IoT Grocery Expiry Management System

To tackle Key challenges in managing restaurant inventory

MIS 6308 – SAPM –Fall 2024

Professor Srinivasan Raghunathan

Group 4 –

Table of Contents

Executive Summary

The proposed **IoT Grocery Expiry Management System (GEMS)** is designed to tackle key challenges in managing restaurant inventory. Traditional manual methods often lead to food wastage, inefficient inventory tracking, and higher operational costs due to expired or poorly stored items. By leveraging IoT technologies, GEMS offers a smarter, more sustainable approach.

Key Features and Benefits:

1. **Real-Time Inventory Tracking**:
   - Smart shelves equipped with weight sensors detect product quantities automatically.
   - Temperature sensors monitor storage conditions, recalculating expiry dates if thresholds are breached.
2. **Automated Expiry Alerts**:
   - The system identifies items nearing expiration (within 7 days) and sends timely alerts to chefs.
   - Alerts are displayed in the system and sent via email, enabling immediate action to prevent waste.
3. **Menu Adjustment**:
   - Chefs can adjust menus based on real-time inventory, prioritizing ingredients nearing expiry.
4. **Simplified Access and Usability**:
   - Secure access through facial recognition for authorized users.
   - User-friendly interfaces for inventory viewing, alert management, and menu updates.
5. **Environmental and Cost Impact**:
   - Reduces food waste, lowers disposal costs, and supports sustainability initiatives.

How It Works:

- Sensors continuously monitor product weight and temperature, updating the system dynamically.
- Alerts are generated for nearing expiry items.
- Chefs and staff can use the system to view inventory, respond to alerts, and adjust menus accordingly.

Implementation: The system integrates seamlessly with existing infrastructure using RFID/barcode tagging, ensuring efficient tracking and scalability. It can accommodate up to 5,000 products and is designed to scale with the restaurant’s growth.

In summary, GEMS enhances operational efficiency, reduces waste, and aligns with modern sustainability goals, providing a smart solution for managing perishable inventory.

Problem Statement

Problem

Managing a large volume of perishable products with various expiry dates is a very difficult process in restaurants. Manual traditional methods often cause errors and a lot of food waste because items are not used before expiration. This has negative effects on efficiency of operations, and therefore the operational costs and environmental waste are increased.

We also have no way of letting chefs know when the items will expire, so that they can work out what they will serve on the menu. The second issue is that there's no automated monitoring, even when it comes to storage conditions like temperature fluctuations that can quickly speed up spoilage and render expiration dates worthless. Such issues go unnoticed by restaurant staff, resulting in wastage and inefficiency until it's too late.

These challenges can be resolved with IoT technologies. With such a system we are able to track inventory in real time, monitor storage conditions, and send out alerts when products are about to expire. This would save restaurants from wastage, help them use their inventory better and reduce their carbon footprint.

Objectives

- **Reduce Food Waste:** Minimize wastage by ensuring timely usage of perishable items through effective tracking and prioritization.
- **Automate Inventory Management:** Eliminate errors caused by manual methods by implementing a real-time IoT-based inventory tracking system.
- **Monitor Storage Conditions:** Account for environmental factors like temperature to prevent premature spoilage and maintain food quality.
- **Provide Timely Alerts:** Notify chefs about items nearing expiration to enable informed decisions for menu planning.
- **Optimize Resource Utilization:** Streamline inventory usage to reduce costs and environmental impact by minimizing food waste.

Scope

1. **Smart Shelves Integration:** Smart shelves with weight / temperature sensors, are used to monitor storage levels and conditions.
2. **RFID and Barcode Tracking:** Attach RFID tags or barcodes in order to detect inventory movements and automatically update the expiration information.
3. **Alert System Development:** Set up an automated system that will send out an alert if items are about to expire.
4. **Dynamic Inventory Updates:** Support real time updating of expiration dates and stock levels in the database, based on item movements.

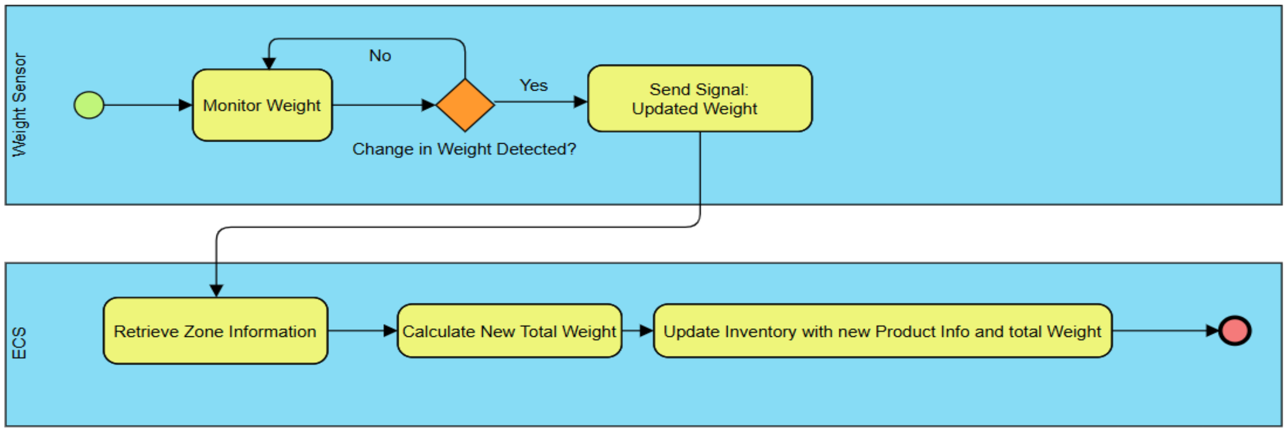
5. **Chef Decision Support:** Help chefs by sending them an alert that will contain information about the products that are about to expire.
6. **Database Foundation:** The new IoT based system will be supported with a preloaded database of expiration data.

Business Process Model Notation

The following are the key business process models for our IoT Grocery Expiry Management System (GEMS):

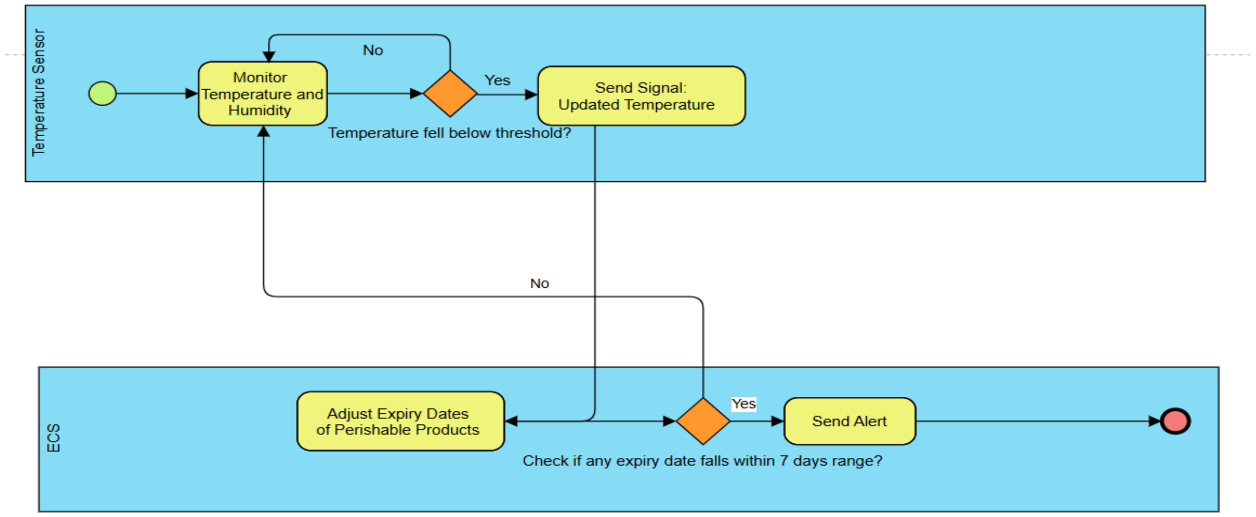
Inventory Tracking using Weighted Sensors

Using weight sensors for inventory tracking entails smart shelves with sensors that track changes in a product's weight. The weight sensor detects when a product is added or removed from the shelf, and signals the system. With the data from the sensors, this is then used, in conjunction with RFID or barcode information to identify the zone and the specific product. It calculates the new total weight to detect quantities added or removed and updates the inventory accordingly. This process ensures real-time, accurate tracking of product quantities, helping to maintain efficient stock levels and reduce waste. **Note:** ECS (Edge Computing System) is a key component of the grocery expiry management system, responsible for processing data locally at the edge of the network. This allows for faster decision-making, such as recalculating expiry dates and updating inventory, without relying on centralized cloud processing.


