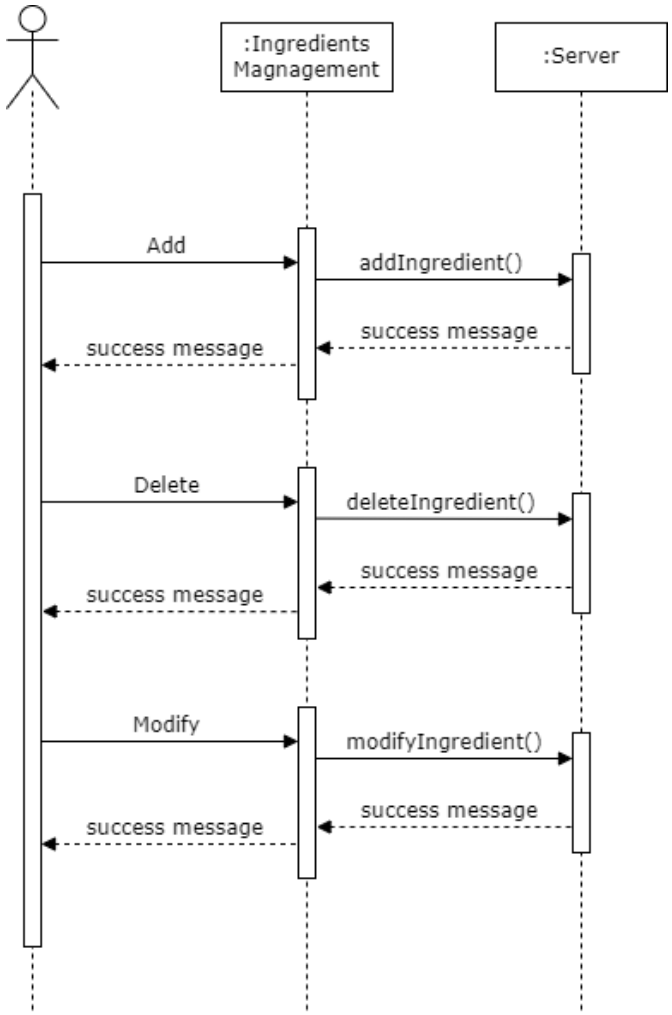

Figure 9 - Ingredients Managements Sequence Diagram

4.2.3. Check Ingredient

4.2.3.1. Attributes

These are the attributes that the ingredients check class has.

- user_id: user ID
- ingredient_info: The information of the search ingredient

These are the attributes that the ingredient class has.

- ingredient_name: Name of cooking ingredient
- Img_url: URL where image of ingredient is stored
- storage: Which storage ingredient stores

- expiration_date: Expiration date of ingredient
- quantity: Quantity of ingredient that is stored
- storage_date: Start date of storage

4.2.3.2. Methods

These are the methods that the ingredients check class has.

- getIngredientInfo()
- showIngredientInfo()

4.2.3.3. Class Diagram

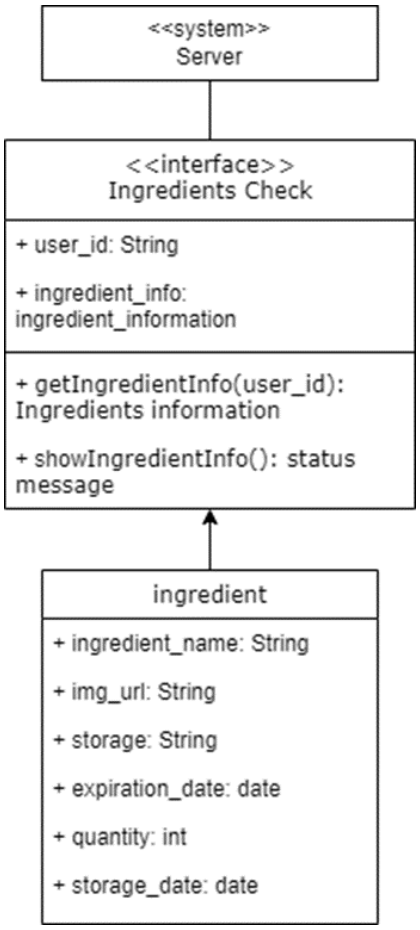

Figure 10 - Ingredients Check Class DIagram

4.2.3.4. Sequence Diagram

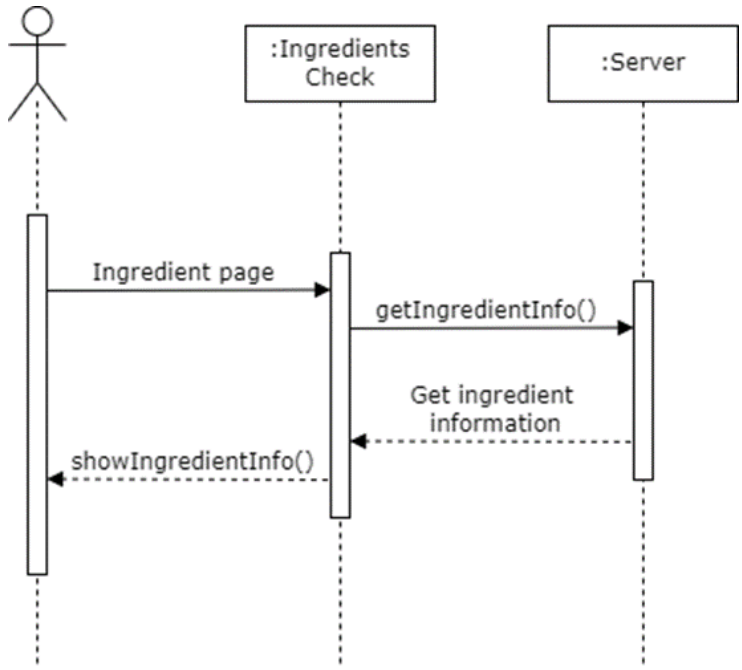

Figure 11 - Ingredients Check Class Diagram

4.2.4. Recipe Recommendation

4.2.4.1. Attributes

These are the attributes that the recipe recommend class has.

- user_id: user ID
- search_record: Recipe search history of user

These are the attributes that the recipe class has.

- recipe_id: ID of recipe
- recipe_name: Name of dish with recipe
- recipe_class: genre of recipe ex) Korean, Japanese
- img_url: URL where image of dishes is stored
- order_url: URL where guide video for cooking is stored

These are the attributes that the recipe search class has.

- recipe_id: ID of recipe
- ingredient_id: ID of cooking ingredient
- expiration_date: Expiration date of ingredient
- quantity: Quantity of material that is stored

4.2.4.2. Methods

These are the methods that the recipe recommend class has.

- getSearchRecord()
- getIngredientList()
- getExpirationDate()
- getRecommnedRecipe()
- showRecmndRecipe()

4.2.4.3. Class Diagram

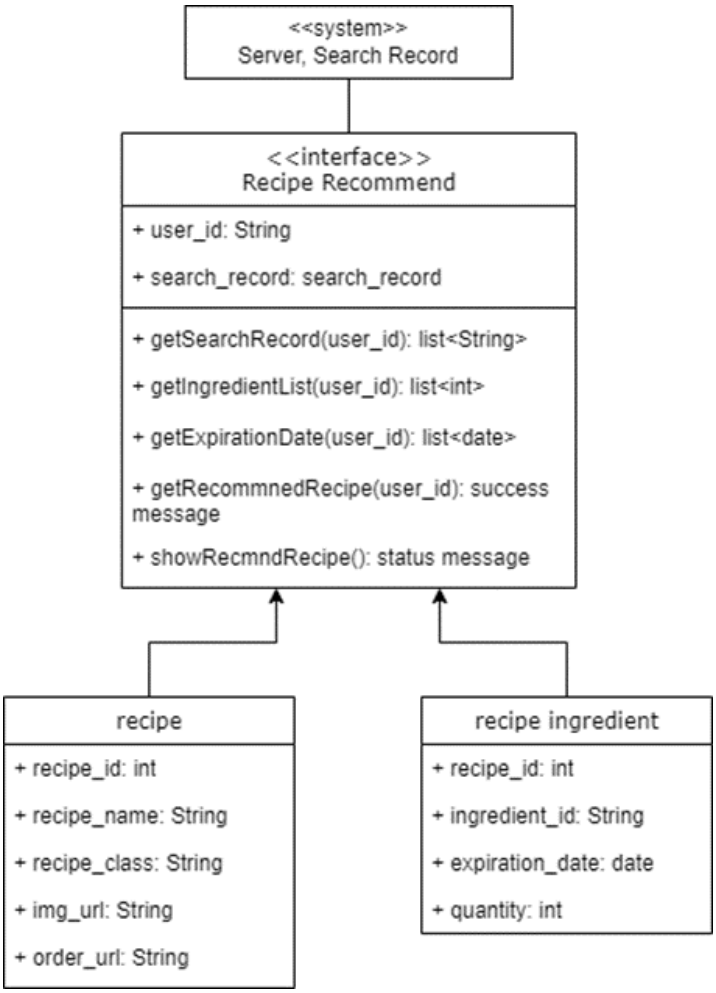

Figure 12 - Recipe Recommendation Class Diagram

4.2.4.4 Sequence Diagram

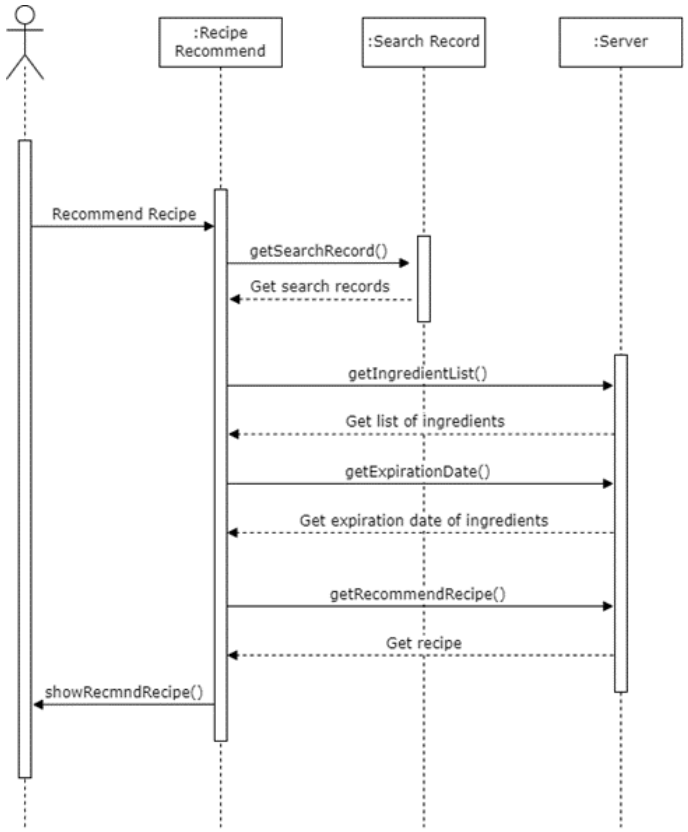

Figure 13 - Recipe Recommendation Sequence Diagram

4.2.5. Recipe Search

4.2.5.1. Attributes

These are the attributes that the recipe search class has.

