MIAMI

CSC 431

Smart Food Storage

System Architecture Specification (SAS)

Team 04

Hill Yu	Developer
Angela Wang	Scrum Master
Jason Donovan	Developer
Alexander Luce	Developer

Version History

Version	Date	Author(s)	Change Comments
1.0	3/30/2022	Angela Wang	First draft

Table of Contents

1. System Analysis

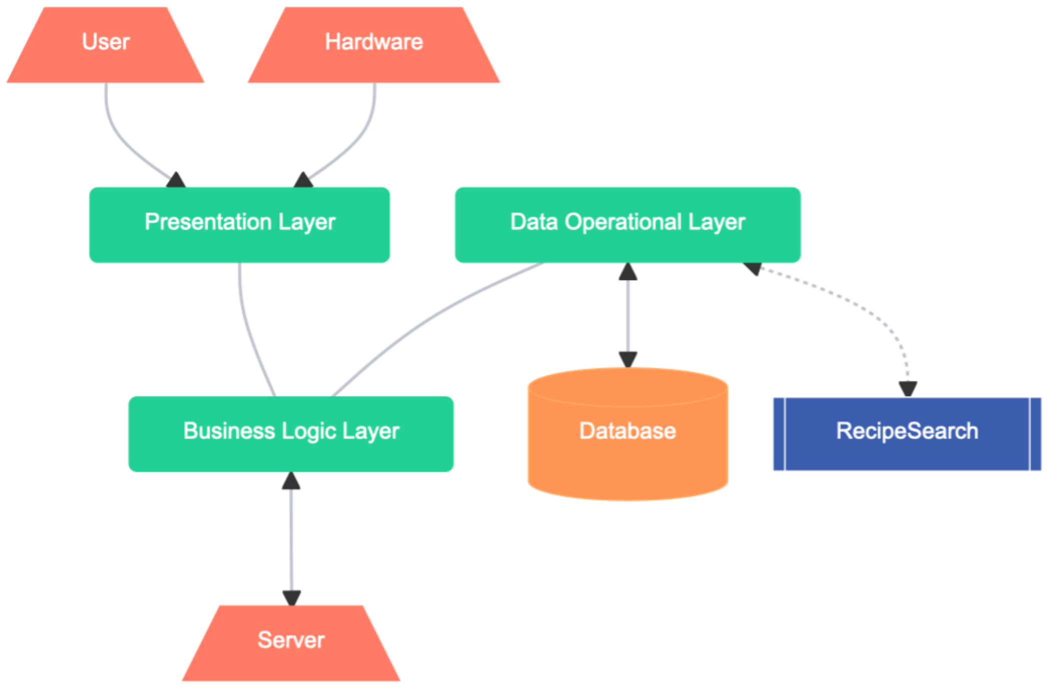
1.1 System Overview

The following document will describe the design and specifications of the Smart Food Storage devices and network. The composition of our system is mainly focused on the following parts: tracking stored food items in the database, and presenting generated recipes based on these items.

The data on food items currently stored inside the Smart Food Storage will be cached locally in the user's application, as well as stored in our database for a backup.

The system is structured as a traditional 3-tier application so the client can directly access the server, we first have the UI written in JavaScript. In the middle tier, information is collected from the user and used to modify data in the final tier, the data tier. In the database tier, recipes can be called upon for the user depending on the food items present.