Weight Sensor
Monitor Weight
No
Yes
Change in Weight Detected?
Send Signal: Updated Weight
ECS
Retrieve Zone Information
Calculate New Total Weight
Update Inventory with new Product Info and total Weight

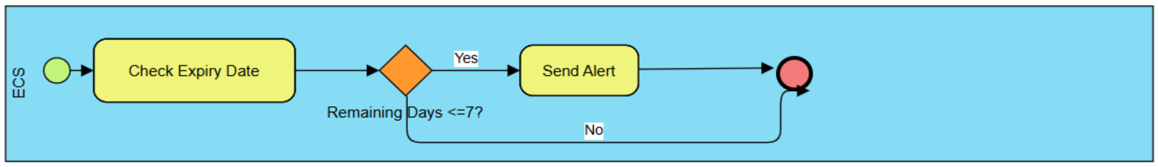
Inventory Tracking using Temperature Sensors

Inventory tracking using temperature sensors involves the installation of sensors in refrigerated areas to monitor and detect changes in temperature or humidity. When the temperature falls below a predefined threshold, the sensor sends a signal to the system, indicating a potential issue with the storage conditions. The system then recalculates the expiry dates of perishable products based on the reported temperature change, updating the inventory accordingly. The system then checks if any items fall within a 7-day expiry range with the new dates and sends an alert to the chef if they do. This ensures that products are stored under optimal conditions, helps prevent spoilage, and provides real-time alerts for items nearing expiry.


Temperature Sensor
Monitor Temperature and Humidity
No
Yes
Temperature fell below threshold?
Send Signal: Updated Temperature
No
ECS
Adjust Expiry Dates of Perishable Products
Yes
Send Alert
Check if any expiry date falls within 7 days range?

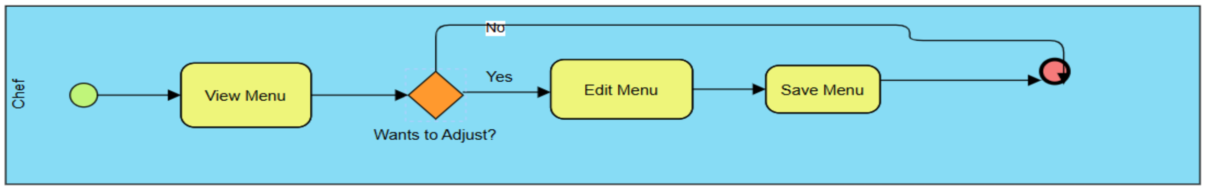
Expiry Date Checking

This is an automated process that runs daily at midnight. The system checks all products' expiry dates in the inventory to see if any items are within a 7-day expiry range. If any items are found to be approaching their expiry date, an alert is triggered. This helps the restaurant stay informed about products that are nearing expiration, enabling timely actions to prevent waste and improve inventory management.



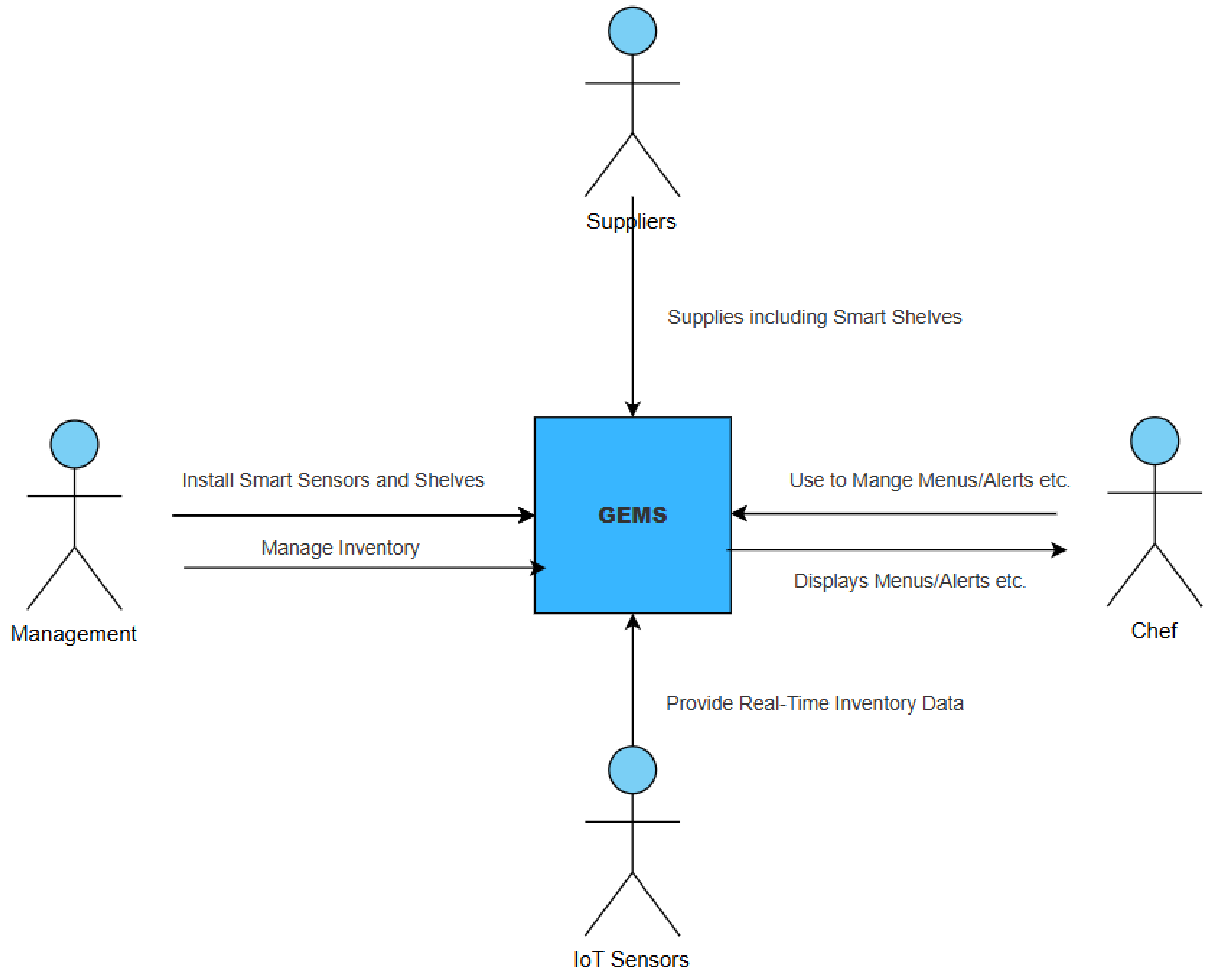
Menu Adjustment

In this process, the chef can simply view the menu and edit it if he wants. He then saves the menu, and it is updated.