- user_id: user ID
- filter: search keywords of the user

These are the attributes that the recipe class has.

- filter_recipe: filter condition of recipe_name
- recipe_id: ID of recipe
- recipe_name: Name of dish with recipe
- recipe_class: genre of recipe ex) Korean, Japanese
- img_url: URL where image of dishes is stored

- order_url: URL where guide video for cooking is stored

These are the attributes that the recipe search class has.

- filter_ingredient: filter condition of ingredient_name
- recipe_id: ID of recipe
- ingredient_id: ID of cooking ingredient
- quantity: Quantity of material that is stored

4.2.5.2. Methods

These are the methods that the recipe search class has.

- getSearchFilter()
- getRecipe()
- saveSearchRecord()
- showSearchRecipe()

4.2.5.3. Class Diagram

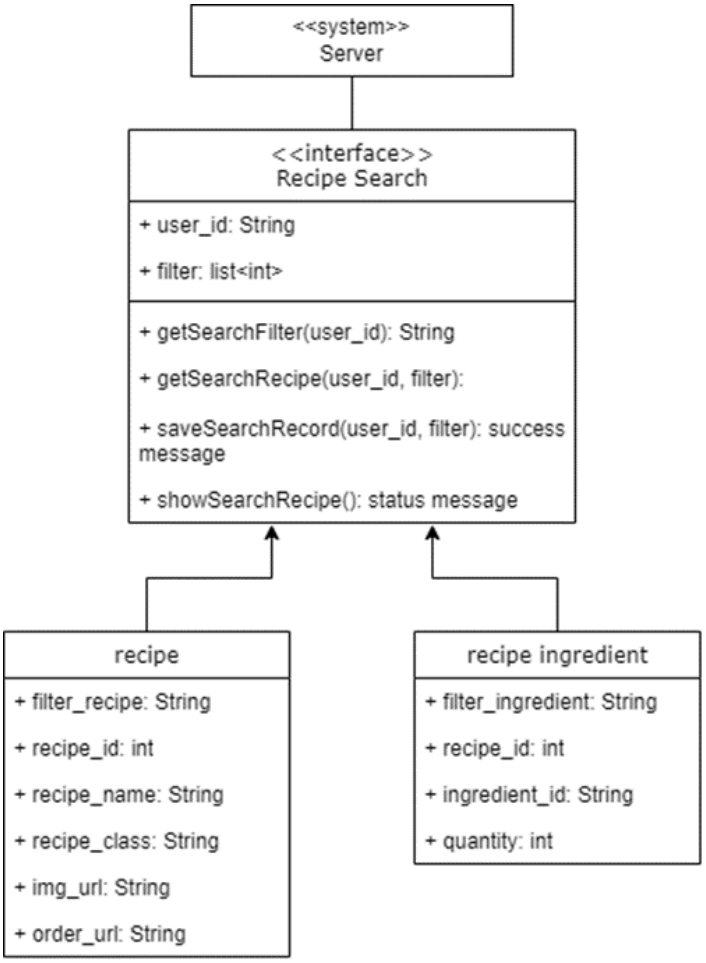

Figure 14 - Recipe Search Class Diagram

4.2.5.4. Sequence Diagram

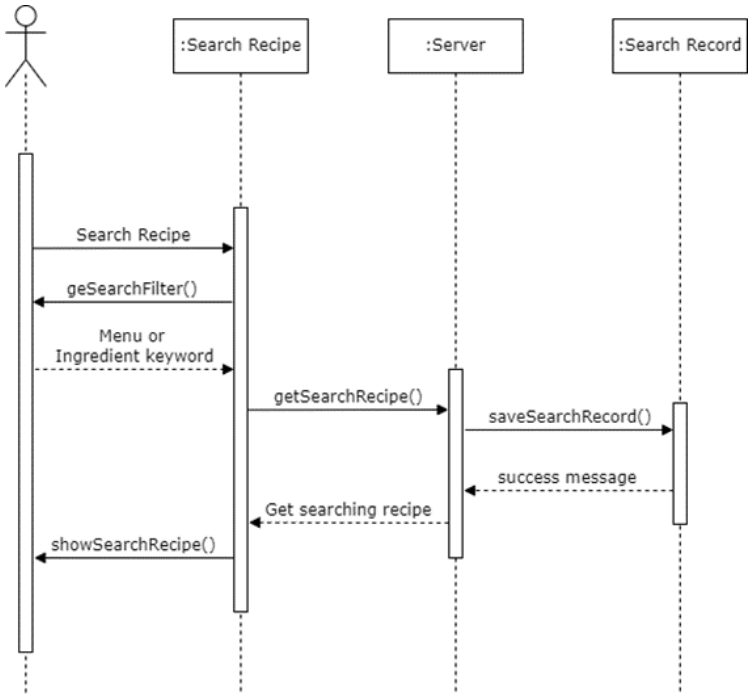

Figure 15 - Recipe Search Sequence Diagram

4.2.6. Temperature Management

This class is related to the temperature control and also the thaw reservation. The user, or the thaw system requests a temperature query consisting of the target components and temperature. The embedded temperature control system interpretes this query and adjusts the cooling level.

4.2.6.1. Attributes

These are the attributes that the Temperature_Query class has.

- target_compartment: the target compartment ID in integer
- temperature: the degree of temperature user has set

These are the attributes that the Thaw_Query class has.

- target_compartment: the target compartment ID in integer
- start_time: when the thawing process should start to meet the user requirements
- end_time: when the thawing process is due
- temperature: the adequate degree of temperature

These are the attributes that the Temperature_Controller class has.

- compartment_map: mapped information of ID and physical compartment
- temperature_query: query for temperature adjustment
- thaw_query: query for thaw reservation

4.2.6.2. Methods

These are the methods that the Temperature_Query class has.

- submit()

These are the methods that the Thaw_Query class has.

- process_thaw_information()
- set_thaw_information()
- submit()

These are the methods that the Temperature_Controller class has.

- set_temperature()
- reserve_temperature_change()

