FPT UNIVERSITY

CAPSTONE PROJECT REPORT

IoT Application Used in Food Inventory Management

Class Name	IOP490 G2
Group Members	Ta Viet Duc - HE163775 Truong Quoc Bao - HE161900 Ha Quoc Viet - HE163904
Supervisor	Hoang Xuan Son
Ext Supervisor	None
Capstone Project Code	SFoodInventory

Hanoi, December 2024

Contents

A Appendix A: List of Tables

B Appendix B: List of Figures

Abstract

Managing perishable inventory is a critical challenge for restaurants, as improper handling can lead to food spoilage, financial losses, and operational inefficiencies. This thesis presents an innovative IoT-based solution to address these issues, focusing on real-time monitoring and data-driven decision-making for food inventory management. The proposed system combines load cells and temperature-humidity sensors with an ESP32 microcontroller to monitor key storage parameters, such as food weight, expiration dates, and environmental conditions. By detecting deviations and providing timely notifications, the system proactively prevents spoilage and reduces waste. Additionally, a mobile application enables centralized inventory control, offering real-time data visualization, expiration alerts, and low-stock notifications. The integration of data analytics provides actionable insights to optimize inventory planning and usage, enhancing operational efficiency. Designed specifically for restaurant settings, this system aims to transform inventory management by ensuring food safety, minimizing waste, and supporting sustainability through the adoption of IoT technologies. This thesis evaluates the system's effectiveness in improving inventory control while reducing costs and environmental impact.

Keywords: IoT (Internet of Things), Restaurant inventory management, Food spoilage prevention, Sustainable food practices, Inventory optimization

1 Introduction

Restaurants face significant challenges in managing perishable food inventory effectively. Inefficient inventory management often results in food waste, financial losses, and compromised customer satisfaction. According to the Food and Agriculture Organization (FAO), nearly one-third of the food produced globally is wasted each year, with a large portion originating from the retail and foodservice sectors [1]. This issue is particularly pressing for restaurants, where rapid inventory turnover and stringent storage requirements directly impact both profitability and sustainability.

This thesis introduces an IoT-based smart food inventory management system tailored for restaurant environments. The system utilizes load cells to measure food weight and temperature-humidity sensors to ensure optimal storage conditions. These components are connected to an ESP32 microcontroller, enabling continuous real-time monitoring and empowering restaurant staff to promptly address potential issues to prevent spoilage and minimize waste. Additionally, a mobile application provides users with a centralized platform to monitor inventory in real-time, view food tray statuses, track expiration dates, and receive timely alerts for low stock or expiring items, enhancing overall inventory control and food safety.

The increasing adoption of IoT technologies in the foodservice industry highlights their potential to reduce waste, lower costs, and improve operational efficiency [2]. Recent research has demonstrated that IoT-based systems significantly optimize food storage and usage, helping businesses reduce spoilage and waste [3]. Moreover, the integration of mobile applications into inventory management provides restaurant staff with actionable insights, allowing for better decision-making and improved operational performance [4].

The proposed system is scalable and adaptable to various restaurant settings, from small eateries to larger establishments. By leveraging IoT technologies, restaurants can improve inventory accuracy, reduce food spoilage, and enhance their overall operational efficiency. This thesis aims to showcase how IoT can transform food inventory management in the restaurant industry, supporting a more sustainable and cost-effective approach to food service.