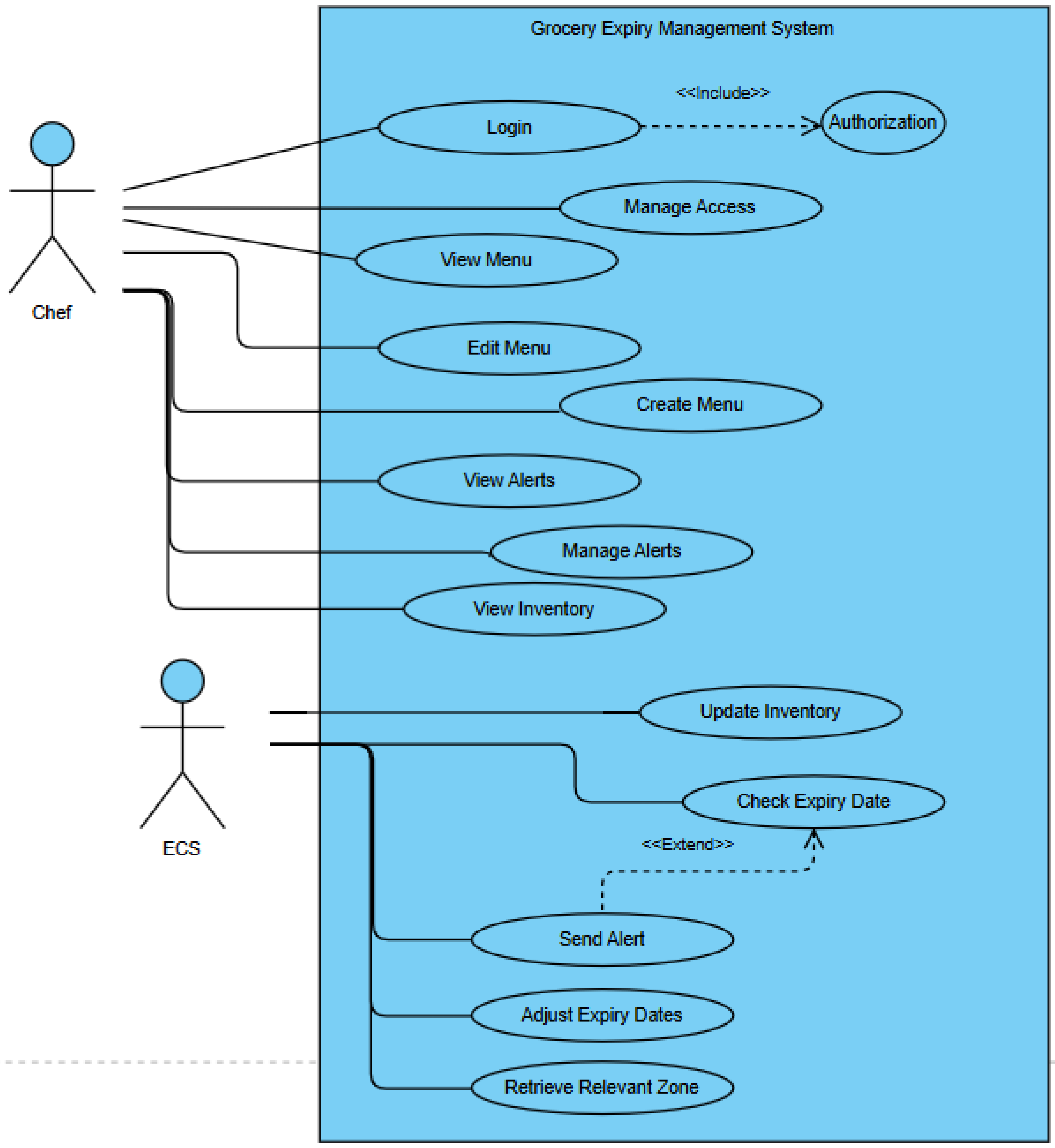
Context Diagram

The context diagram for the GEMS places the system at the center, with several key external entities interacting directly and indirectly with it. **Management** does not interact directly with the system but is responsible for the installation of **smart shelves** and **sensors**. **Suppliers** provide the products to the system but do not have direct interaction. **IoT sensors** (temperature and weight) send real-time data to the system, helping monitor the inventory conditions. The system then updates its inventory based on this data. The **Chef** interacts bidirectionally with the system, managing **menus**, responding to **alerts**, and viewing the **inventory etc**. The system in turn displays **menus** and **alerts etc.** to the chef.


Suppliers
Supplies including Smart Shelves
Management
Install Smart Sensors and Shelves
Manage Inventory
GEMS
Use to Mange Menus/Alerts etc.
Displays Menus/Alerts etc.
Chef
Provide Real-Time Inventory Data
IoT Sensors

Process Model

Use-Case Diagram



Use-Case Descriptions

Use Case: Edit Menu

Field	Description
Use Case Name	Edit Menu
Actors	Chef
Brief Description	The chef edits the menu.
Preconditions	- The chef must be logged into the system with appropriate permissions. - An existing menu must be available for editing.
Postconditions	- The updated menu is saved in the system. - Changes are reflected for future reference.
Normal Flow of Events	1. The chef logs into the system. 2. The chef navigates to the "Menus" section. 3. The system displays the current menus. 4. The chef selects a menu to edit. 5. The chef clicks on the Edit Menu button and changes any content that he wants. 6. The chef confirms the changes. 7. The system updates the menu. 8. The system notifies the chef that the menu has been successfully updated.
Exception Flow	**4a.** If the menu is not available: - The system displays an error message. - The use case ends. **7a.** If the system fails to save changes: - The system displays an error message.

Use Case: Check Expiry Dates

Field	Description
Use Case Name	Check Expiry Date
Actors	ECS, Chef
Brief Description	ECS checks expiry dates of items, either at midnight daily, or based on a trigger by temperature change.

Preconditions	- Inventory data must be up to date. - Items expiring in the next 7 days must be identified.
Postconditions	- Alerts are sent to the chef via email or displayed in the system.
Normal Flow of Events	1. The system checks the inventory at midnight or based on the temperature change trigger for items expiring within the next 7 days. 2. The system identifies items meeting the criteria. 3. The system generates an alert including information regarding each identified item. 4. The system sends the alert via email to the chef as well as displays it on the system. 5. The chef receives the alert.
Exception Flow	**3a.** If no items are expiring: - The system generates no alert. - The use case ends. **4a.** If email delivery fails: - The system retries sending the email. - If retries fail, the system logs the failure and raises a notification for system administrators.

Use Case: Retrieve Relevant Zone

Field	Description
Use Case Name	Retrieve Relevant Zone
Actors	ECS
Brief Description	ECS acts upon the weight sensors' signal and update the quantity(s) of the product(s).
Preconditions	- Weight sensors must be installed and operational. - Zones must be defined within the smart shelves.
Postconditions	- Inventory is updated with the adjusted quantity based on weight changes.
Normal Flow of Events	1. The weight sensor detects a change in weight. 2. The sensor sends the detected weight change to the system. 3. The system identifies the relevant zone from the sensor data. 4. The system uses RFID or barcode data to identify the product associated with the weight change.

	5. The system calculates the updated <u>total weight</u> and determines the <u>quantity</u> of the product added or removed. 6. The inventory is updated with the new <u>quantity</u>. 7. The system <u>logs</u> the update.
Exception Flow	**2a.** If the sensor data is incomplete: - The system requests additional data from the sensor. - If no valid data is received, the system logs an error and ends the process. **5a.** If product identification fails: - The system logs the failure and raises a notification for manual intervention.

Use Case: Adjust Expiry Dates

Field	Description
Use Case Name	Adjust Expiry Dates
Actors	ECS, Chef
Brief Description	ECS acts upon the temperature sensors' signal and adjust expiry dates of all products.
Preconditions	- Temperature sensors must be installed and operational. - Temperature thresholds must be predefined in the system.
Postconditions	- Expiry dates are recalculated, and inventory is updated accordingly. - Alerts are sent if any items fall within the next 7 days of expiry after recalculation.
Normal Flow of Events	1. The temperature sensor detects a change in <u>temperature or humidity</u>. 2. The sensor sends the detected <u>change</u> to the system. 3. The system verifies if the reported <u>value</u> falls below the <u>predefined threshold</u>. 4. If the <u>threshold</u> is breached the system recalculates <u>expiry dates</u> for all affected perishable products. 5. The updated <u>expiry dates</u> are reflected in the inventory. 6. The system checks if any items fall within the next <u>7 days</u> based on the new expiry dates. 7. If any such items are found, the system triggers an <u>alert</u> for the chef.
Exception Flow	**2a.** If the sensor data is incomplete: - The system logs the issue and requests a retransmission from the sensor. - If no valid data is received, the process ends.

	4a. If recalculating expiry dates fails: - The system logs the failure and raises a notification for manual intervention.

Use Case: Manage Alerts

Field	Description
Use Case Name	Manage Alerts
Actors	Chef
Brief Description	Chef manages the alerts.
Preconditions	- The chef must be logged into the system.
Postconditions	Alerts are updated, and irrelevant or resolved alerts are removed.
Normal Flow of Events	1. The chef logs into the system. 2. The chef navigates to the "Alerts" section. 3. The chef views a list of all active alerts. 4. The chef selects specific alerts to delete or mark as resolved. 5. The system updates the alerts database to reflect the changes.
Exception Flow	**5a.** If the system cannot update the alerts database, an error message is displayed, and changes are not saved.

Use Case: Manage Access

Field	Description
Use Case Name	Manage Access
Brief Description	The chef manages users' access.
Actors	Chef

Preconditions	- The user must have administrative rights. - Facial recognition grants system access.
Postconditions	The system updates the access permissions in its database.
Normal Flow of Events	1. The user logs into the system via facial recognition. 2. The user navigates to the "Users" section. 3. The user adds or removes personnel details. 4. The system updates the access database and confirms the changes.
Exception Flow	**4s.** If the system fails to update the database, an error message is displayed, and changes are not saved.

Use Case: View Inventory

Field	Description
Use Case Name	View Inventory
Actors	Chef
Brief Description	The chef views inventory.
Preconditions	- The chef must be logged into the system.
Postconditions	The system displays up-to-date inventory information.
Normal Flow of Events	1. The chef logs into the system. 2. The chef navigates to the "View Inventory" section. 3. The system retrieves and displays the inventory data, including products' quantities and expiry dates.
Exception Flow	**3a.** If inventory data cannot be retrieved, an error message is displayed, and the chef can retry.

Data Dictionary

Use Case: Edit Menu

- Menu Data = Menu ID + Menu Description + Menu Creation Date + Menu Last Updated Date
- Menu ID = Unique identifier for menu
- Menu Description = List of dishes

- Menu Creation Date = Date (Format: YYYY-MM-DD)
- Menu Last Updated Date = Date (Format: YYYY-MM-DD)

Use Case: Check Expiry Date

- Inventory Data = Product ID + Product Name + Quantity + Expiry Date
- Expiry Date = Date (Format: YYYY-MM-DD)
- Alert = Alert ID + Product ID + Expiry Date + Alert Description + Alert Timestamp
- Alert ID = Unique identifier for alerts
- Alert Description = Descriptive message for the alert (e.g., "Item nearing expiry")
- Alert Timestamp = DateTime (Format: YYYY-MM-DD HH:MM:SS)

Use Case: Retrieve Relevant Zone

- Weight Data = Zone ID + Weight Value + Timestamp
- Zone ID = Identifier for smart shelf zones
- Weight Value = Float (Measured in grams/kilograms)
- Timestamp = DateTime (Format: YYYY-MM-DD HH:MM:SS)

Use Case: Adjust Expiry Dates

- Temperature Data = Sensor ID + Temperature Value + Timestamp
- Sensor ID = Unique identifier for the temperature sensor
- Temperature Value = Float (Measured in °C, Range: -50°C to 50°C)
- Timestamp = DateTime (Format: YYYY-MM-DD HH:MM:SS)
- Threshold Value = Float (Predefined temperature threshold in °C for alerts)

Use Case: Manage Alerts

- Alert = Alert ID + Product ID + Expiry Date + Alert Description + Alert Timestamp
- Alert ID = Unique identifier for alerts
- Alert Description = Descriptive message for the alert (e.g., "Item nearing expiry")
- Alert Timestamp = DateTime (Format: YYYY-MM-DD HH:MM:SS)

Use Case: Manage Access

- Access Data = User ID + User Role + Authentication Method + Permission Level
- User ID = Unique identifier for the user
- User Role = Role of the user (e.g., Chef, Admin)
- Authentication Method = String (e.g., "Facial Recognition", "Password")

- Permission Level = Access rights assigned to the user (e.g., "View Only", "Admin")

Use Case: View Inventory

- **Inventory Data** = Product ID + Product Name + Quantity + Expiry Date + Zone ID
- **Product ID** = Unique identifier for products in the inventory
- **Product Name** = String
- **Quantity** = Integer
- **Zone ID** = Identifier for storage zone

Note: All elements are data elements.