4.2.6.3. Class Diagram

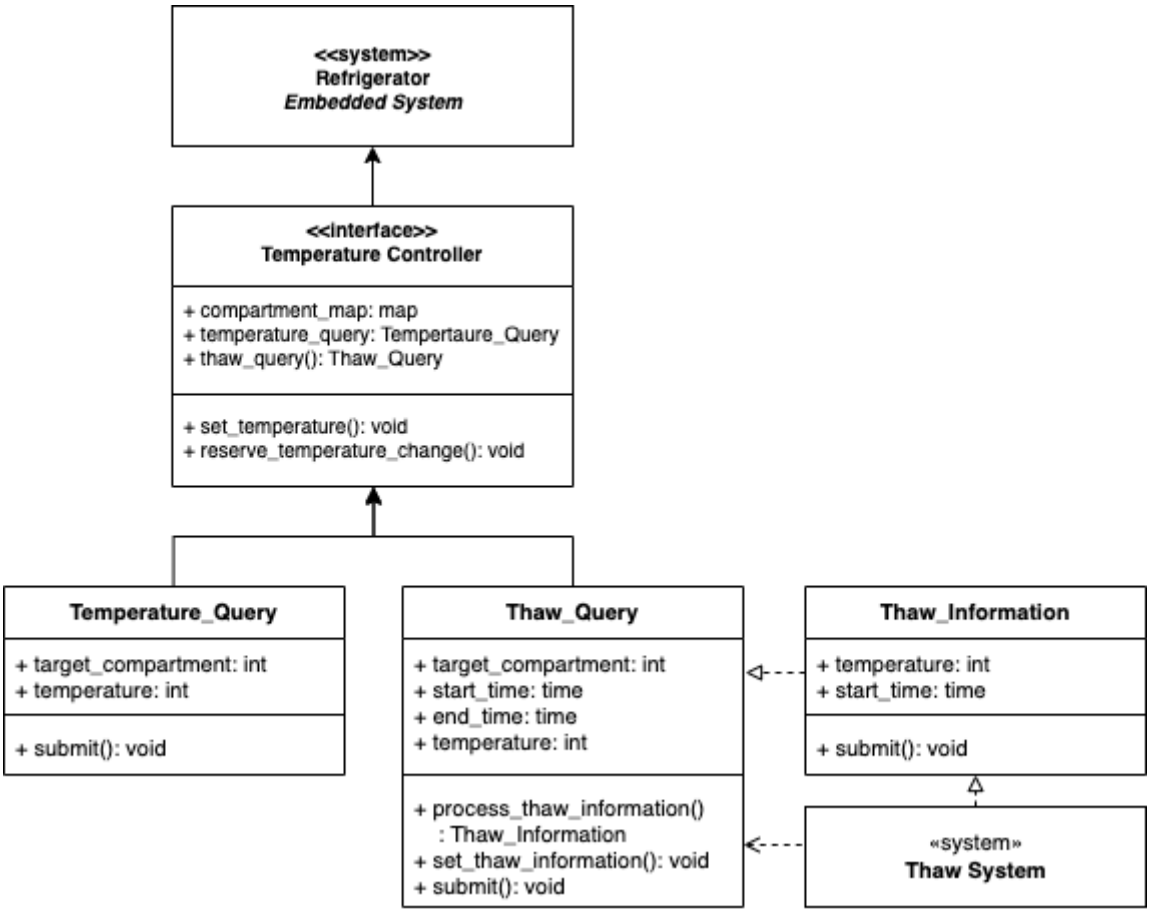

Figure 16 - Temperature Management Class Diagram

4.2.6.4. Sequence Diagram

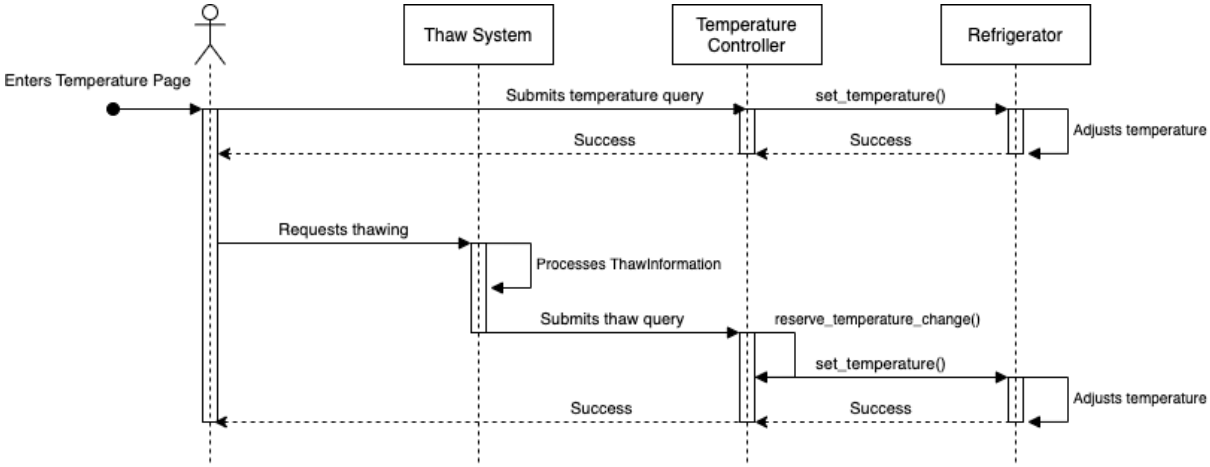

Figure 17 - Temperature Management Sequence Diagram

4.2.7. Notification Management

The notification class is related to the settings and regular check for the expiration date and scarcity notification. Notification_Setter manages add and delete of the user settings of each notification. Notification_Handler gets the list of current notification and sends notification where conditions are met.

4.2.7.1. Attributes

These are the attributes that the Notification_Setter class has.

- user_id: user ID
- ingredient_id: ID of cooking ingredient
- expiration_notify: when to be notified of the coming up expiration date, default set to 3 days
- quantity_notify: amount of this ingredient to get the scarcity notification

These are the attributes that the Notification_Handler class has.

- user_id: user ID

These are the attributes that the Notification class has.

- ingredient_id: ID of cooking ingredient
- expiration_date: Expiration date of ingredient
- quantity: Quantity of ingredient that is stored

4.2.7.2. Methods

These are the methods that the Notification_Setter class has.

- set_notification()
- delete_notification()

These are the attributes that the Notification_Handler class has.

- get_notification()
- notify()

