Smart Fridge Connected Smart Container

A Revolutionary Smart Container That Connects to Your LG Smart Fridge

Kieran Choo
School of Electrical and Electronic Engineering
Nanyang Technical University
Singapore
Email:KCHOO012@e.ntu.edu.sg

Cyriaque Denniel
ECE Paris École d'Ingénieurs
School of Engineering
Paris, France
Email: cyriaque.denniel@edu.ece.fr

Le Mee Thomas
ECE Paris École d'Ingénieurs
School of Engineering
Paris, France
Email: thomas.lemee@edu.ece.fr

Kim Seung Hyun
Department of Information Systems
Hanyang University
Seoul, South Korea
Email: skim21@hanyang.ac.kr

***Abstract*—This electronic "Smart Container" aims to provide a convenient way to allow users to track and maintain the freshness of the container's contents within a smart fridge. By using a combination of a LAN connection and a built in timer, the Smart Container intends to connect to the Smart Fridge and provide real time information of the duration the contents within have been present for. By constantly communicating with the Smart Fridge through a seamless connection, the Smart Container will relay time and location information of the container, informing the user via Built-in lights and a Notification alert sent to the user's preferred device of the duration the container has been in the fridge for and whether the container has been outside the fridge for a set duration of time. By informing the user of this information, we aim to minimize food spoilage and potential food waste from accumulating within the fridge of the user.**

Roles	**Name**	**Task and Description**
User	**Cyriaque Denniel**	**The role of the user is to test the Smart Home system in real-life situations, identify usability issues, suggest improvements, and help make the system intuitive, reliable, and truly useful.**
Customer	**Thomas Le Mee**	**The role of the customer is to use the Smart Home system, evaluate its usefulness and satisfaction, and provide feedback to help ensure a practical, enjoyable, and reliable experience.**
Software Developer	**Kieran Choo**	**The role of software developer is to design, build, and maintain the Smart Home system's applications and backend. The developer implements features, ensures smooth integration with devices and protocols, and resolves technical issues, contributing to a reliable, efficient, and user-friendly system that meets real-world needs.**
Development manager	**Kim Seung Hyun**	**The role of the development manager is to oversee the creation and delivery of the Smart Home system, coordinating teams, setting priorities, and ensuring technical and functional goals are met. The manager guides the development process, aligns efforts with user needs, and ensures the final product is reliable, efficient, and ready for real-world use.**

I. Motivation

Our motivation for creating the product was to address the lack of fridge management systems currently on the market. Despite the introduction of Smart Fridges to potentially combat this problem, and the use of AI powered cameras within some of the latest generations of Smart Fridges, the reality is fridge management is still done manually by most of the fridge's users, and many are either unaware of, or find it difficult to utilize, the advanced features their Smart Fridges provide. As a result, the problem of unmonitored food storage persists.

This situation often leads to food waste, as leftovers or unused ingredients are placed into containers and subsequently forgotten. Over time, these neglected items spoil unnoticed, contributing not only to unnecessary household waste but also to larger environmental and economic concerns. By integrating a Smart Container system that connects directly to a Smart Fridge, we aim to bridge the gap between technological potential and everyday usability.

Our product is designed to automatically track how long food has been stored, provide notifications when items have been left out of the fridge for too long, and remind users when food is nearing expiration. This system seeks to make fridge management more intuitive and accessible for everyday users, reducing the cognitive burden of manual tracking while promoting sustainability and smarter household habits. Ultimately, our goal is to empower users with a practical and user-friendly tool that minimizes waste, optimizes food freshness, and enhances the overall efficiency of modern kitchen management.

II. Potential Use Cases

A. *In a Home Setting*

The introduction of the Smart Container system provides several benefits within a home setting. By automating the monitoring and tracking of stored food, the system significantly reduces the likelihood of waste caused by forgotten or spoiled items. Users are informed when food has been stored for an extended period or when a container has been left outside the fridge for too long, enabling them to make timely decisions about consumption or disposal. This not only enhances food safety but also encourages more responsible consumption habits within the household.