1.2 System Diagram

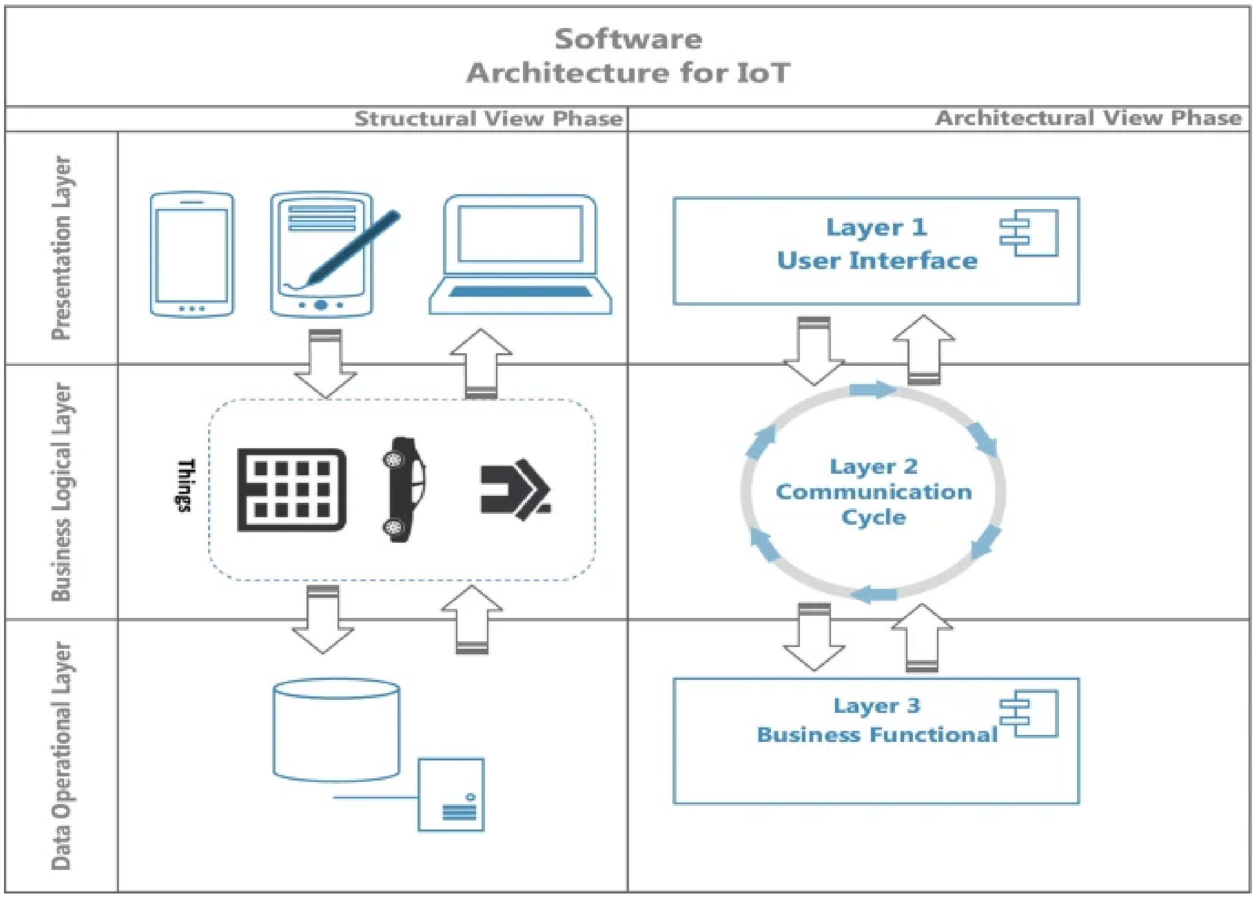


1.3 Actor Identification

There are 4 primary actors within the system: the user, device, server, and database. The user actor takes on the role of an entity that physically interacts with the system. This includes actions such as moving items in and out of the refrigerator, searching for recipes through the system, removing expired foods from the fridge, creating / managing accounts within the system, and many other possible actions. The device actor is one of the mediums between the user and the rest of the system. The device must accept input from the user actor. Any hardware capable of running the platform can be classified as a device actor, such as a laptop or phone. The device actor must also interact with the servers to request recipes, perform authentication, etc. The server actor provides the ‘cloud’ component of the system. It is a centralized computational resource that serves as the gateway to the database. It handles authentication, recipes, software updates to the system, and many other things. The server is responsible for all computation necessary between the device and database system actors. The database actor is the storage component of the system. It maintains a database of food items, recipes stored on the system, as well as user and password information.