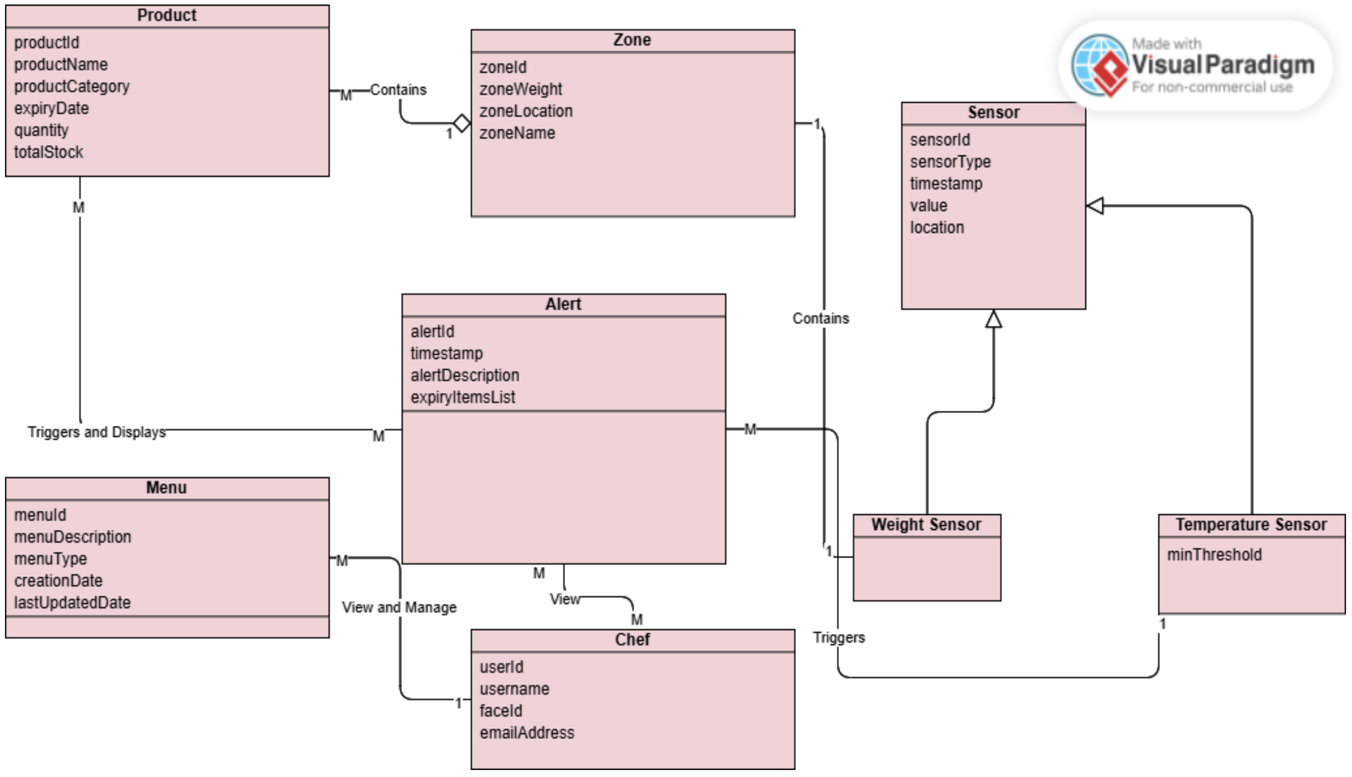
Menu Data = Data Element	Menu ID = Data Element
Menu Description = Data Element	Menu Creation Date = Data Element
Menu Last Updated Date = Data Element	Inventory Data = Data Element
Expiry Date = Data Element	Alert = Data Element
Alert ID = Data Element	Alert Description = Data Element
Alert Timestamp = Data Element	Weight Data = Data Element
Zone ID = Data Element	Weight Value = Data Element
Timestamp = Data Element	Temperature Data = Data Element
Sensor ID = Data Element	Temperature Value = Data Element
Timestamp = Data Element	Threshold Value = Data Element
Alert = Data Element	Alert ID = Data Element
Alert Description = Data Element	Alert Timestamp = Data Element
Access Data = Data Element	User ID = Data Element
User Role = Data Element	Authentication Method = Data Element
Permission Level = Data Element	Inventory Data = Data Element
Product ID = Data Element	Product Name = Data Element
Quantity = Data Element	Zone ID = Data Element

Class Diagram without Methods

The data model for GEMS is shown below. We can see that there are multiple defined relationships between different entities. Let's look at them one by one:

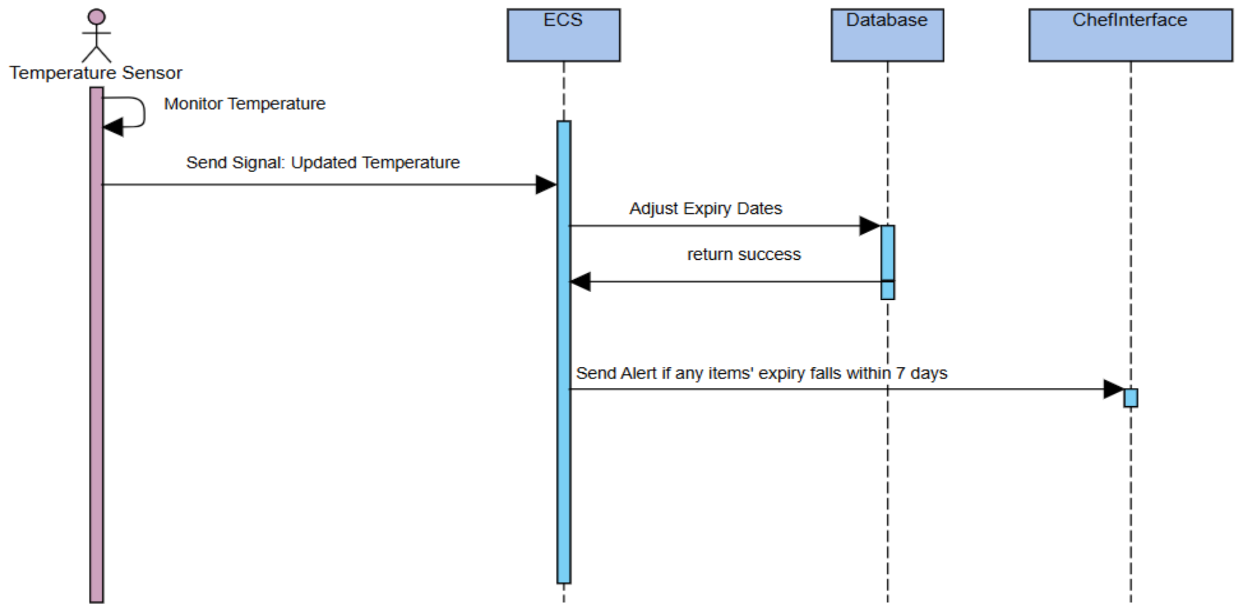
- **Zone and Product:** Each zone **contains** many products, but a particular product can only be placed in one zone.
- **Zone and Weight Sensor:** Each zone **contains** a weight sensor.
- **Temperature Sensor and Alert:** Temperature sensor **triggers** an alert. Each temperature sensor can trigger many alerts, but a particular alert can only be triggered by a specific temperature sensor.
- **Chef and Alert:** A chef can **view** alerts. A chef can view many alerts, and an alert can be viewed by many authorized chefs.

- **Product and Alert:** Checking of expiry date of products can **trigger** alerts. Moreover, an alert can **display** many products. A single product can trigger multiple alerts, as an alert will be displayed every day within the 7-day range leading up to its expiration as well as post expiration, if no attention is given to it.
- **Chef and Menu:** A chef can **view and manage** menus. A chef can manage multiple menus, but a specific menu can only be managed by its owner.