In addition, the Smart Container system simplifies daily routines by removing the need for manual tracking or labeling of food. Through its integration with Smart Fridge systems and mobile applications, users can easily view the status of their stored items, receive real-time updates, and plan meals more effectively. This creates a more organized and efficient kitchen environment, reducing the mental stress often associated with managing perishable goods.

B. *In a Commercial Setting*

The Smart Container system also offers significant benefits in commercial settings, where efficient food management and safety standards are critical. In environments such as restaurants, cafes, and grocery stores, the system can serve as a valuable tool for monitoring ingredient freshness and ensuring compliance with food safety regulations. By automatically tracking storage durations, and container movement, the system reduces the risk of human error in food handling and inventory management—areas where mistakes can result in financial loss or health risks.

For restaurant staff and store employees, the Smart Container simplifies operations by providing real-time updates on ingredient status through a centralized dashboard. Staff can quickly identify which items are approaching expiration or have not been returned to the fridge, allowing the staff to make decisions to return or dispose of said food items. This not only minimizes waste and prevents spoilage but also supports more accurate stock rotation, ensuring that products are always used in the proper order of freshness.

Furthermore, the integration of the Smart Container system into commercial kitchens enhances transparency and accountability. Detailed data logs can assist with health inspections, internal audits, and sustainability reporting, offering clear evidence of proper food storage practices. By reducing unnecessary waste, improving operational efficiency, and reinforcing food safety, the Smart Container contributes to a more sustainable and cost-effective business model. In this way, it aligns with the growing demand for smarter, technology-driven solutions in the food service and retail industries.

III. Requirements (TBD)

Though incomplete, these are the current requirements our group has determined to be necessary to create this product. The requirements will be split into a physical requirement and software requirement as our current plan is to create a physical device that could be presented to the LG judges.

A. *Physical Requirements*

- Container - A physical container composed of either glass or plastic would be ideal, and we are still discussing the potential viability of a metal or steel container.

- Communication Device - A communication device that could be attached and connected to the container to communicate with the Smart Fridge. Must be small and portable to be usable, and must ideally fit within the lid or within a small portion of the container.

- Onboard Computer - An onboard computer would be necessary to calculate the location of the container, track the time with an onboard timer, and maintain constant connection with the Smart Fridge to send and receive alerts about time and location information.

- Wireless charging Method - A method of wireless charging would be required to use this device. A wireless charging dock that acts as the fridge divider would go in tandem with the wireless charging dock on the bottom of the container, charging the device.

- Small power source - A small power source would be necessary to power the devices while they are not within the fridge to allow for location based information and alerts to reach the user's devices.

B. *Software Requirements*

- LG ThinQ Smart Fridge API Connectivity - A necessary software component to connect our Smart Container to seamlessly integrate with existing LG Smart Fridges and the LG ThinQ App.
- Timer and Location function - Our code would need to be able to track and process both the time, either based on an onboard timer or through constant time requests from the Smart Fridge, and the time required before an alert is sent out should be customizable. The code should also be able to track the container's location and determine whether it is within or outside of the fridge.
- Alert and Notification function - The application must be able to generate an alert and send a notification to the user based on the time and place of the container. The notification should be visible from the LG ThinQ application and from the phone's built in pop-up notification.
- Wireless network or LAN connection - Each box should be able to either connect or disconnect from the fridge whenever necessary, and the smart container should be able to send and generate these alerts through the smart fridge, hence why a wireless connection through either bluetooth or a network or other wireless connection method is necessary.
- UI Integration with ThinQ - The code should also be visible and usable on the LG ThinQ App, and customizable from within the LG ThinQ App. Making the user download a separate app to control the containers would diminish the user-friendliness of our product.
- Customization Options within the App - Each container needs to have the ability to be able to be customizable in terms of name and color of the light to indicate the contents in the box. The alert messages, alert and notification message type, and the time necessary for the notification to be sent out should also all be customizable.