4.2.7.3. Class Diagram

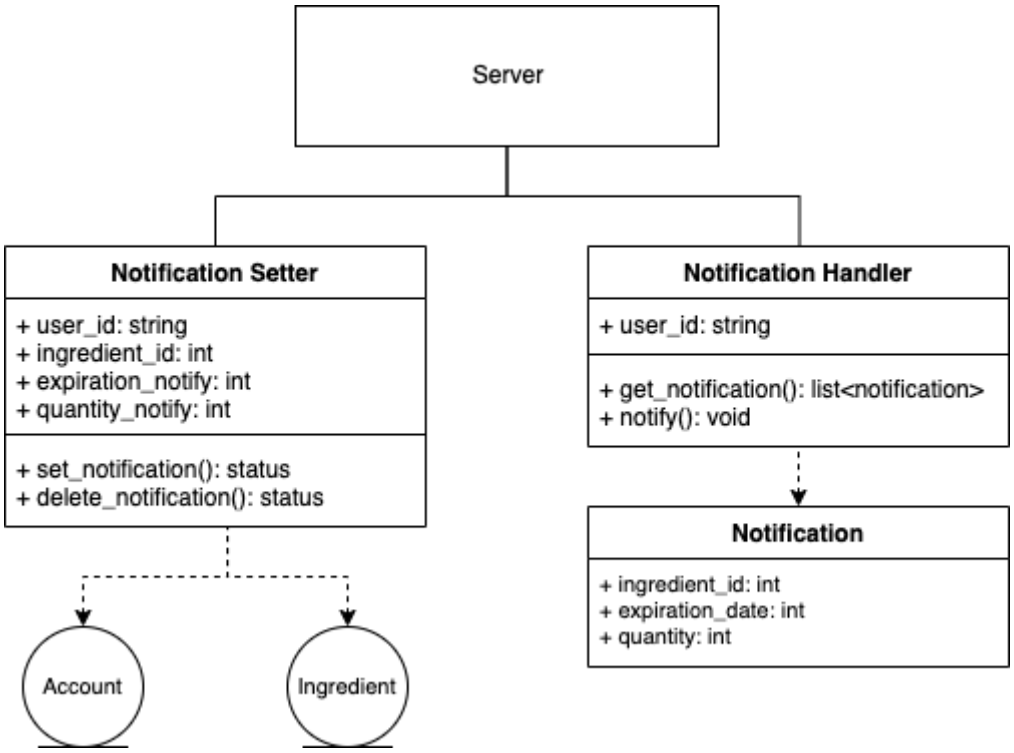

Figure 18 -Notification Management Class Diagram

4.2.7.4. Sequence Diagram

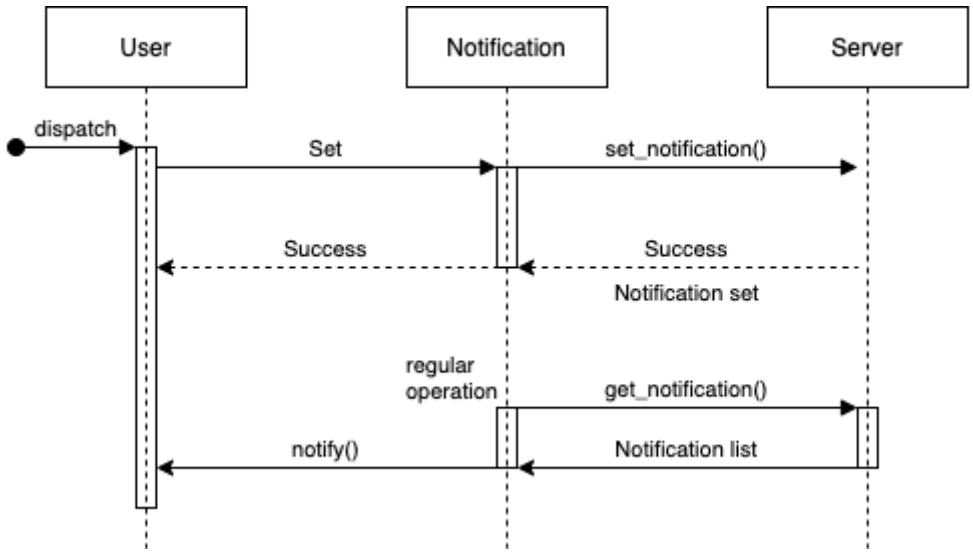

Figure 19 -Notification Management Sequence Diagram

5. System Architecture - Backend

5.1. Objectives

5.2. Overall Architecture

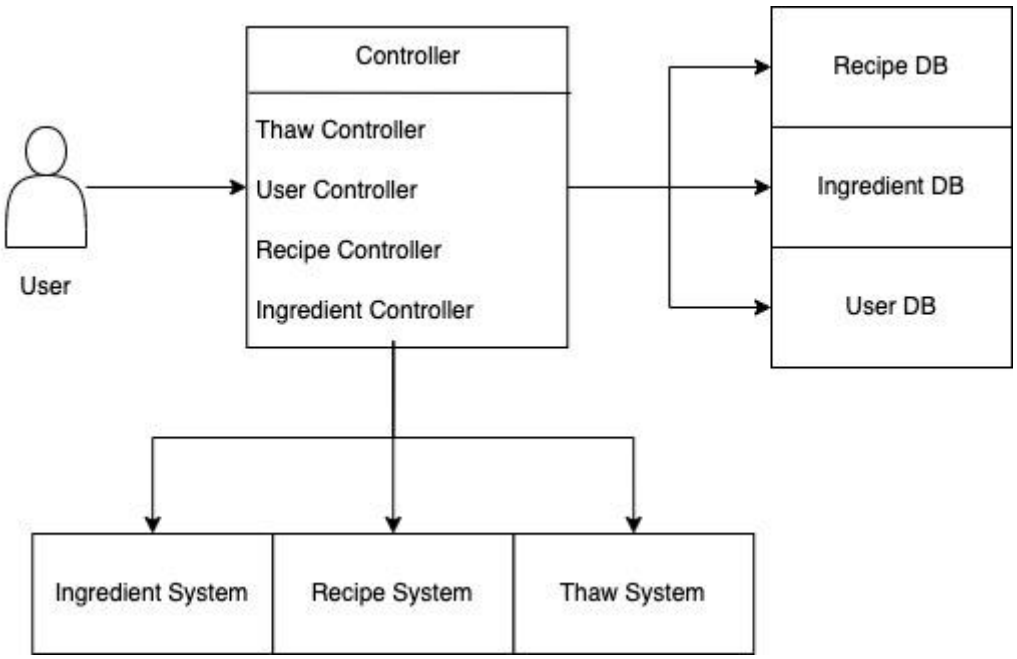

Figure 20 - Overall Architecture of the Backend System

5.3. Subcomponents

5.3.1. Thaw System

5.3.1.1. Class Diagram

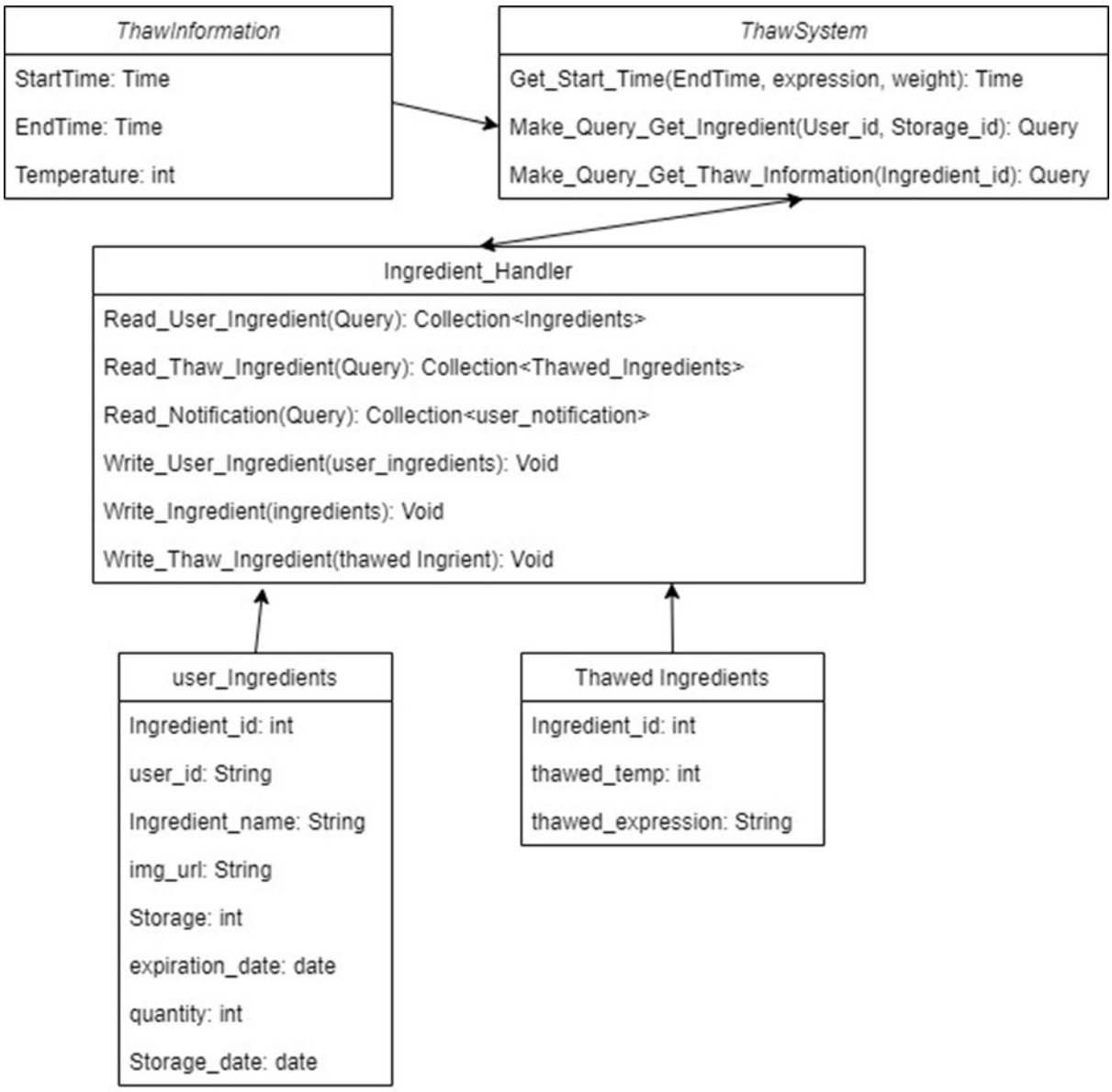

Figure 21 - Thaw System

Thaw Class is an interface for request thawing, User Send End time and storage number to Thaw, then server search DB for what ingredient exist in that storage. (Make_Query_Get_Ingredients) using that ingredient id, Server search database for expression information and temperature information of ingredient (Make_Query_Get_Thaw_Information), Lastly Get Start time and send ThawInformation to user's Refrigerator System.

5.3.1.2. Sequence Diagram

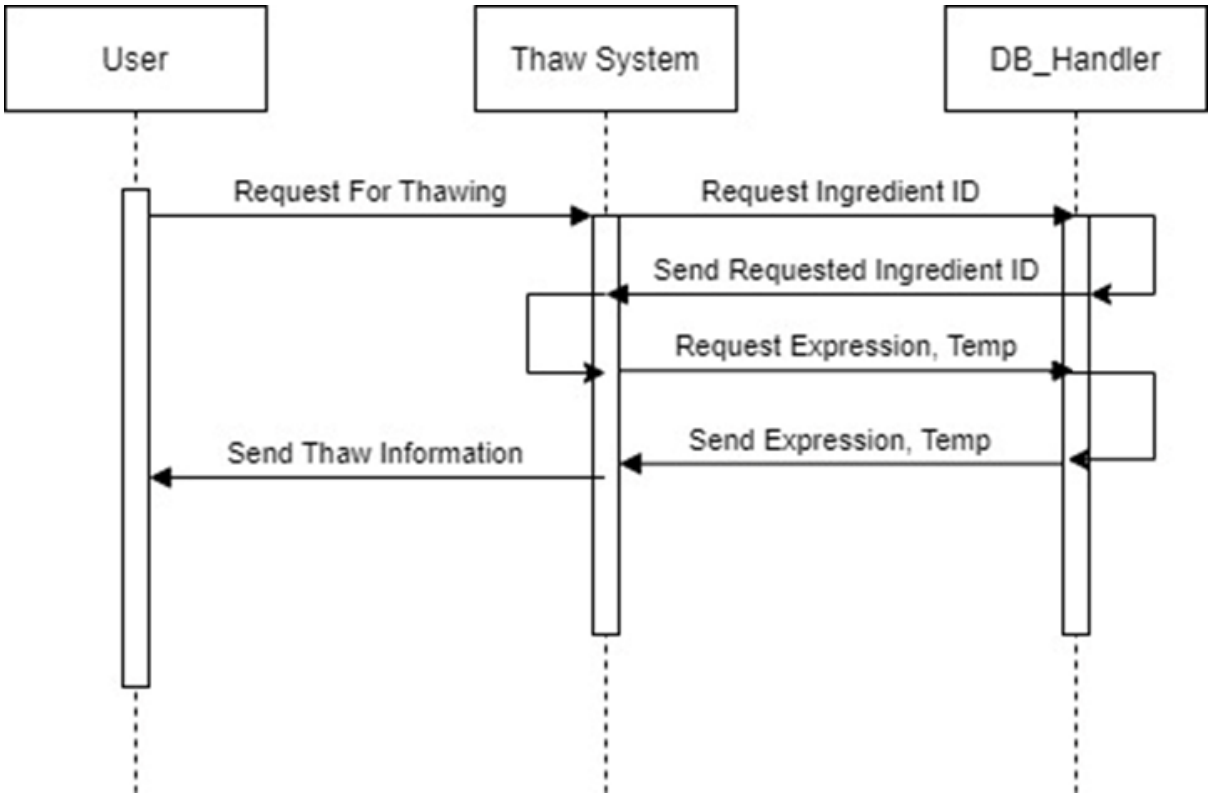

Figure 22 - Thaw System

5.3.2. Ingredient System

5.3.2.1. Class Diagram

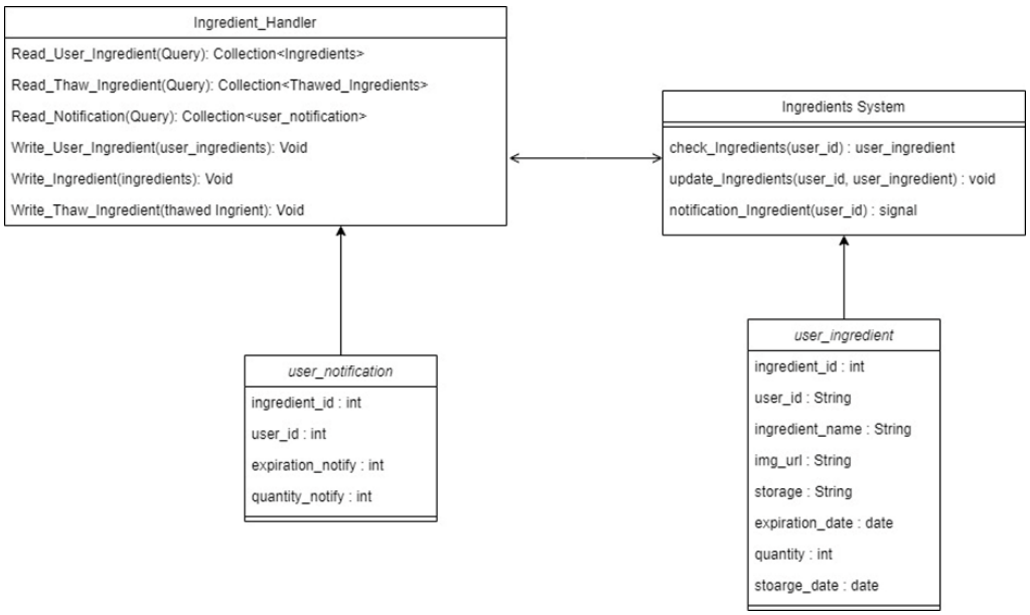

Figure 23 - Ingredient System Class Diagram

The Ingredient System is for overall control of ingredients. You can check the ingredients you have by calling checkIngredient(user_id) with "ingredient_Handler". It can be changed through updateIngredient(user_id) when adding or using ingredients in the refrigerator. In addition, the expiration date and scarcity notifications are called through notificationIngredient(user_id).