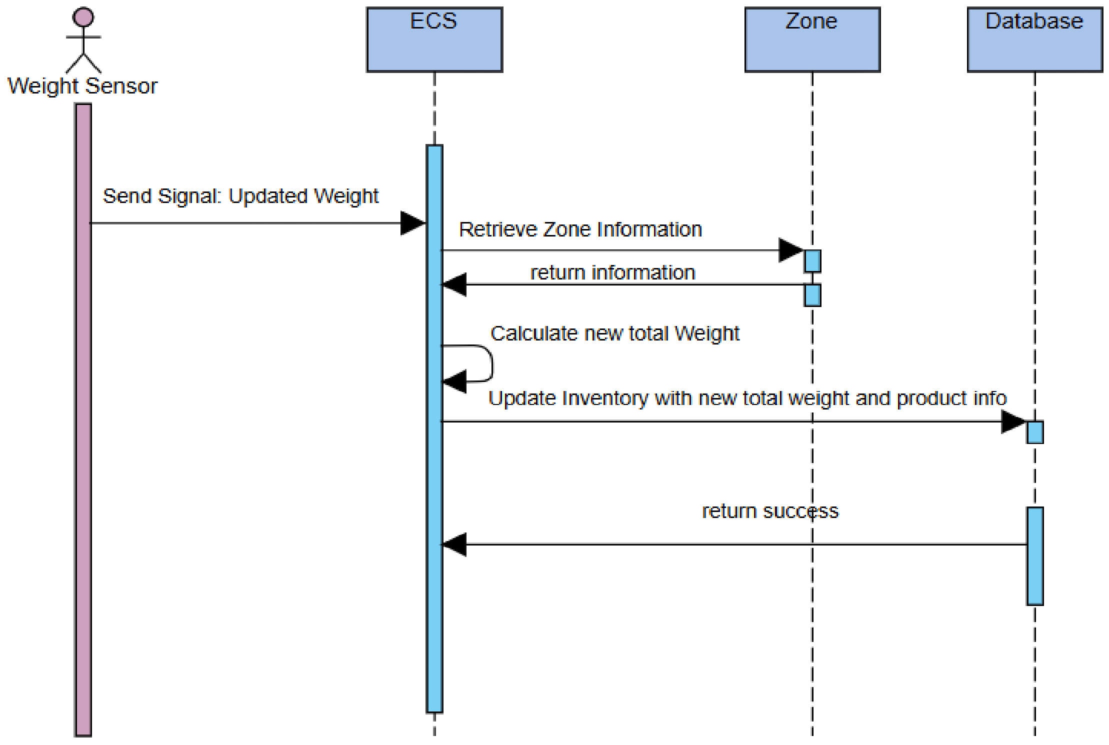


Sequence Diagram

Inventory Tracking using Temperature Sensor

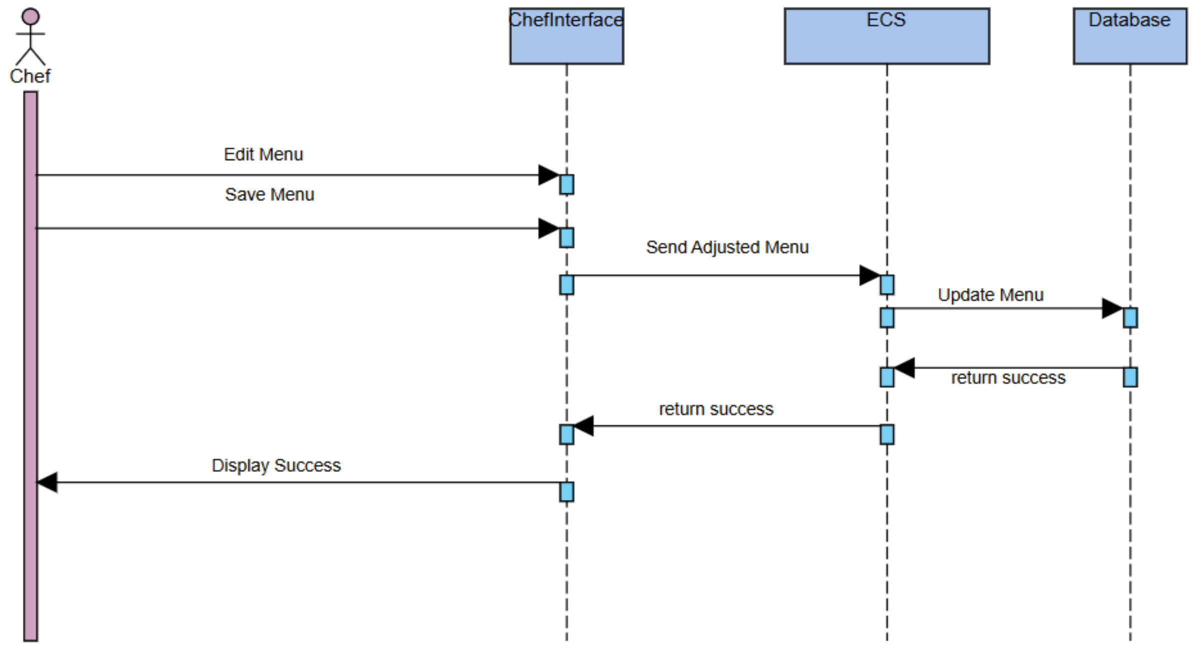


Inventory Tracking using Weight Sensor



Weight Sensor
ECS
Zone
Database
Send Signal: Updated Weight
Retrieve Zone Information
return information
Calculate new total Weight
Update Inventory with new total weight and product info
return success

Edit Menu



Functional Specification Document

User Stories - Functional Requirements

Manage Access

- As a system user, I want the system to have a facial recognition feature so that I can authorize access securely.
- As an authorized individual, I want to be able to add or remove users so that I can manage system access effectively.

Adjust Expiry Dates

- As a system user, I want temperature sensors to monitor refrigerated items so that any change in temperature or humidity is detected.
- As a system user, I want the temperature sensors to send a signal to the system if the temperature falls below a certain threshold so that I can take necessary actions.
- As a system user, I want the system to recalculate the expiry dates of all perishable products based on the temperature sensor's reported values so that the inventory is updated accordingly.
- As a system user, I want the system to check for any items expiring in the next 7 days due to changes in expiry dates so that I can receive related alerts.

Retrieve Relevant Zone

- As a system user, I want zones to be defined within the smart shelves and each zone to contain one weight sensor so that changes in product weight can be detected.
- As a system user, I want the system to retrieve zone information and use RFID or barcode data so that the product type can be identified.
- As a system user, I want the system to calculate the new total weight of products so that newly added or removed quantities are accurately reflected in the inventory.

Manage Menu

- As a chef, I want to be able to create a menu so that I can plan my dishes effectively.
- As a chef, I want to be able to edit the menu so that I can make changes as needed.
- As a chef, I want to be able to view the menus I have created or delete them so that I can keep track of my menu options.

Manage Alerts

- As a chef, I want the system to display and send alerts via email if any item is expiring in the next 7 days so that I can take timely action.
- As a chef, I want to be able to view all the alerts so that I am aware of potential issues.
- As a chef, I want to be able to manage alerts, such as deleting them, so that I can organize and prioritize my notifications.

View Inventory

- As a chef, I want to be able to view the inventory and see all products along with their information, such as expiry dates, so that I can plan effectively.

User Stories - Non-Functional Requirements

Performance

- As a system user, I want the system to process and update inventory data in real-time so that product weight and expiry dates are always accurate.
- As a system administrator, I want the system to handle up to 5000 products simultaneously without performance issues so that operations remain smooth during peak times.
- As a chef, I want alerts for expiring items to be sent within 1 second after a change is detected so that I can take timely actions.

Scalability

- As a system administrator, I want the system to scale to accommodate more products, users, and sensors so that it supports the restaurant's growth.
- As a system administrator, I want the system to support adding new smart shelves and temperature sensors without major modifications so that expansion is seamless.

Reliability

- As a system administrator, I want the system to maintain at least 99.9% uptime so that operations are minimally disrupted.
- As a system administrator, I want the system to log processes and provide error handling during failures (e.g., sensor malfunction) so that issues can be diagnosed and resolved quickly.

Security

- As a system user, I want all user data and product information to be securely stored so that data protection regulations are met.
- As a system user, I want communications between the system and devices (e.g., RFID readers, sensors) to be encrypted so that data integrity is preserved.

Usability

- As a chef or authorized individual, I want the system to have a user-friendly interface so that I can operate it effectively with minimal training.
- As a chef, I want alerts to include clear details (e.g., item name, expiry date, required actions) so that I can act quickly and correctly.