IV. Current Plans to Fulfill the Requirements

A. *Current Plans to Fulfill the Physical Requirement*

- Container - Containers could be bought in bulk through online and offline stores for a low cost. Any food-safe containers that are made of plastic or glass with the measurements of 10cm ~ 15cm in width and length and 5cm ~ 10cm are currently being considered.
- Communications Device and Onboard Computer - Our current plan is to fulfill both of these requirements by using a Raspberry Pi or an Arduino to serve both functions.
- Wireless Charging and Power Storage - We are still in discussion as to how we could fulfill these requirements without diminishing the efficiency of the batteries and these components occupying most of the container's usable space. Current solutions are small capacitors, Lithium-ion Phosphate batteries, or Phase Change Material batteries.

B. *Current Plans to Fulfill the Software Requirement*

- LG ThinQ API - LG has released their ThinQ API to the public to allow developers to utilize their products and create a more well connected Smart Home Environment.
- Timer and Location function - By utilizing a close range connection, we plan to start generating an alert based on if the connection has been lost, though this function still needs consideration.
- UI Integration with the ThinQ App - LG has released their ThinQ App SDK, but more research is needed to determine whether the SDK is compatible with the coding environment we plan to use.
- Alert and Notification system - By utilizing the device's built-in pop-up notification function of both Apple and Android devices, we plan to alert the users wirelessly via the smart fridge.

V. Specifications

A. *Physical Specifications*

- Container - Food-safe plastic must be used with a small sealed compartment in the lid to house the electronic components. A transparent section will be added on the lid as a window for light indication of the status of the container.
- Communication Device/ Onboard Computer - A Raspberry Pi Zero W will be used to utilise Bluetooth Low Energy while making sure the device is compact enough to be fitted onto the container.
- Wireless charging - Implemented using Qi-compatible receiver coil. Brackets will be made to align the container's base embedded with the coil, and the transmitter built into the fridge divider.
- Small Power Source - A 1000 mAh battery will provide power to the container when disconnected from the fridge. A microcontroller will monitor the charge status and will relay the status to the LED (green - fully charged, red - not fully charged) and the fridge (to be sent to the ThinQ App).

B. *Software Specifications*

- LG ThinQ API Connection - The container's microcontroller will communicate with the LG ThinQ API via REST calls using HTTPS and OAuth 2.0 authentication. The firmware will send updates such as connection status and alerts every few seconds. Error-handling routines will retry failed transmissions up to three times before signaling disconnection.

- Timer and Location function - Bluetooth Low Energy (BLE) will indicate the distance of the box from the reference of the fridge. The program will use periodic checks to determine whether the container has been removed or inactive over a set duration. If the threshold is exceeded, an alert will be triggered. This logic will be fed through a loop that compares elapsed time and connection state at fixed intervals. A real-time clock module will handle time tracking, maintaining a set accuracy of 1 second/hour.

- Alert and Notification function - Alerts will be generated by the onboard system and sent to the ThinQ app through the fridge.

    - Alert Message - Each alert will contain the container ID or user designated label, alert type, and timestamp. On the app side, users will receive both a push notification for their mobile device and a visible status update on the app.

    - Alert Notification Customization - Alert notifications will be customizable in the integrated application. Customization options will include changing the alert message and the visible status indicator color. The visible status update will, by default, use colors corresponding to the alert type (i.e. red - immediate actions needed, green - no issues), but can be modified by the user through the app.

    - Alert time customization - Users will be able to customize the duration before the alert notice will be sent out. The default setting will be set to:

        - Container is out of fridge - 1 hour

        - Container is in the fridge - 1 week (caution alert), 2 weeks (danger alert)

      However, the user may either manually adjust the time, or use the integrated AI to automatically set the alert duration. Time will be adjustable by year, week, day, hour, and minutes, and can either be a count-down timer or a set calendar date.

- Wireless network or LAN connection - Bluetooth Low Energy will handle automatic connection between the box and the fridge. A callback function to re-initialise and re-establish encryption of connection will be included. AES-CCM (128-bit) encryption will be used.