5.3.2.2. Sequence Diagram

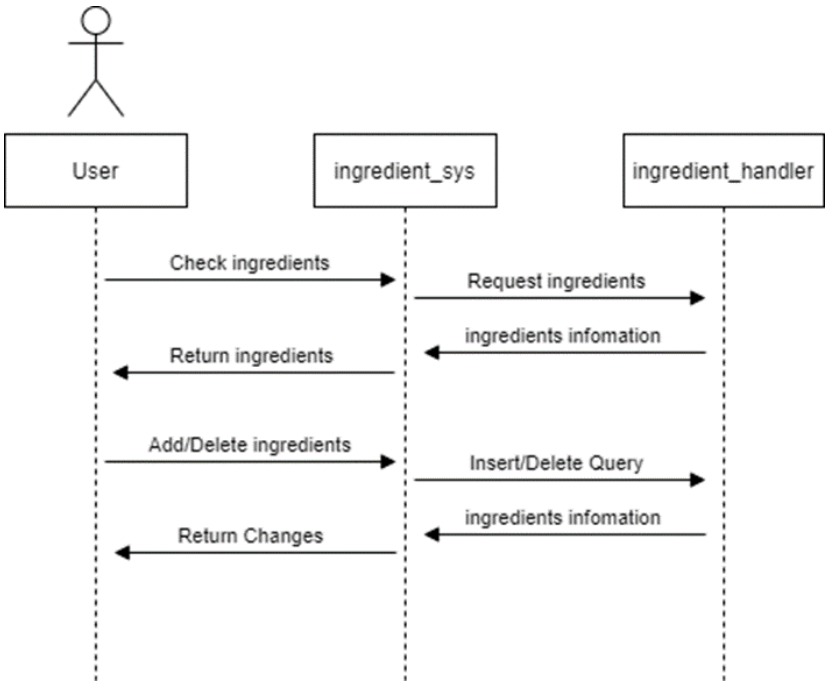

Figure 24 - Ingredient System Sequence Diagram

5.3.3. Recipe System

5.3.3.1. Class Diagram

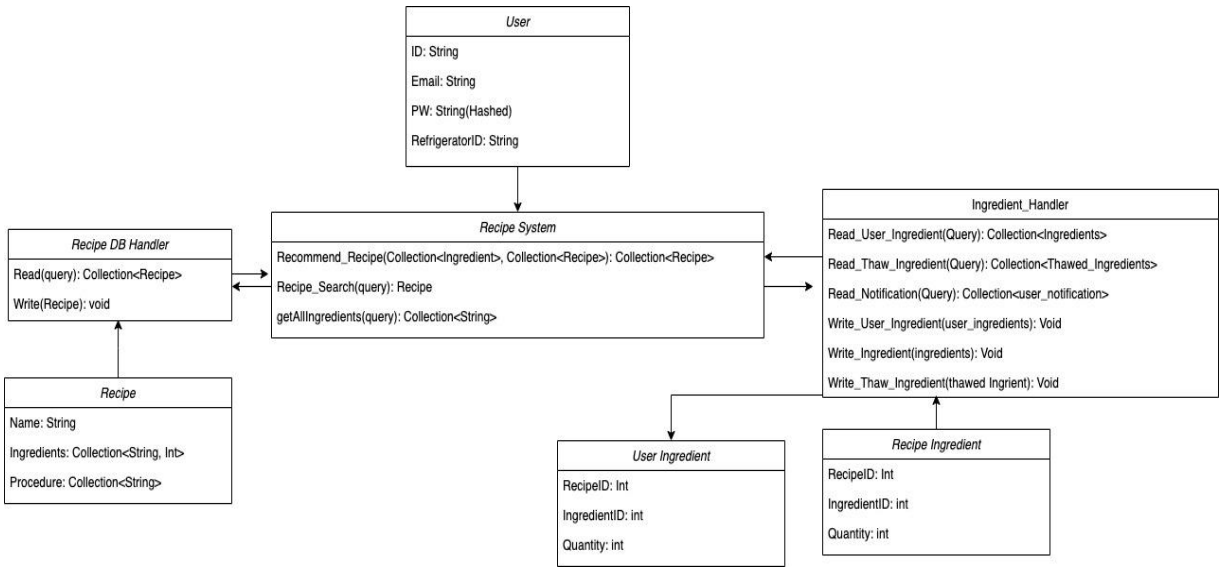

Figure 25 - Recipe System Class Diagram

The Recipe System deals with all actions relevant to recipe such as its inquiry or recommendation. You can check the recipe you have by calling Recipe_Search(query) with "Recipe DB Handler". You can also check what ingredient you have in the refrigerator by calling getAllIngredients(query) with “Ingredient DB Handler”. Lastly, you can get recommendations of recipes by calling Recommend_Recipe(Collection <Ingredient>, Collection<Recipe>) with “Recipe DB Handler”.

5.3.3.2. Sequence Diagram

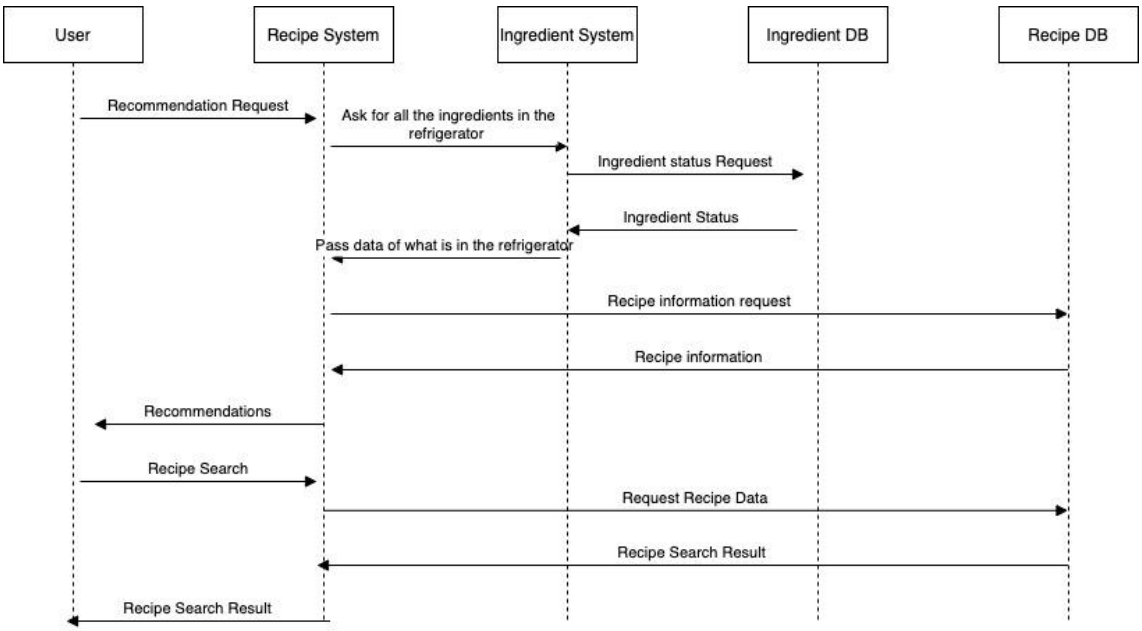

Figure 26 - Recipe System Sequence Diagram

6. System Architecture - AI

6.1. Objectives

This chapter describes the structure of AI system including YOLO and Tesseract OCR.

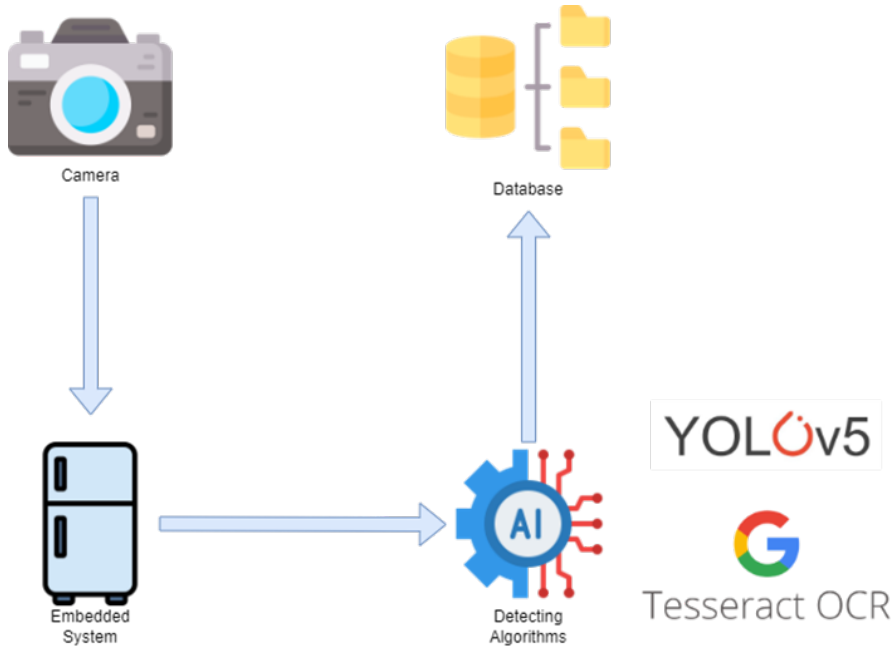

Figure 27 - Overall Architecture

6.2. Overall Architecture

The overall architecture of the system is as above. The camera in the refrigerator captures the image whenever the door of the refrigerator is closed. As the image is captured, the image is sent to the server. At the server YOLO model and Tesseract OCR detects the information of the ingredient in the image (name, quantity, and expiration date). Then the information is bound to a class with additional information of ingredient id and added date, and the class is stored in the database.