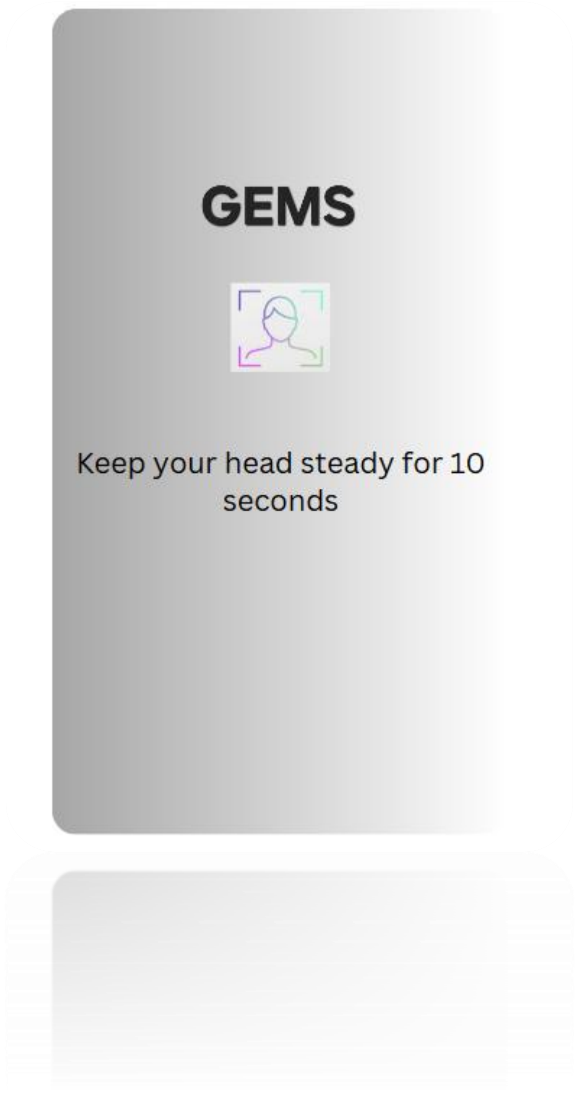
Compatibility

- As a system administrator, I want the system to be compatible with common RFID tags, barcodes, and weight sensors so that it integrates with the existing infrastructure.
- As a system administrator, I want the system to work on multiple operating systems and browsers so that I can manage and monitor it remotely from various devices.

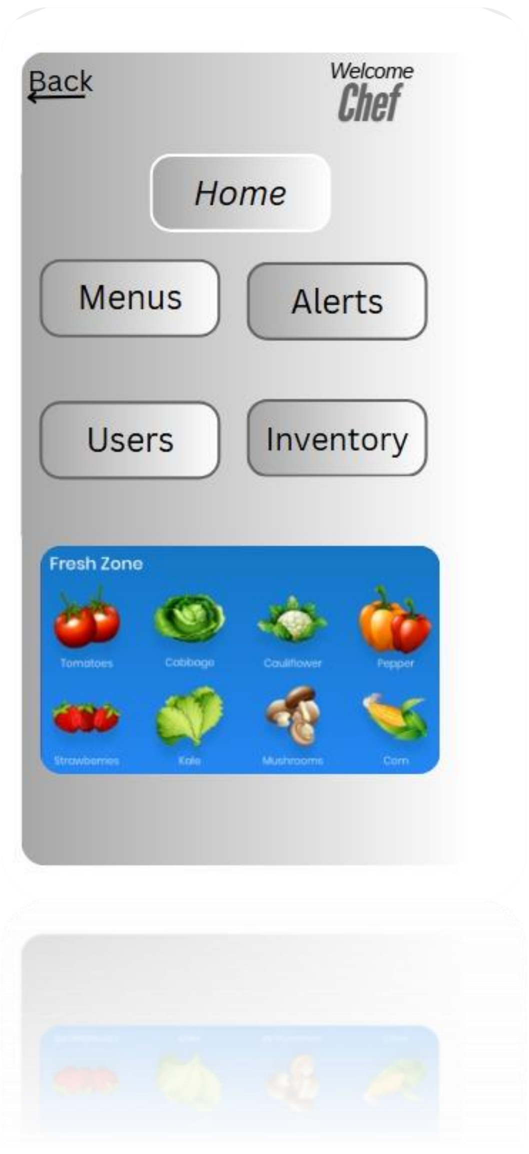
Interface Designs

The following interface designs have been created to give a better idea of the system:

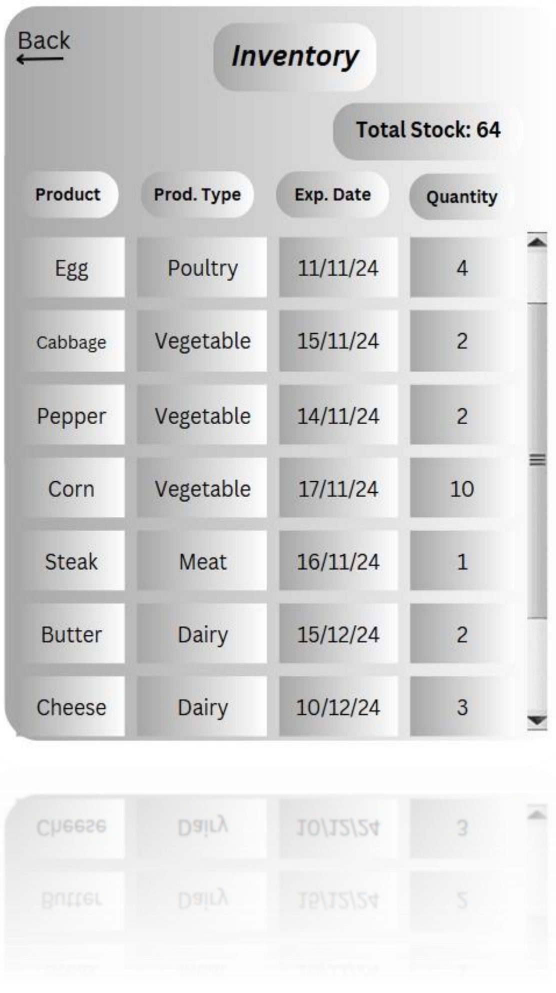
Login



Home Screen



View Inventory



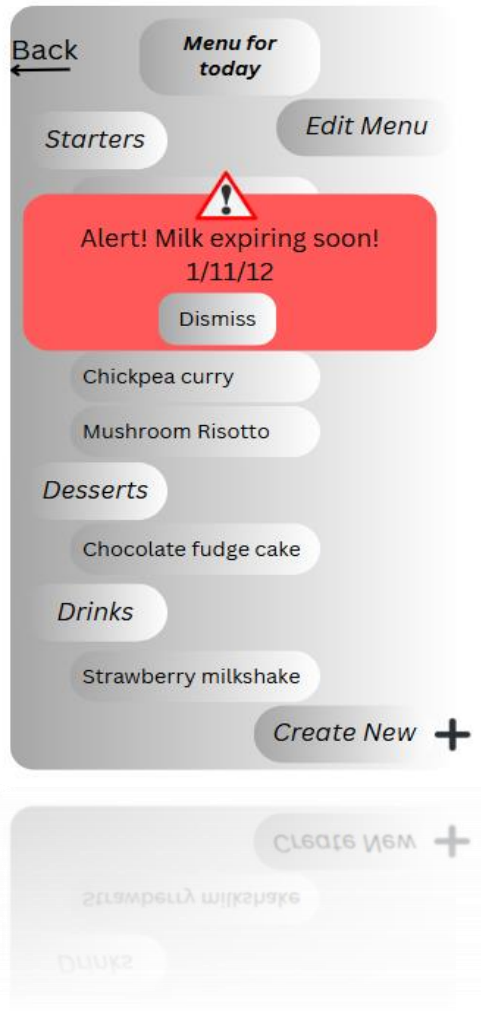
Back

Inventory

Total Stock: 64

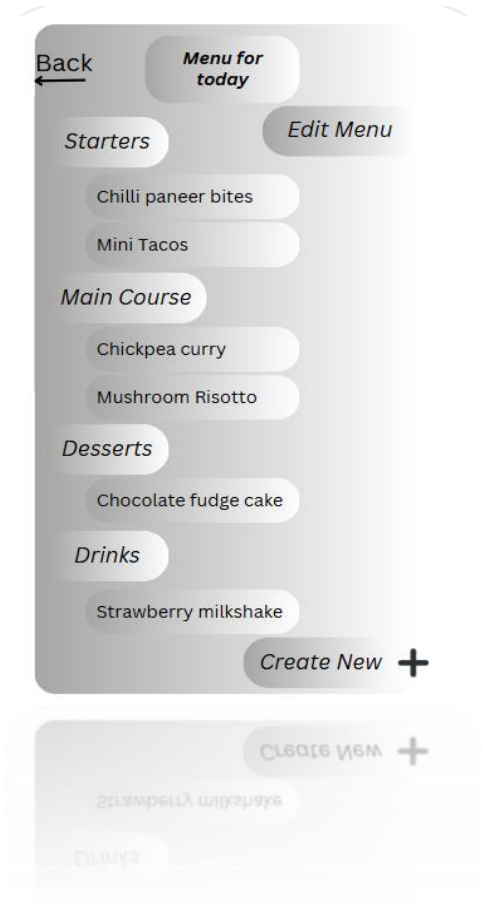
Product	Prod. Type	Exp. Date	Quantity
Egg	Poultry	11/11/24	4
Cabbage	Vegetable	15/11/24	2
Pepper	Vegetable	14/11/24	2
Corn	Vegetable	17/11/24	10
Steak	Meat	16/11/24	1
Butter	Dairy	15/12/24	2
Cheese	Dairy	10/12/24	3