- UI integration with ThinQ - With the use of LG ThinQ's SDK, unique data fields (status, alerts) for the container will be integrated directly into the ThinQ App interface. Updates to the UI will be triggered by cloud events from the Smart Fridge API, and any updates made to the containers will be sent through BLE.

    - UI Customization Options - The UI itself will be customizable, with the user being able to change the displayed information using blocks. The planned blocks will include:

        - A summary of how many containers total are connected, and categorized by warning levels (normal, caution, danger)

        - What specific containers need attention

        - Battery lifetime and total uptime of each container

        - Categorization of the container's contents (chosen either by the user or sorted by the app)

    - Container Labeling - Each container will be able to be labeled with up to 20 characters and numbers. The only special characters that can be added will be (. , ? !) to minimize the data management for the database.

    - Container Content Categorization - An option to categorize the contents of the containers will be provided. By default, 5 main categories will be provided (Meat, Vegetables, Fruits, Cooked items, Long-term storage), but up to 20 categories could be added by the user, each with a small icon and a description of up to 20 letters, and with color customization. Users will be able to add individual containers to the categories, or set up a condition for automatic sorting.

- Local and Online Database - The Smart Fridge will communicate and store data onto both a local database and online database so the user can access the information from anywhere, and can write and retain their settings should the user decide to replace their Smart Fridge or move locations.

    - DBMS - The Database Management Software will be in MySQL, as it is both open-source and free to use for any purpose, including commercial purposes.

MySQL also supports backup and restore functions, allowing for a quick reboot and restoration in the event of a loss of communication or blackout.

- ○ Local Database - The Smart Fridge will contain a localized database that it calls to and updates every 10 minutes, which will allow the fridge to maintain data integrity even in the event of a loss of communication with the online database, and restore functions quickly in case of a blackout.

- ○ Online Database - This database will act as an online backup in case the local database is unavailable. This database will be stored on an AWS instance, and will be updated every 4 hours, with at least 2 previous versions of the database available for download in case of file corruption or loss of data.

VI. Implementation

A. Physical Implementation

A 32cm x 23cm x 15cm box was used to construct the product. The following items were used to achieve the usability of the box:

- Micro-controller Unit (MCU): Raspberry Pi 5
- Power: 1000 mAh Power bank
- Camera: Logitech C270 HD

Future hardware optimisation of the MCU may replace the Raspberry Pi 5 with a smaller option, but for prototyping, readily available devices have been picked to use.

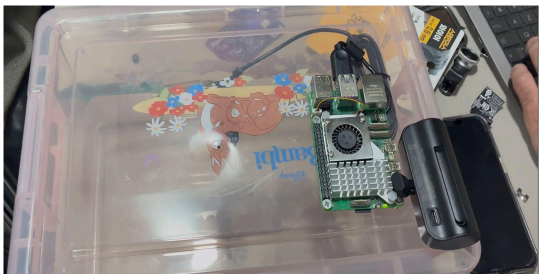


Figure 1. Top lid of the box (MCU + Power bank)

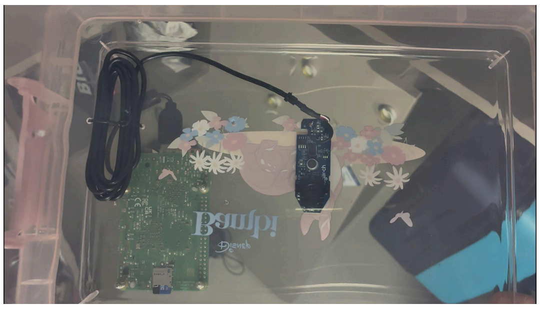


Figure 2. Bottom of the lid (Camera)

The MCU is connected to two items, the Camera and the Power bank. The Power bank provides 4-5 hrs of lifetime before requiring a recharge, allowing ample time for the user to store and maneuver the box. The Camera provides a significant field-of-view (FOV) to capture all the contents of the box to be fed to the model.

B. Software Implementation

Due to the limitations of the LG ThinQ Application, an application was developed to simulate the landing page, item inventory and editing of variables that would come along with this product.