6.3. Class Diagram

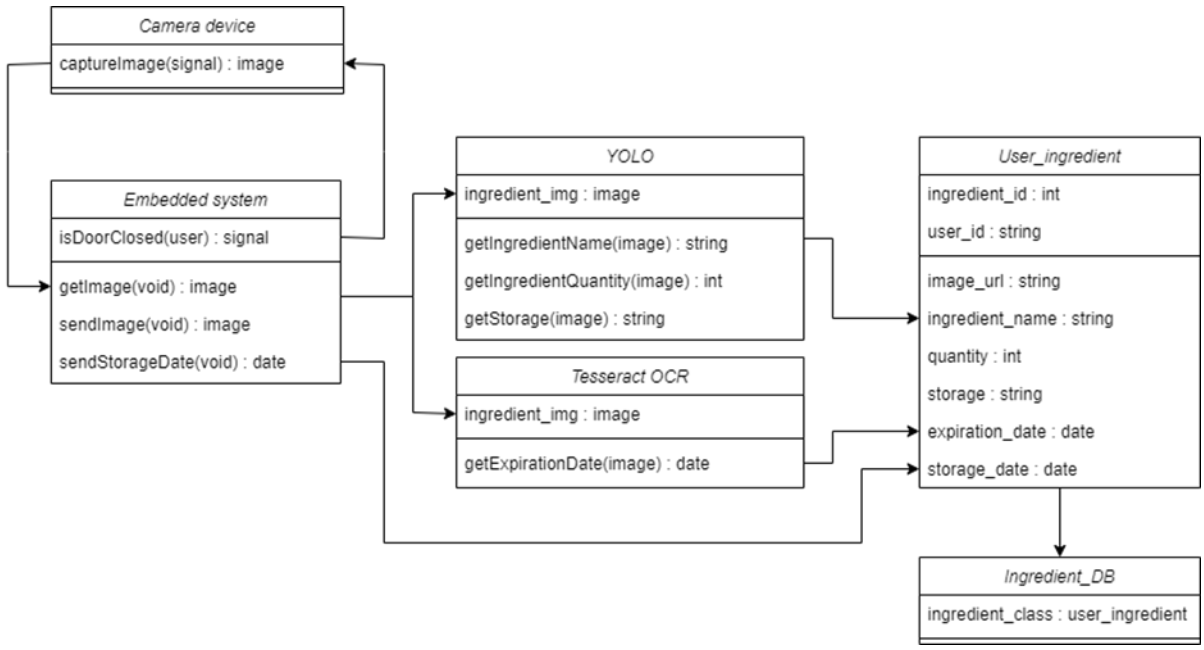

Figure 28 - AI System Class Diagram

Class description

AI System: This is the system for data handling from the raw image to DB class. When the door of the refrigerator is closed, the camera captures the change inside the refrigerator. The image is converted into DB class by AI detection system (YOLO and Tesseract OCR), and stored in DB.

6.3.1. Camera Device

Running a camera always takes too much unnecessary power and makes the embedded system performance lower. Therefore, the camera captures the image only when the door of the refrigerator is closed (which means, there can be a change for ingredients). The signal of the closed door is sent from the embedded system.

6.3.2. Embedded System

Our refrigerator includes an embedded system to get the image of the inside, to communicate with the server, and to provide the refrigerator control interface to the user. Especially in this part, the embedded system receives the image from the camera, and sends it to the server.

6.3.3. YOLO

To recognize the ingredient's information, the system use YOLO image detecting model. This AI works on the server receiving the images from each user's embedded system. The ingredient's name, quantity, and storage are detected from this algorithm, and this information becomes a portion of the ingredient class.

6.3.4. Tesseract OCR

Not only recognizing the ingredient's name and quantity, Tesseract OCR makes expiration date detected if it is specified on the ingredient.

6.3.5. User ingredient

This is the class to store every information of each ingredient. user_id and storage_date is received from the user's embedded system. image_url, ingredient_name, quantity, storage, and expiration_date is received from the YOLO and Tesseract OCR. ingredient_id is created when the class is created. If the class is created, it is stored in the Database.

6.4. Sequence Diagram

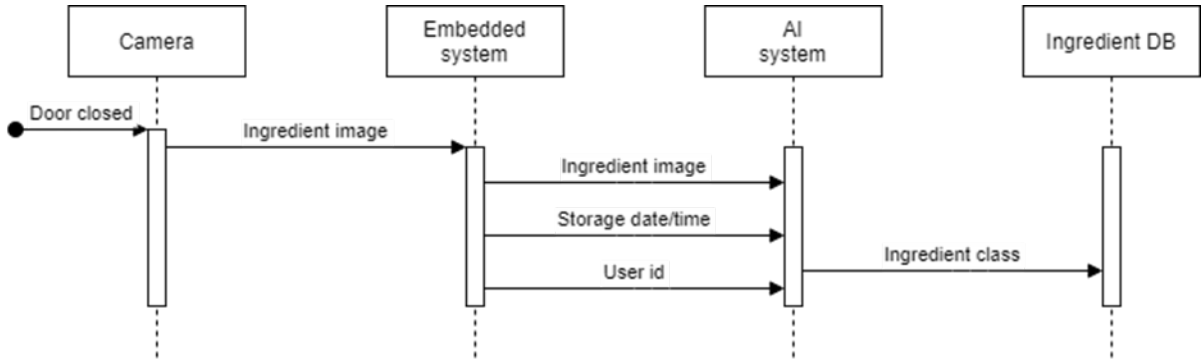

Figure 29 - Review System Sequence Diagram

7. Protocol Design

7.1. Objectives

This part clarifies on which rule and formats the subcomponents of the system communicates with each other.

7.2. Design Basis

7.2.1. HTTP

HTTP stands for HyperText Transfer Protocol and supports application layer communication of hyper-media documents. It is an extensible protocol relying on concepts like resources and Uniform Resource Identifiers, simple message structure and client-server communication flow. The request and response formats of HTTP are in human language and customizable. Tables in this part contain HTTP semantics that the system components would follow.

7.2.2. JSON

JSON is an abbreviated word of Javascript Object Notation, Lightweight data exchange format using key-value format that is often widely used to store or transmit data. It is not dependent on a particular language but similar with JS's object construct. Most Programming Languages provide a library that can handle JSON-formatted data. So, It can be easily created using other programming languages like kotlin, dart and so on. JSON format will be used as the standard in all the data exchange.

7.3. Account System

7.3.1. Registration

- **Request**

Attribute	Detail	
URI	/user/registration	
Method	POST	
Parameter	User ID	User E-mail
	User Password	User password
	Refrigerator ID	ID each refrigerator has

Table 1 - Registration Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
	HTTP 201 Created	
Failure Code	HTTP 400 Bad Request	
	HTTP 404 Not Found	
Success response body	Authorization Token	Registration token
	Message	Message: “Access Success”
Failure response body	Message	Message: “Access Fail”

Table 2 - Registration Response

7.3.2. Log-in

- **Request**

Attribute	Detail	
URI	/user/login	
Method	POST	
Parameter	User ID	User E-mail
	User Password	User password

Table 3 - Login Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
	HTTP 201 Created	
Failure Code	HTTP 400 Bad Request	
	HTTP 401 Unauthorized	
	HTTP 404 Not Found	
Success response body	Authorization Token	Log-in token
	Message	Message: “Access Success”
Failure response body	Message	Message: “Access Fail”

Table 4 - Login Response

7.4. Ingredient System

7.4.1. Add Ingredient

- **Request**

Attribute	Detail	
URI	User/:id/ingredient	
Method	GET	
Parameter	User	Basic User Information
	Contents	User’s ingredient content

Header	Authorization	User authentication

Table 5 - Add Ingredient Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for Access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 6 - Add Ingredient Response

7.4.2. Delete Ingredient

- **Request**

Attribute	Detail	
URI	User/:id/ingredient	
Method	POST	
Parameter	User	Basic User Information
Header	Authorization	User authentication

Table 7- Delete Ingredient Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for Access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 8 - Delete Ingredient Response

7.4.3. Modify Ingredient

- **Request**

Attribute	Detail	
URI	User/:id/ingredient	
Method	POST	
Parameter	User	Basic User Information
	Contents	User’s ingredient content
Header	Authorization	User authentication

Table 9 - Modify Ingredient request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for Access
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 10 - Modify Ingredient Response

7.4.3. Check Ingredient

- **Request**

Attribute	Detail	
URI	User/:id/ingredient	
Method	GET	
Parameter	User	Basic User Information
Header	Authorization	User authentication

Table 11 - Check Ingredient request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for Access
	Ingredient list	Show the list of ingredients
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 12 - Check Ingredient Response

7.5. Recipe System

7.5.1. Search Recipe

- **Request**

Attribute	Detail	
URI	User/:id/recipe	
Method	GET	
Parameter	User	Basic User Information
	Keyword	User’s search keyword
Header	Authorization	User authentication

Table 13 - Search Recipe Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for Access
	Recipe list	Show the list of recipes
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 14 - Search Recipe response

7.5.2. Recommend Recipe

- **Request**

Attribute	Detail	
URI	User/:id/recipe	
Method	GET	
Parameter	User	Basic User Information
	Record	User’s search history
Header	Authorization	User authentication

Table 15 - Recommend Recipe Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 404 (Not found)	
Success response body	Access Token	Token for Access
	User	User's preference
	Recipe list	Show the list of recipes
	Message	Message: "Access success"
Failure response body	Message	Message: "Access fail"

Table 16 - Recommend Recipe Response

7.6. Temperature System

7.6.1. Temperature Check

- **Request**

Attribute	Detail	
URI	/user/:id/temperature	
Method	GET	
Parameter	User	Basic user information
Header	Authorization	User authentication

Table 17 - Temperature Check Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 Bad Request	
	HTTP 401 Unauthorized	
	HTTP 404 Not Found	
Success response body	Access Token	Token for access
	Storage Information	Current temperature state of each compartment
	Message	Message: “Access Success”
Failure response body	Message	Message: “Access Fail”

Table 18 - Temperature Check Response

7.6.2. Temperature Control

- **Request**

Attribute	Detail	
URI	/user/:id/temperature/query	
Method	POST	
Parameter	User	Basic user information
	Target Comparment	Target storage to change temperature

	Temperature	Target temperature
Header	Authorization	User authentication

Table 19 - Temperature Control Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
	HTTP 201 Created	
Failure Code	HTTP 400 Bad Request	
	HTTP 401 Unauthorized	
	HTTP 404 Not Found	
Success response body	Access Token	Token for access
	Message	Message: “Access Success”
Failure response body	Message	Message: “Access Fail”

Table 20 - Temperature Control Response

7.6.3. Thaw System

- **Request**

Attribute	Detail	
URI	User/:id/thaw	
Method	POST	
Parameter	User	Basic User Information

	Storage	Storage to thaw
	Time	End time to thaw
	is_it_app	False
Header	Authorization	User authentication