Receiving Alert

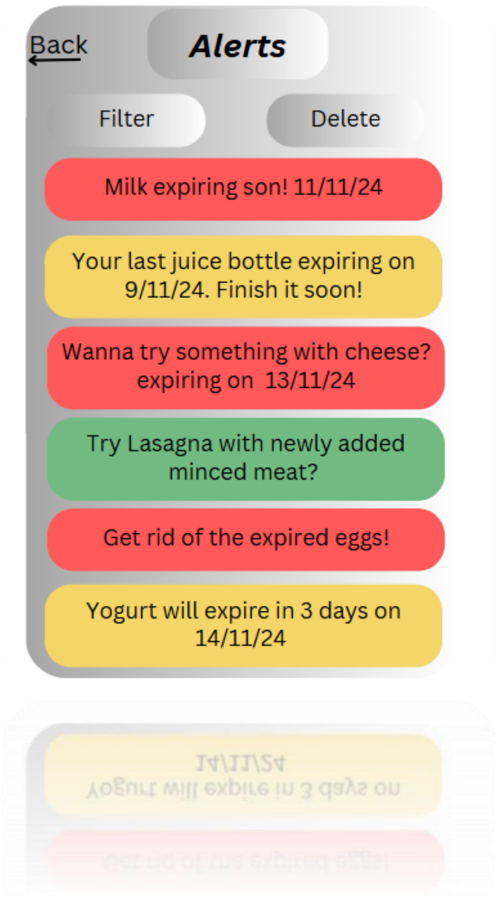


Alert! Milk expiring soon! 1/11/12

Menu



Manage Alerts



Milk expiring son! 11/11/24

Your last juice bottle expiring on 9/11/24. Finish it soon!

Wanna try something with cheese? expiring on 13/11/24

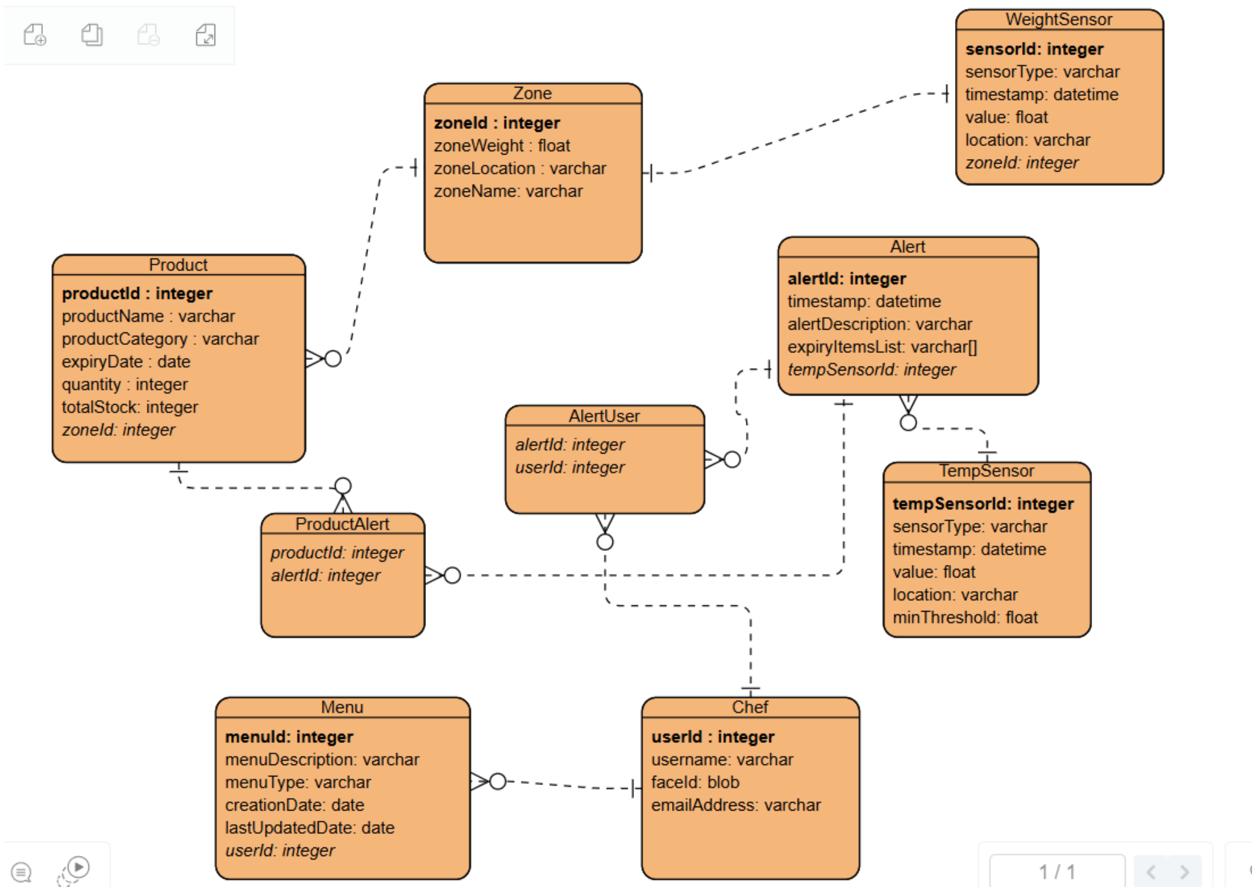
Try Lasagna with newly added minced meat?

Get rid of the expired eggs!

Yogurt will expire in 3 days on 14/11/24

Database Design

The database design for GEMS is shown below. It clearly states all the attributes, cardinalities and constraints.



Database Constraints

Product Table

- **Primary Key(s)**: Product ID
- **Unique Constraint(s)**: Product ID
- **Foreign Key(s)**: Zone ID reference Zone (Zone ID)
- **Not Null Constraint(s)**: Product Name, Product Category, Quantity, Expiry Date, Total Stock

Weight Sensor Table

- **Primary Key(s):** Sensor ID
- **Unique Constraint(s):** Sensor ID

- **Foreign Key(s):)**: Zone ID reference Zone (Zone ID)
- **Not Null Constraint(s):** Sensor Type, Location, Value, Timestamp

Temperature Sensor Table

- **Primary Key(s):** Temperature Sensor ID
- **Unique Constraint(s):** Temperature Sensor ID
- **Not Null Constraint(s):** Sensor Type, Location, Value, Timestamp, Minimum Threshold

Zone Table

- **Primary Key(s):** Zone ID
- **Unique Constraint(s):** Zone ID
- **Not Null Constraint(s):** Zone Name, Zone Location, Zone Weight

Alert Table

- **Primary Key(s):** Alert ID
- **Unique Constraint(s):** Alert ID
- **Foreign Key(s):** Temperature Sensor ID references Temperature Sensor (Temperature Sensor)
- **Not Null Constraint(s):** Alert Description, Alert Timestamp, Expiry Items List

Menu Table

- **Primary Key(s):** Menu ID
- **Unique Constraint(s):** Menu ID
- **Foreign Key(s):** User ID references User (User ID)
- **Not Null Constraint(s):** Menu Description, Creation Date, Last Updated Date, Menu Type

User Table

- **Primary Key(s):** User ID
- **Unique Constraint(s):** User ID, Email Address, Face Id