The application can be categorised into 4 features:

- Landing page - Front page where users should be able to easily view the status of boxes, content expiry and container names. A visually ____ should be adopted to ease the use of the application.
- All Items Page - Should show all containers in an orderly manner that reflects the order shown on the landing page. Statuses, names and items should be reflected just like one the landing page.
- Settings Page - Page where users are able to view further settings (e.g. notifications, sound, etc.) and application version. Theming can be added to provide personalisation of the app
- Add Items Page - Users should be allowed to easily adjust the variables that are tied to the individual boxes. Ease of adjustments should also be considered. This will also be the page used for editing items, except for the continents filled up to what it originally was.

The design of the features is heavily considered to allow swift and efficient use of the app. It should also visually appeal to the theme of the current ThinQ app to signify the relation between both. A more in depth look into the features will be done:

Landing page:

- Navigation - Important pages will be put in a menu easily accessible from the front page:
  - ○ Bottom of page - Home, All Items, Settings
  - ○ Top of page - Add Items

  The addition of a swipe features enable navigation through the items at the bottom of the page.

- Display of Containers - Two features will be employed for this:
  - ○ Fridge-like diagram - 3x2 rectangular grid to mimic the fridge where users will use the boxes. The containers will be added left to right, top to bottom and they will be sorted by date. In the case of >6 containers being added, the diagram will be scrollable to access and view any container required.

- ○ Expiring Soon list - Scrollable list which is sorted by the containers of food that are expiring soon showing up first. The size should mimic the size of the containers in the Fridge grid.

The containers will also employ the use of highlights (outline colours) to signify which containers contain expired (red, <0 days left), expiring soon (amber, <2 days left), normal (default grey, >2 days left).

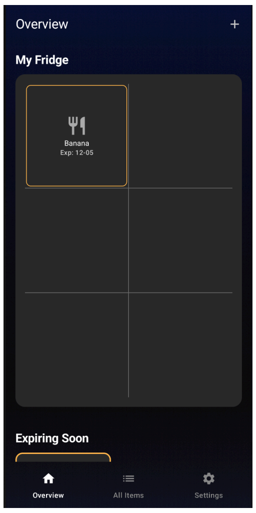


Figure 3. Overview (Landing) page for the application

All Items Page:

- Layout - Items should be listed at a reasonable size to compact all details and show at least 10 items. Items will be arranged based on the same specifications as the Expiring Soon list.
- Design - Containers will be colour coded based on the conditions mentioned for highlights. Expiry date will also mimic this colour scheme at a brighter shade to signify importance.

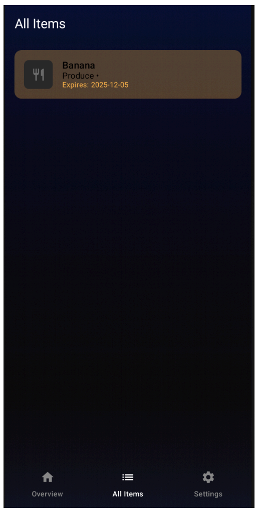


Figure 4, All Items page

Settings Page:

- Notifications - Allow the versatility of turning on/off the notifications of the application. Users can select the day till expiry to receive the notification
- Theme - Users are able to customise and personalise the app by allowing the selection of different themes.
- Version details - Provides information to the user of which build the app is in.

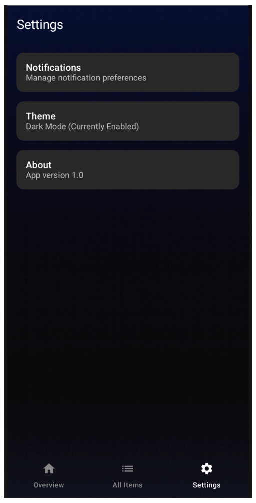


Figure 5. Settings page

Add Items Page:

- Variables - Include multiple variables to not only name, but label and assign food types for the food in the container.
- Calendar - Allow the adjustment of dates to suit the needs of the user. A calendar will be provided to enhance the ease of adjustments with a point and press on the date they will want to adjust to.
- Photo - An option to retake the photo for display in the app is provided to users.
- Edit Items - The edit items page will have a replicated Add Items page with the original components filled up. Users are then able to adjust the values.