Table 21 - Thaw Request on Application

Attribute	Detail	
URI	User/:id/thaw	
Method	POST	
Parameter	User	Basic User Information
	Storage	Storage to thaw
	Time	End time to thaw
	is_it_app	False
Header	Authorization	User authentication

Table 22 - Thaw Request on Refrigerator

- **Response**

Attribute	Detail
Success Code	HTTP 200 OK
Failure Code	HTTP 400 (Bad request, overlap)
	HTTP 304 (Forbidden)

	HTTP 404 (Not found)	
Success response body	Start_time	Start time to thaw
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 23 - Thaw Response on Application

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 (Bad request, overlap)	
	HTTP 304 (Forbidden)	
	HTTP 404 (Not found)	
Success response body	Start_time	Start time to thaw
	End_time	End time to thaw
	Temperature	Thaw temperature
	Message	Message: “Access success”
Failure response body	Message	Message: “Access fail”

Table 24 - Thaw response on Refrigerator

7.6. Notification System

7.6.1. Add Notification Settings

- **Request**

Attribute	Detail	
URI	/user/:id/notification	
Method	POST	
Parameter	User	Basic user information
	Ingredient ID	Target ingredient ID
	Expiration Notify	When to notify expiration date
	Scarcity Notify	Amount to get notified
Header	Authorization	User authentication

Table 25 - Notification Addition Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 Bad Request	
	HTTP 401 Unauthorized	
	HTTP 404 Not Found	
Success response body	Access Token	Token for access

	Message	Message: “Access Success”
Failure response body	Message	Message: “Access Fail”

Table 26 - Notification Addition Response

7.6.2. Notification Check

- **Request**

Attribute	Detail	
URI	/user/:id/notification	
Method	GET	
Parameter	User	Basic user information
Header	Authorization	User authentication

Table 27 - Notification Check Request

- **Response**

Attribute	Detail	
Success Code	HTTP 200 OK	
Failure Code	HTTP 400 Bad Request	
	HTTP 401 Unauthorized	
	HTTP 404 Not Found	
Success response body	Access Token	Token for access
	Notification List	List of notification settings
	Message	Message: “Access Success”

Failure response body	Message	Message: “Access Fail”

Table 28 - Notification Check Response

8. Database Design

8.1. Objectives

This section describes the system data structures and how these are to be represented in a database. It identifies entities and ER-diagram (Entity Relationship diagram) Then, it generates SQL DDL (Data Description Language) specification.

8.2. ER diagram

There are seven entities: user, ingredient, recipe, recipe ingredient, user ingredient, thawed ingredient, user notification. ER-diagram expresses each entity and their relationship.

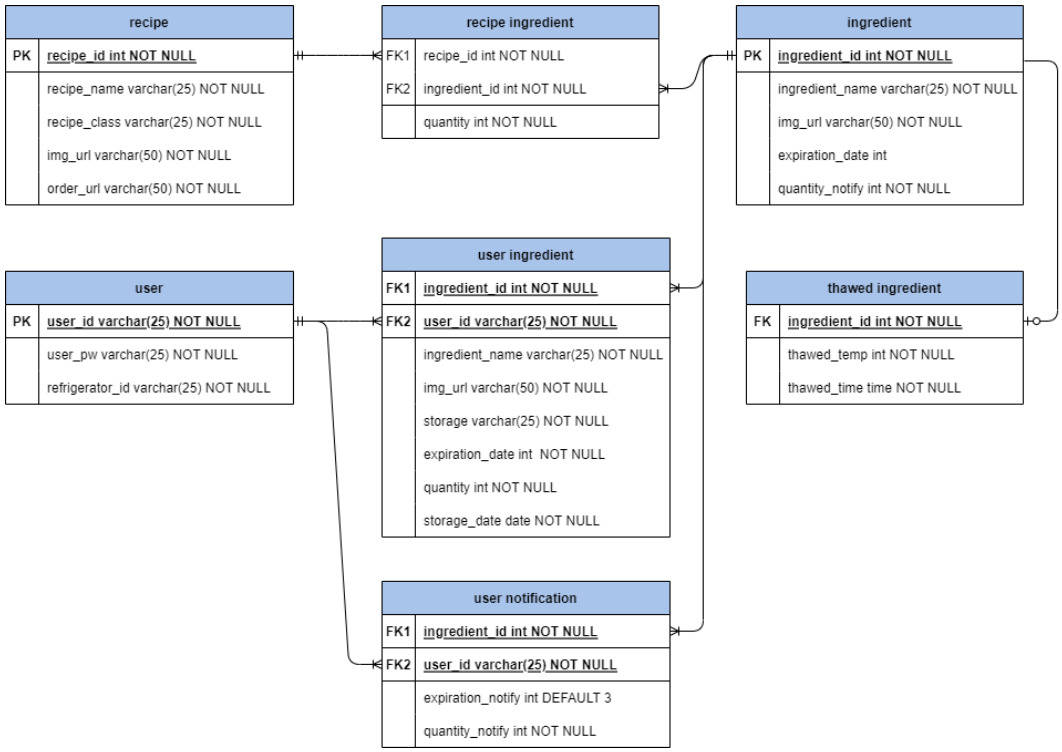

Figure 30 - ER Diagram

8.2.1. Entities

8.2.1.1. User

User entity represents the user of this system. This consists of user_id(PK), user_pw, refrigerator_id. This can control the user's authentication.

user	
PK	user_id varchar(25) NOT NULL
	user_pw varchar(25) NOT NULL
	refrigerator_id varchar(25) NOT NULL

Figure 31 - Entity : User

8.2.1.2. Ingredient

Ingredient entity represents every ingredient the system database has. This consists of ingredient_id(PK), ingredient_name, img_url(image of ingredient path), expiration_date(default date of each ingredient), quantity_notify(default quantity that system notifies scarcity).

ingredient	
PK	ingredient_id int NOT NULL
	ingredient_name varchar(25) NOT NULL
	img_url varchar(50) NOT NULL
	expiration_date int
	quantity_notify int NOT NULL

Figure 32 - Entity : Ingredient

8.2.1.3. Recipe

Recipe entity represents every recipe the system database has. This consists of recipe_id(PK), recipe_name, img_url(image of cooking), order_url(URL of cooking guide).

recipe	
PK	recipe_id int NOT NULL
	recipe_name varchar(25) NOT NULL
	recipe_class varchar(25) NOT NULL
	img_url varchar(50) NOT NULL
	order_url varchar(50) NOT NULL

Figure 33 - Entity : Recipe

8.2.1.4. Recipe ingredient

Recipe ingredient entity represents the quantity of ingredients that the system needs for the chosen recipe. This consists of recipe_id(FK), ingredient_id(FK), quantity.

recipe ingredient	
FK1	recipe_id int NOT NULL
FK2	ingredient_id int NOT NULL
	quantity int NOT NULL

Figure 34 - Entity : Recipe Ingredient

8.2.1.5. User ingredient

User ingredient entity represents ingredients that user has. This consist of ingredient_id(FK), user_id(FK), ingredient_name, img_url(image of ingredient path), expiration_date(default date of each ingredient), quantity, storage(where it stored), storage_date(when it stored).

user ingredient	
FK1	**ingredient_id int NOT NULL**
FK2	**user_id varchar(25) NOT NULL**
	ingredient_name varchar(25) NOT NULL
	img_url varchar(50) NOT NULL
	storage varchar(25) NOT NULL
	expiration_date int NOT NULL
	quantity int NOT NULL
	storage_date date NOT NULL

Figure 35 - Entity : User Ingredient

8.2.1.6. Thawed ingredient

Thawed ingredient entity represents information that ingredient needs to thaw. This consist of default temperature and time that each ingredient needs.

thawed ingredient	
FK	**ingredient_id int NOT NULL**
	thawed_temp int NOT NULL
	thawed_time time NOT NULL

Figure 36 - Entity : Thawed Ingredient

8.2.1.7. User notification

User notification entity represents setting for user notification about each ingredient. This consist ingredient_id(FK), user_id(FK), expiration_notify, quantity_notify.

user notification	
FK1	ingredient_id int NOT NULL
FK2	user_id varchar(25) NOT NULL
	expiration_notify int DEFAULT 3 quantity_notify int NOT NULL

Figure 37 - Entity : User Notification

8.4. SQL DDL

8.4.1. User

```
CREATE TABLE user(
        user_id VARCHAR(25) NOT NULL
        user_pw VARCHAR(25) NOT NULL
        refrigerator_id VARCHAR(25) NOT NULL
        PRIMARY KEY(user_id)
)
```

8.4.2. Ingredient

```
CREATE TABLE ingredient(
        ingredient_id INT NOT NULL
        ingredient_name VARCHAR(25) NOT NULL
        img_url VARCHAR(50) NOT NULL
        expiration_date INT
        quantity_notify INT NOT NULL
        PRIMARY KEY(ingredient _id)
)
```

8.4.3. Recipe

```
CREATE TABLE recipe(
        recipe_id INT NOT NULL
        recipe_name VARCHAR(25) NOT NULL
        img_url VARCHAR(50) NOT NULL
        recipe_class VARCHAR(25) NOT NULL
        order_url VARCHAR(50) NOT NULL
        PRIMARY KEY(recipe _id)
)
```

8.4.4. Recipe ingredient

```
CREATE TABLE recipe_ingredient(
        recipe_id INT NOT NULL
        ingredient_id INT NOT NULL
        quantity INT NOT NULL
        FOREIGN KEY (recipe_id) REFERENCES recipe (recipe_id)
        FOREIGN KEY (ingredient_id) REFERENCES ingredient (ingredient_id)
)
```

8.4.5. User ingredient

```
CREATE TABLE user_ingredient(
        ingredient_id INT NOT NULL
        user_id VARCHAR(25) NOT NULL
        ingredient_name VARCHAR(25) NOT NULL
        img_url VARCHAR(50) NOT NULL
```

```
        storage VARCHAR(25) NOT NULL
        expiration_date INT NOT NULL
        quantity INT NOT NULL
        storage_date DATE NOT NULL
        FOREIGN KEY (user_id) REFERENCES user (user_id)
        FOREIGN KEY (ingredient_id) REFERENCES ingredient (ingredient_id)
)
```

8.4.6. Thawed ingredient

```
CREATE TABLE thawed_ingredient(
        ingredient_id INT NOT NULL
        thawed_temp INT NOT NULL
        thawed _time TIME NOT NULL
        FOREIGN KEY (ingredient_id) REFERENCES ingredient (ingredient_id)
)
```

8.4.7. User notification

```
CREATE TABLE user_notification (
        ingredient_id INT NOT NULL
        user_id VARCHAR(25) NOT NULL
        expiration_notify INT DEFAULT 3
        quantity_notify INT NOT NULL
        FOREIGN KEY (ingredient_id) REFERENCES ingredient (ingredient_id)
        FOREIGN KEY (user_id) REFERENCES user (user_id)
)
```

9. Testing Plan

9.1. Objectives

This part of the document describes our plan of how to test the system that is implemented referring to this document. Unit, Integration, System testing are the subgroups that are included in the whole entire plan. These tests aim to detect all potential errors, since even a single error might cause severe damage to the enterprise.