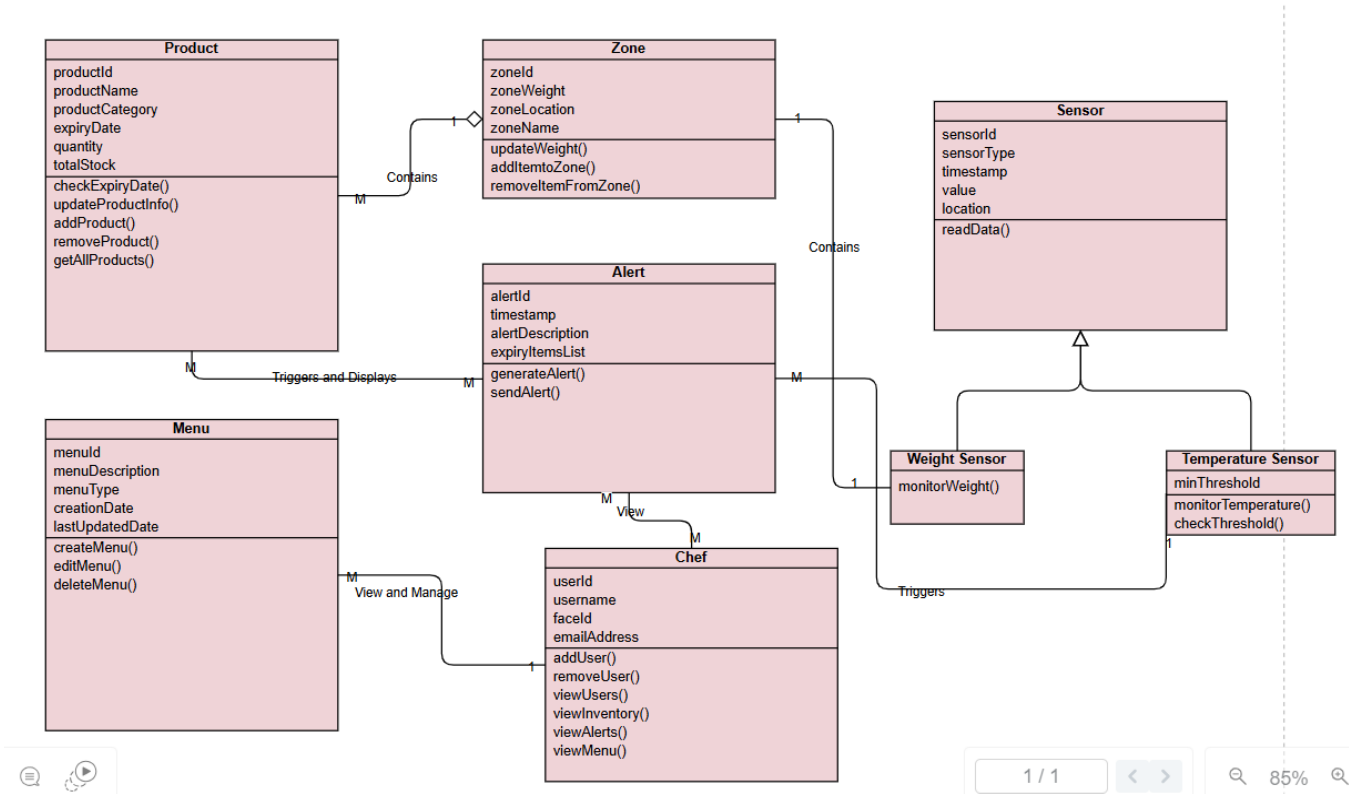
AlertUser Table

- **Foreign Key(s):** Alert ID references Alert (Alert ID), User ID references User (User ID)

ProductAlert Table

- **Foreign Key(s):** Product ID references Product (Product ID), Alert ID references Alert (Alert ID)

Class Diagram with Methods



Software Design

Method Name	AdjustExpiryDates()
Class Name	ECS (Edge Computing System)
ID	Temperature Monitoring #01, Inventory Update #01, Alert #01
Clients (Consumers)	Chef, ECS
Associated Use Case(s)	Adjust Expiry Dates, Update Inventory, Send Alert
Description of Responsibilities	Recalculates expiry dates when temperature thresholds are breached. Sends alerts if necessary.
Arguments Received	Temperature Data (sensor input)
Type of Value Returned	Void
Preconditions	Temperature sensors must be installed and operational. Predefined thresholds must be available.
Postconditions	Expiry dates are updated; alerts are generated for items nearing expiry.

```
Method AdjustExpiryDates(temperatureData)
  For each product in inventory:
    If temperatureData exceeds threshold:
      Log the temperature change.
      Recalculate expiry date using the temperature adjustment formula.
      Update the expiry date in the inventory.
      If new expiry date is within 7 days:
        Generate an alert for the chef.
        Send email and display alert.
      Log recalculated expiry date and alert status.
  End For
End Method
```

Method Name	RetrieveRelevantZone()
Class Name	ECS
ID	Weight Monitoring #02, Inventory Update #02
Clients (Consumers)	Chef, ECS
Associated Use Case(s)	Retrieve Relevant Zone, Update Inventory
Description of Responsibilities	Tracks changes in product quantities using weight sensors through retrieving relevant zone and updates inventory accordingly.
Arguments Received	Weight Change Data (sensor input)
Type of Value Returned	List of Product Data
Preconditions	Weight sensors must be operational. Zones in smart shelves must be predefined.
Postconditions	Inventory is updated with new product quantities.

```
Method RetrieveRelevantZone(weightData)
  Identify zone using weightData.
  Retrieve product data in the zone using RFID/barcode.
  Calculate weight difference:
    New Quantity = Current Quantity ± Weight Change.
  Update inventory with the new quantity.
  Log transaction for audit.
  Return updatedProductData
End Method
```

Method Name	CheckExpiryDates()
Class Name	ECS
ID	Inventory Update #03, Alert #02
Clients (Consumers)	Chef, ECS
Associated Use Case(s)	Check Expiry Dates, Send Alert
Description of Responsibilities	Automates daily checks or check on temperature change trigger for products nearing expiry and generates alerts.
Arguments Received	None (scheduled daily or triggered by the system)
Type of Value Returned	Void
Preconditions	Inventory system must have accurate expiry data.
Postconditions	Alerts are generated and logged for items nearing expiry.

```
Method CheckExpiryDates()
  Retrieve inventory data.
  For each product:
    If expiry date is within the next 7 days:
      Generate alert for the chef.
      Send alert via email and system notification.
      Log alert details.
  End For
End Method
```

Method Name	EditMenu()
Class Name	MenuManagement
ID	Menu Management #04
Clients (Consumers)	Chef

Associated Use Case(s)	Log In, Edit Menu
Description of Responsibilities	Enables chefs to update menus based on inventory availability or preferences.
Arguments Received	List of Changes (chef input)
Type of Value Returned	True or False
Preconditions	Chef must be logged in with permissions; menu must exist in the system.
Postconditions	Menu updates are saved in the system.

```
Method EditMenu(menuChanges)

   Retrieve current menu.

   Apply menuChanges to the retrieved menu.

   Save the updated menu to the database.

   If success

      Return True

   Else Return False

   Log the edit action.

End Method
```

Method Name	ManageAlerts()
Class Name	AlertsManagement
ID	Alerts Management #05
Clients (Consumers)	Chef
Associated Use Case(s)	Log In, Manage Alerts
Description of Responsibilities	Allows chefs to resolve or delete irrelevant alerts to maintain system efficiency.
Arguments Received	Alert management action (resolve or delete)
Type of Value Returned	True or False
Preconditions	Chef must be logged in with permissions; active alerts must exist.
Postconditions	Alerts are resolved or removed from the system.

```
Method ManageAlerts(alertAction)

   Retrieve list of active alerts.
```

```
    Apply alertAction (resolve/delete) to the selected alerts.
    Update alert database to reflect changes.
    If uccess
        Return True
    Else Return False
    Log alert updates.
End Method
```

Project Presentation

Presentation recording:
Presentation PowerPoint:

Project Management Deliverables

Project Activities

Allocation of Activities to Team Members

The group met and collaborated on every aspect of this project.

Weekly Project Timeline

Planned Due Date	Actual Completion Date	Tasks
September 2, 2024	September 2, 2024	Executive Summary
September 2, 2024	September 2, 2024	Systems Proposal
September 9, 2024	September 9, 2024	Business Process Model
September 16, 2024	September 16, 2024	Context Diagram
September 23, 2024	September 23, 2024	Process Model
October 7, 2024	October 7, 2024	Data Dictionary
October 14, 2024	October 14, 2024	Class Diagram without Methods
October 21, 2024	October 21, 2024	Sequence Diagram
October 28, 2024	October 28, 2024	Functional Specification Doc
November 4, 2024	November 4, 2024	Interface Design
November 11, 2024	November 11, 2024	Complete Class Diagram

November 18, 2024	November 18, 2024	Database Design
November 25, 2024	November 25, 2024	Software Design
December 2, 2024	December 2,2024	Document Revision Confirmation

Meeting Minutes

The most important meeting minutes have been described below:

Meeting 1

Details	Notes
Date	September 2, 2024
Time	5PM
Attendees	Saikiran Reddy, Srinidhi, Devang
Discussion	- Finalizing the problem statement for the system. - Defining key deliverables such as BPMN diagrams, use-case diagrams, and data models. - Assigning initial responsibilities for drafting the Systems Proposal.

Meeting 2

Details	Notes
Date	September 9, 2024
Time	5 PM
Attendees	Saikiran Reddy, Srinidhi, Devang
Discussion	- Reviewing the draft of the BPMN diagram. - Discussing the Context Diagram and Process Model deliverables. - Resolving doubts on the structure of the Functional Specification Document.

Meeting 3

Details	Notes
Date	October 7, 2024
Time	5 PM
Attendees	Saikiran Reddy, Srinidhi, Devang

Details	Notes
Discussion	- Reviewing progress on data documentation. - Finalizing the class diagram structure. - Planning interface designs and database layouts.

Meeting 4

Details	Notes
Date	November 25, 2024
Time	5 PM
Attendees	Saikiran Reddy, Srinidhi, Devang
Discussion	- Reviewing the draft of the Software Design deliverables. - Validating the 5 documented methods and algorithms. - Finalizing the project deliverables list.

Meeting 5

Details	Notes
Date	November 30, 2024
Time	5 PM
Attendees	Saikiran Reddy, Srinidhi, Devang
Discussion	- Reviewing all deliverables for completeness and consistency. - Finalizing project documentation, including diagrams, specifications, and data models. - Discussing the submission process and next steps.

Meeting 6

Details	Notes
Date	December 2,2024
Time	5 PM
Attendees	Saikiran Reddy, Srinidhi, Devang
Discussion	- Finalizing the content for the 10-minute voice-over PowerPoint presentation. - Assigning specific sections of the presentation to each team member. - Reviewing design and layout of slides for clarity and professionalism. - Discussing voice-over recording process and timing for each team member.