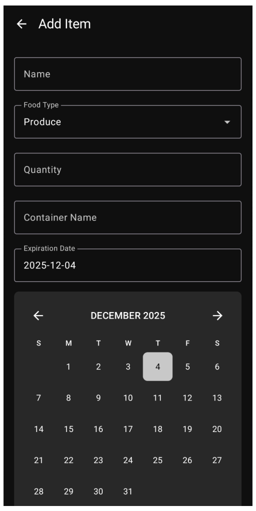


Figure 6. Add Items page

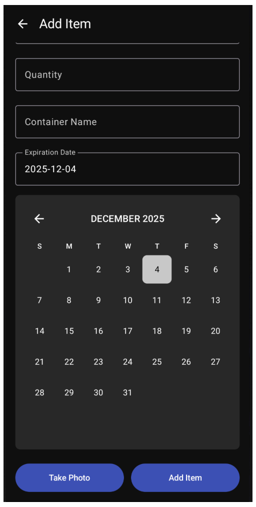


Figure 7. Calendar and add photo

It should be added that all objects in the application will follow the design philosophy of the theme (e.g. same colour scheme, rounded rectangular boxes). Every container displayed will include the original image that was taken for the processing (can be changed by the user).

AWS Integration

Our application utilizes an AWS server as the main controller. Our AWS is structured as follows:

- An EC2 Instance running Ubuntu will be our main control hub, managing both the AI model and traffic flow from both FAST APIs and data transfers between the different AWS services. The instance will receive HTTP requests through the FAST APIs and pass them onto the MySQL RDS, which then executes the appropriate actions and sends them back to the EC2.
- A MySQL RDS Instance will manage the databases responsible for keeping track of the different settings from the app. Each new app gets a new table with the following categories:
    - Container Name
    - Container Location in the Fridge
    - If the Container is Connected
    - Constant connection check
    - Date the food item was added
    - Date the food item is set to expire

  While the HTTP port 80 is open and available, we have set it so that the RDS Instance would refuse any other HTTP connections that are not the designated EC2 instance.
- Lastly, the S3 bucket will serve as both the storage for the AI model's dataset, and the images the EC2 will receive from both the box and the devices using the application. Once the image has been received, the S3 bucket will store it in one of 2 folders; Food-Analysis or App-Icon-UserNºX (X representing the user number). Once stored, the S3 bucket will then send the URL of the stored image back to the EC2, which will then either send it to the AI model for analysis, or store it on the MySQL DB if it is used for the icon.

The determination of the shelf life of the identified food will be done through a Machine Learning (ML) model with a data set:

- Dataset - A dataset with the inclusions of images of food with the shelf lives provided in the folder names (e.g. ~/Downloads/Dataset/Apple (1-3)). The provided fruits are Apple, Banana, Carrot, Tomato and expired variations of these fruits.
- ML - The ML model will handle two components, feature extraction and predicting the shelf life. The feature extraction is done with the use of ViT for feature extraction. The model will be trained with XGboost, providing a good balance between accuracy and low error rate. (High percentages provided by other models are rejected in fear of overfitting while low percentages are rejected)

The workflow goes as follows:

1. Box is opened, food item is placed inside and subsequently closed.
2. Box is plugged in the fridge, triggering the MCU to trigger the camera to take a picture. This function is delayed slightly to ensure that the lid is fully closed and stable.
3. Image is sent and stored in the AWS bucket through API.
4. Image will be put through the trained model, predicting the shelf life of the food.
5. Variables (shelf life, image, name) will be sent to the application and stored locally.
6. Application displays the image, name and calculates the date of expiry, displaying them on the respective locations

The add items workflow goes as follows:

1. Select "+" on the Overview page
2. Fill in details and select an expiration date
    a. Picture - If the user requires a change of the image, Select "Take Photo"
    b. Values/Names - User can change directly in the occupied boxes
    c. Date - User can select a date from the calendar to change the expiration date to how they see fit
3. Select "Add Item"