9.2. Testing Policy

9.2.1. Unit Testing

In the unit(component) testing step, check whether each unit/component that makes up the system operates without any problems. The system consists of several buttons and each button has a space that contains numerous components such as a list of ingredients and food photos. These components should be verified to operate normally in each space, and problems in the operation process should be identified and corrected. Through this process, numerous components can be integrated into one system.

9.2.2. Integration Testing

After unit testing, combine the components one by one and verify that the interaction between the components is working properly. If all components are combined at once and then tested, it becomes difficult to identify the problematic part. Therefore, the test should be repeated several times while combining the components one by one.

9.2.3. System Testing

Combine all components to complete one system/software and verify that it operates normally without any problems. The system tests focusing on performance, reliability, and security.

9.2.3.1. Performance

The system consists of several pages, such as checking and managing food ingredient lists, recommending recipes, and controlling temperature. User who uses the system moves through these multiple pages, so the time required to move pages should be minimized. Therefore, as specified in the proposal, the time required to move the page should be within 5 seconds and the results should be provided to the user. It should be tested assuming various situations, such as when two or more users simultaneously access the same smart refrigerator. Recommendation and notification systems, such as recipe recommendation or expiration date notification, should not exceed 5 percent of the average number of errors in response. It should be tested for missing or conflicting information.

9.2.3.2. Reliability

In order to increase the user's reliability to the system, it is necessary to eliminate errors that may occur when the user uses the system. To this end, each sub-component constituting the system must be implemented without error, and the interaction between components must be accurately combined into one system according to the architecture. Therefore, the development test should proceed from the unit development stage. This should be repeated whenever other components are added.

9.2.3.3. Security

Protecting users' personal information is an important issue for developers to deal with. Personal information should not be leaked to the outside world and should be protected from unauthorized access by unauthorized users.

For the security of the system, security issues should be identified and corrected through manual code review by accessing a nearly complete version of the mobile application and embedded system. This should be repeated even after the development and evolution of each version of the system is completed. In addition to direct analysis at the developer level, application analysis can be performed automatically through services such as Ostorlab.

9.2.4. Acceptance

Since the development testing phase has prepared the system apt for deployment, the system will be tested in terms of the actual use in this phase. Whether the system provides completeness for the given requirements, and thus qualifies the ultimate purpose—to give more convenience and experience to the user in order to keep them cooking at home—will be evaluated with the user feedback.

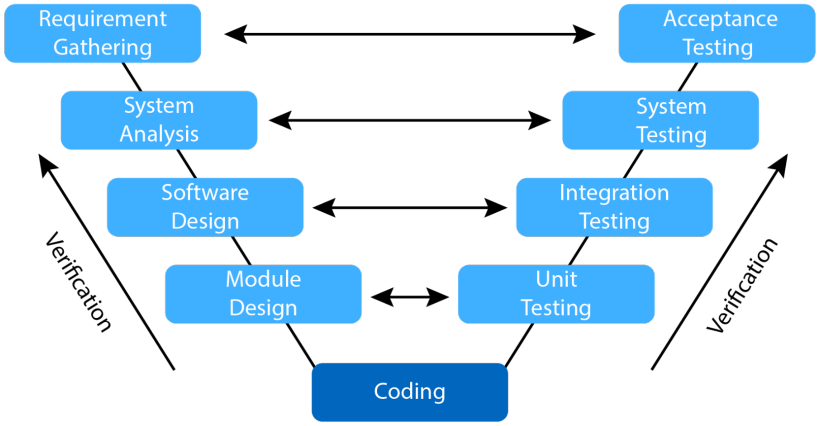

Figure 38 - Testing in Software Development Life Cycle

To maintain or enhance the system reliability to a reasonable level, the CV system and the recipe recommendation algorithm may need a regular update to provide the best user experience. Report cases would be collected. The target cases will be: user modification of the ingredient information for the CV system, and the precision and recall rate for the recommendation result for the recommendation system.

10. Development Plan

10.1. Objectives

This part introduces the programs and development tools used to construct the system environment.

10.2. Frontend Environment

10.2.1. Adobe Photoshop: UI Design

Figure 39 - Adobe Photoshop Logo

Adobe Photoshop is a photo editing and raster graphic design software. Since the system has two discrete user interfaces, the refrigerator LCD screen and the application, providing a suitable design for each is one of the important tasks in frontend development. This program would be used to create layout and icon designs for a pleasant user experience.

10.2.2. Figma: UI/UX Design

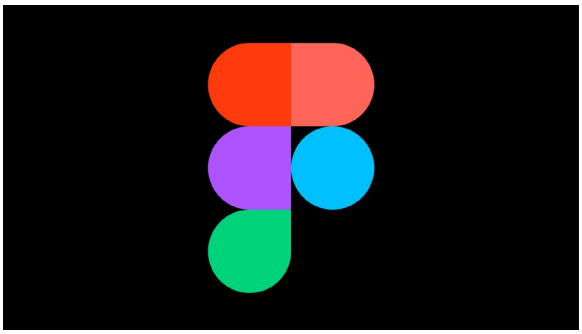

Figure 40 - Figma Logo

Figma is a web-based vector graphic editor and prototyping tool that allows real-time collaboration. It works in diverse environments including browsers, macOS and Windows desktop, and iOS and Android mobile environment. With many features supportive of fast and efficient decision making, this platform allows designers and programmers to easily create and share ideas.

10.2.3. Flutter

Figure 41 - Flutter Logo

Flutter is an open source GUI application framework based on Dart language. Since it yields high-performance in both mobile and smart device environments with one code base, this framework would enable well-organized design and efficient maintenance of frontend environment.

10.3. Backend Environment

10.3.1. MySQL (RDBMS)

Figure 42 - MySQL logo

MySQL is an open-source relational database management system (RDBMS). A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. It uses SQL, which is a language programmers use to create, modify, and extract data from the relational database, as well as control user access to the database.

10.3.2. AWS EC2 (server)

Figure 43 - AWS EC2 logo

It is the center of Amazon's cloud computing platform Amazon Web services, allowing users to rent virtual machines and run their own computer applications on them. EC2 encourages

scalable application deployment by providing a web service that allows users to boot into the Amazon Machine Image (AMI) and configure virtual machines that Amazon calls "instances" with the desired software. EC2 provides us with control over geographic instance locations that allow for latency optimization and high levels of multiplexing.

10.3.3. AWS RDS

Figure 44 - AWS RDS logo

Amazon Relational Database Service is a distributed relational database service by Amazon Web Services (AWS). It is a web service running "in the cloud" designed to simplify the setup, operation, and scaling of a relational database for use in applications. Administration processes like patching the database software, backing up databases and enabling point-in-time recovery are managed automatically. Scaling storage and compute resources can be performed by a single API call to the AWS control plane on-demand.

10.3.4. Spring

Figure 45 - Spring logo

The Spring Framework is an open-source application framework for the Java platform. It provides various services for developing dynamic web sites. It features its own model–view–controller (MVC) web application framework.

10.4. AI System

10.4.1. YOLO v5

Figure 46 - PyTorch logo

PyTorch is an open source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

Figure 47 - YOLO v5 logo

YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images. YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects. This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities and bounding boxes simultaneously.

10.4.2. Tesseract OCR

Figure 48 - Tesseract OCR logo

Tesseract is an open source text recognition (OCR) Engine, available under the Apache 2.0 license. It can be used directly, or (for programmers) using an API to extract printed text from images. It supports a wide variety of languages. Tesseract doesn't have a built-in GUI, but there are several available from the 3rdParty page. Tesseract is compatible with many programming languages and frameworks through wrappers that can be found here. It can be used with the existing layout analysis to recognize text within a large document, or it can be used in conjunction with an external text detector to recognize text from an image of a single text line.

10.5. Constraints

The system is to be implemented and designed in reference to this document. The details not mentioned in this document may be freely decided by the developer of that part, following the rules below

- Protect all the personal information including user account information and preference records. Information use should take place with user agreement.
- Aim to provide utmost user-experience.
- Consider wide range of users such as those with color-blindness or low digital literacy when constructing UI.
- Recipe database must consist of data that are collected under proper copyright ethics.
- Make decisions considering the future scalability and maintenance.
- Optimize the source code to fully utilize the system resources.
- Refactor and comment the source code for other developers.
- Development should be directed to overall system performance improvement.
- Avoid usage of APIs that require external software license or royalty. Use open source software as much as possible.
- Write source code in Windows 10 environment and compile with Android Studio build tool version 30.0.2.
- Target these environments when developing:

  Tizen OS version 6

  Android version 12, minimum 10

  iOS version 14, minimum 13

- Emulate the system using Android version 12 (API 31) and iOS 14.

10.6. Assumptions and Dependencies

All systems in this document are designed and implemented based on smartphones (Android/IOS), smart refrigerator, and open sources. Therefore, all content is based on the smartphone operating system with a minimum API version 23 and smart refrigerator with embedded system. It may not be applicable to other operating systems or versions.

11. Supporting Information

11.1. Software Design Specification

This software design specification was written in accordance with the IEEE Recommendation (IEEE Recommended Practice for Software Design Description, IEEE-Std-1016).