2 Literature review

2.1 International Research

The application of Internet of Things (IoT) technologies in food inventory management has transformed how food stocks are monitored, controlled, and optimized, addressing critical issues such as waste reduction, food safety, and operational efficiency. IoT-enabled systems leverage interconnected sensors, devices, and cloud-based platforms to provide real-time data on inventory levels, storage conditions, and supply chain activities. This real-time monitoring capability significantly enhances decision-making by providing stakeholders with accurate and timely insights into inventory status, reducing the risks of overstocking, understocking, and food spoilage [5]. One of the most significant contributions of IoT in this field is its role in ensuring food safety and quality. IoT devices are equipped to monitor environmental factors such as temperature, humidity, and light exposure—parameters critical for maintaining the freshness and safety of perishable goods. For instance, IoT systems can alert managers if temperatures deviate from acceptable ranges in cold storage, thereby preventing potential losses due to spoilage. Studies have also shown that integrating IoT with blockchain technology improves traceability and accountability, enabling seamless tracking of food products throughout the supply chain [6]. This integration helps detect contamination sources promptly, reducing risks to public health and enhancing consumer trust. IoT applications also improve cost-efficiency by automating many labor-intensive processes, such as inventory tracking and reordering. Automation reduces human error, cuts operational costs, and ensures a consistent flow of information across all levels of inventory management. According to research, IoT-enabled inventory systems have led to a 25% reduction in overall costs for food storage and management in certain sectors [7]. Furthermore, these systems support predictive analytics, allowing businesses to forecast demand accurately and optimize stock replenishment schedules. Despite these benefits, challenges remain in adopting IoT for food inventory management. High initial investment costs, technical complexity, and concerns over data security and privacy are some of the primary barriers to widespread implementation. Additionally, smaller businesses often lack the infrastructure and expertise required to deploy and maintain IoT systems effectively [8]. Addressing these challenges requires collaborative efforts between technology developers, policymakers, and industry stakeholders to ensure that IoT solutions are accessible, secure, and scalable. IoT applications in food inventory management present transformative opportunities to enhance efficiency, improve food safety, and reduce costs. However, their successful implementation demands overcoming existing

challenges, particularly in affordability and security. Future research should focus on creating cost-effective, secure, and user-friendly IoT solutions that can be widely adopted across different scales of operations.

2.2 Related work

2.2.1 IoT in Food Inventory Tracking

A study by Omkar Mulay et al. (2020) presented an IoT-based food inventory tracking system designed for both domestic and commercial kitchens. This system utilized load cells, ESP8266 microcontrollers, and MQTT protocols to monitor inventory levels and send notifications when critical thresholds were reached, such as when inventory dropped below 10%. The system emphasized real-time tracking, automated inventory updates, and efficient alert mechanisms. By integrating cloud computing, the system enhanced data accessibility and visualization, allowing users to track consumption patterns over time. This research highlights IoT's potential to streamline inventory management, reduce manual effort, and minimize food waste, offering practical insights into the automation of food inventory systems [9].

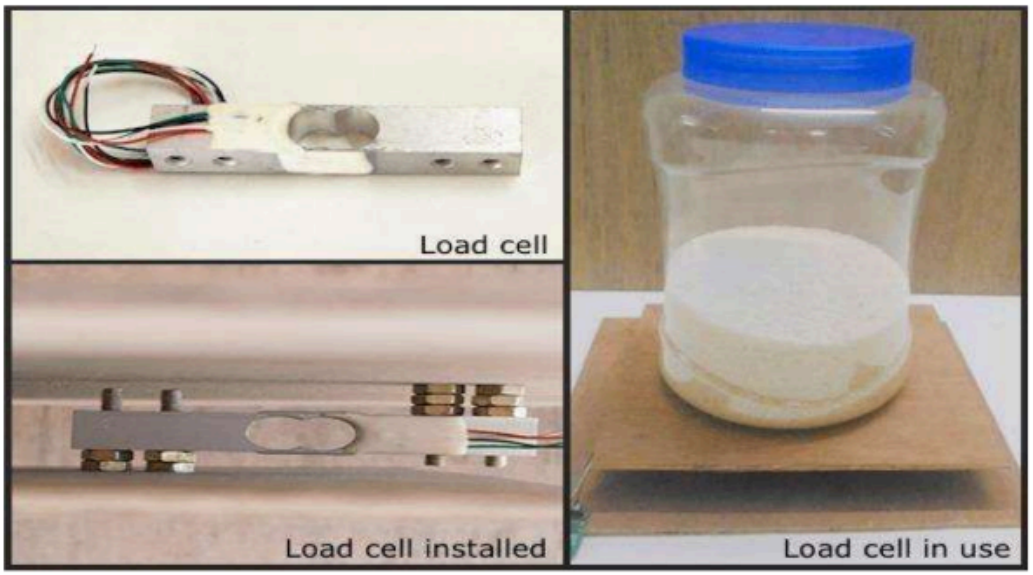


Figure 1: Smart kitchen cabinet

2.2.2 IoT-Based Smart Warehouse Monitoring

In another notable contribution, Muhammad Ehsan Rana et al. (2023) explored the application of IoT in warehouse inventory tracking. They proposed a multi-layer IoT architecture consisting of sensor, communication, application support, and application