The editing workflow goes as follows:

1. Select item from any menu
2. Select "Edit Item"
3. Menu screen will pop up with different boxes to mark out variables to change

a. Picture - If the user requires a change of the image, Select "Take Photo"
b. Values/Names - User can change directly in the occupied boxes
c. Date - User can select a date from the calendar to change the expiration date to how they see fit

4. After made changes, select "Edit item" - items will now show up as their edited form in the application

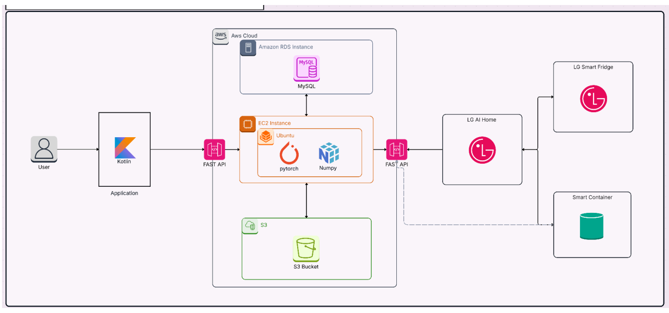
The deleting workflow goes as follows:

1. Select item from any menu
2. Select "Delete Item"

With the separate saving of the local data and the saving of cloud data, a check between the two data sets will have to be done. This is implemented during the launch of the app/app resume, when the local data changes (edit items), and on network connection.

V. Architecture

A. Architecture Diagram



B. Directory Organisation

Table I: Front end

Directory	Key Files	Module	Description
`~/app/src/main/kotlin/foodinventory/ui/screens`	`OverviewScreen.kt` `AddItemScreen.kt` `AllItemsScreen.kt` `FoodDetailsScreen.kt` `NotificationsScreen.kt` `OverviewScreen.kt` `SettingsScreen.kt`	UI	Landing pages for the different menus
App/Androidv1.1/app/src/main/kotlin/foodinventory	MainActivity.kt	System	Hosts NavHost and sets the theme
App/Androidv1.1/app/src/main/kotlin/foodinventory/ui/screens	MainScreen.kt	System	Holds the scaffold for the top and bottom menu bars
App/Androidv1.1/app/src/main/kotlin/foodinventory/viewmodel	FoodInventoryViewModel.kt	UI	Communicates with repository and list items as a state for the UI to be observable and reactable
App/Androidv1.1/app/src/main/kotlin/foodinventory/data	FoodItem.kt FoodInventoryDatabase.kt FoodItemDao.kt	Data handling	Handles the storage of data locally and sends insert, delete and retrieve queries from the cloud
App/Androidv1.1/app/src/main/kotlin/foodinventory/data	Converters.kt	Data conversion	Contains type converters to convert complex types into a format that can be stored

VI. Conclusion & Future Improvements

The Smart Container system successfully demonstrates a modern, practical, and user-oriented approach to food management by integrating hardware sensing, onboard computation, and a companion mobile application. Through the combination of a Raspberry Pi–based prototype, BLE connectivity, a custom Android interface, and a cloud-backed MySQL database, the system provides timely alerts, intuitive organization, and clear visibility of stored items. While the current prototype focuses on validating core functionality, it lays a strong foundation for future enhancements such as improved microcontroller efficiency, deeper ThinQ integration, and more advanced AI-based expiry prediction. Ultimately, this project highlights a viable and scalable solution that complements LG's Smart Home ecosystem by promoting convenience, reducing food waste, and improving everyday user experience.

For future improvements:

- Wireless charging - Future versions can integrate Qi-based wireless charging so the container recharges automatically when placed in the fridge.
- In fridge detection - Improved sensing methods like stronger BLE profiling or NFC/RFID can enhance accuracy in detecting the container's exact fridge location.
- Upgrade of box - The container can be redesigned with a slimmer, more durable build and a more compact electronics housing for better practicality.