1.4 Design Rationale

1.4.1 Architectural Style——-layered architecture style



In accordance with common IoT design patterns, the system uses a three layer architecture: a data operational layer, a business logic layer, and a presentation layer. The presentation layer handles users when they interact with the system. It provides a UI (User Interface), and keeps track of the state of hardware in the refrigerator. Data such as refrigerator temperature, contents, and freezer temperature are served to the user as well as the business / data layers from the presentation layer. The business logic layer coordinates between the data and the presentation layer. This layer prepares data for the presentation layer, and acts as an authenticator for the database system. The data operational layer is necessary to store all data pertaining to users, recipes, software versioning, and persist other relevant information about users. It maintains the integrity of the data it stores and allows CRUD operations to be performed on it from authenticated users.

1.4.2 Design Patterns

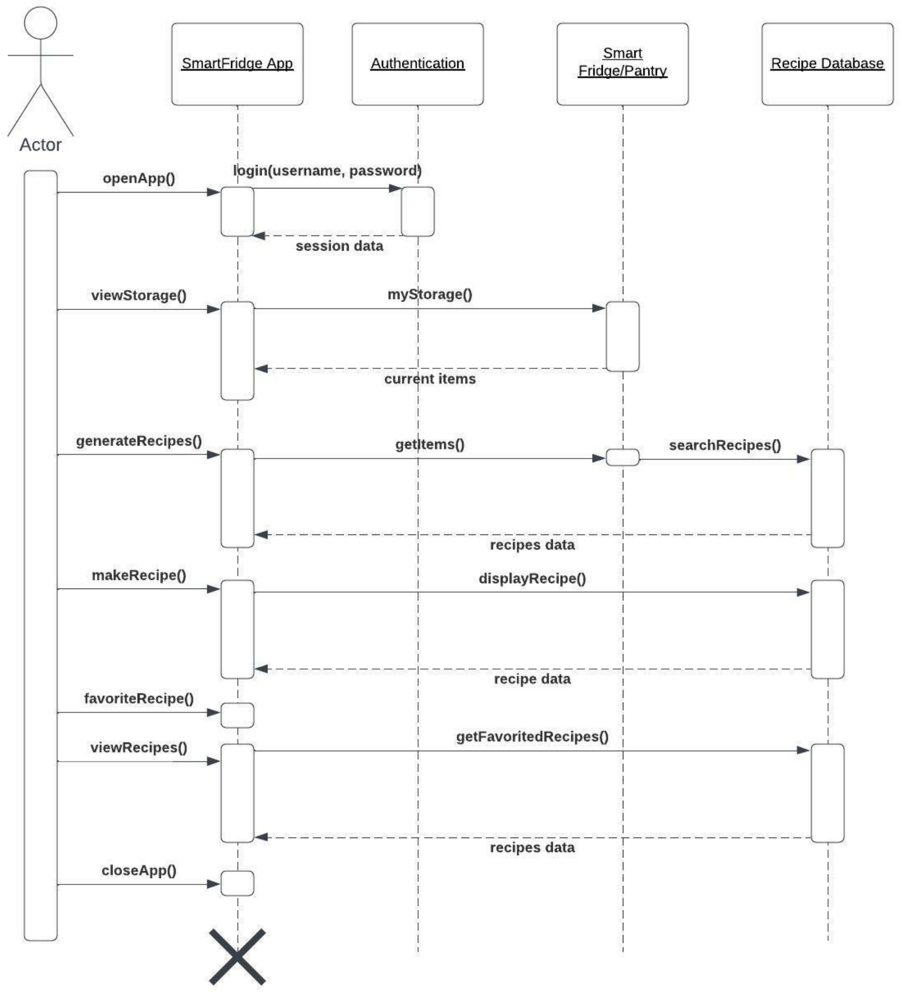
We utilize a ‘Strategy’ design pattern as it fits the needs of the system best. It allows the system to serve a variety of algorithms depending on the use case and other parameters of the system such as network load, search breadth, etc. This design pattern can efficiently add new algorithms in the future, in accordance with planned evolutionary changes to the system. A Navigator will be used as the context class, which will implement the strategy class: RecipeSearch. RecipeSearch includes different algorithms for searching recipes, such as searching for recipes based on the current food items, or searching for recipes based on substitutions.

1.4.3 Framework

Smart Food Storage will utilize the Spring framework, which includes Java, Hibernate, and MySQL. Spring also has the mobile extension, Spring Mobile, that will aid mobile development for the Smart Food Storage app. While Java will be used for the majority of development, the MySQL framework will be used for the storage of multiple recipes. The Hibernate Framework will map Smart Food Storage classes to the recipe database. Using the Hibernate framework will free Smart Food Storage from data persistence problems that may arise from changes in the recipe database, which is based on external APIs (recipe sites).

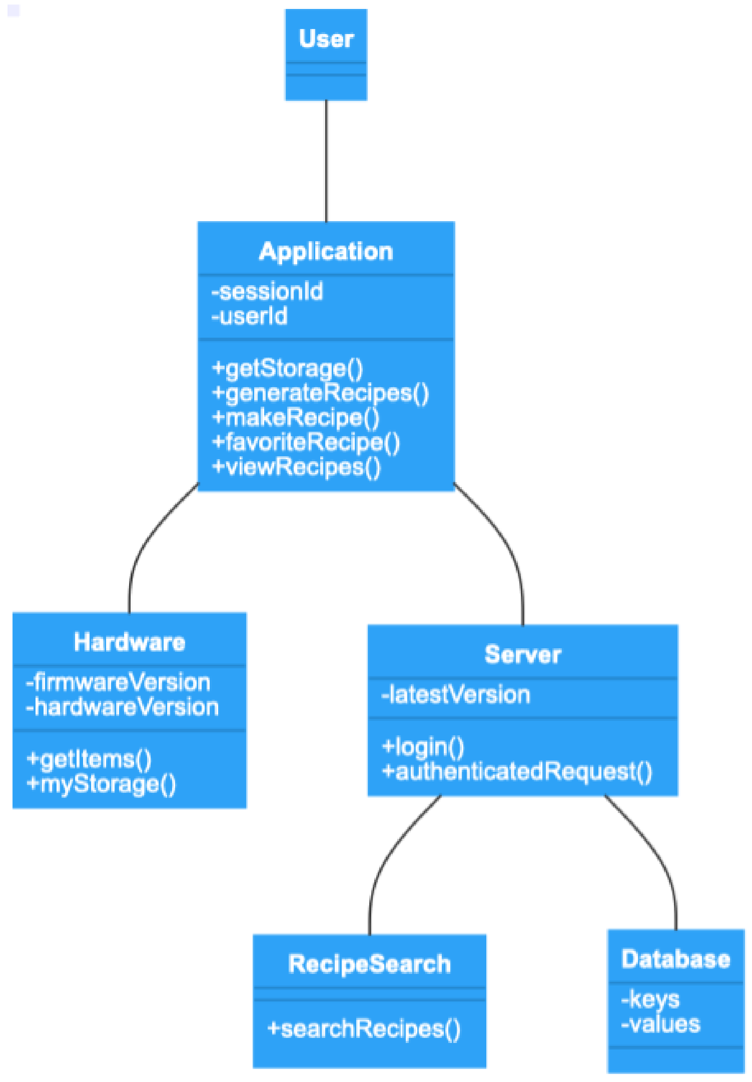
2. Functional Design

2.1 Diagram Title



In this diagram, the user is represented as an Actor. The user interacts solely with the SmartFridge Application. The software interacts with both the Authentication and the hardware Smart Fridge. The Authentication class represents the server along with the user database. It acts as an intermediary for software requests and routes them to the correct location. The recipe database acts as its own class rather than a storage system as it can either be a database or a wrapper around an external API.

3. Structural Design


User
Application
-sessionId
-userId
+getStorage()
+generateRecipes()
+makeRecipe()
+favoriteRecipe()
+viewRecipes()
Hardware
-firmwareVersion
-hardwareVersion
+getItems()
+myStorage()
Server
-latestVersion
+login()
+authenticatedRequest()
RecipeSearch
+searchRecipes()
Database
-keys
-values