layers. This architecture used sensors to monitor temperature, humidity, and stock levels to ensure optimal storage conditions for perishable items. Data was processed and visualized via a web-based platform, providing real-time monitoring and predictive analytics to improve efficiency and reduce costs. The research underscores the importance of automation and environmental monitoring, which aligns closely with the goals of the current project. The use of DHT11 sensors for environmental tracking in food storage environments illustrates a practical IoT application for improving inventory management [10].

2.3.3 IoT for Real-Time Inventory Analytics

Mustafa Sabri et al. (2021) presented an IoT-based smart warehouse monitoring system for both domestic and commercial use. The system employed load cells for precise weight measurement and used cloud services for data storage and analytics. The system's real-time monitoring capabilities allowed users to identify underused or overstocked inventory, providing actionable insights to optimize purchasing decisions. The integration of NodeMCU microcontrollers and load sensors with cloud platforms to improve inventory visualization and analytics further emphasizes the value of IoT-enabled weight measurement tools. This approach highlights how IoT can enhance decision-making by providing real-time insights and statistical analysis of food consumption, which is integral to the proposed system [11].

Figure 2: 3D prototype

These studies collectively demonstrate the transformative potential of IoT in inventory management. By combining real-time monitoring, sensor-based data collection, and cloud integration, they address critical challenges such as **f**ood waste, storage optimization, and the reduction of manual intervention. The proposed IoT-based food inventory management system builds upon these advancements by incorporating automatic expiration date management, load cell-based weight monitoring, and environmental condition tracking. The system further differentiates itself with advanced analytics and automated notifications, making it suitable for both domestic and commercial applications.

3 System architecture

3.1 General IoT system

A general IoT system consists of interconnected devices and services working together to collect, process, and transmit data for decision-making and automation. Typically, such systems include sensors, communication networks, data processing units, and user-facing applications:

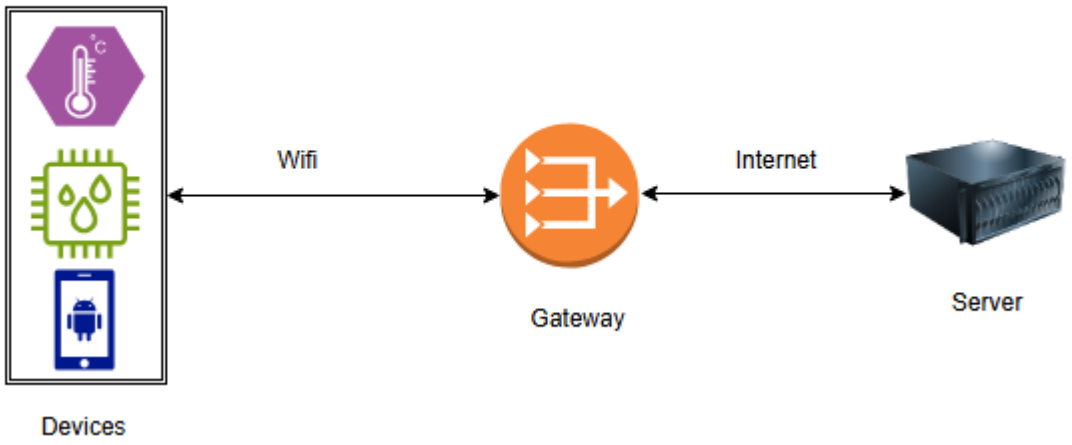


Figure 3: General IoT System Architecture

3.2 Food Inventory System architecture

3.2.1 General architecture

The architecture of the proposed system leverages IoT technology in combination with REST API and WebSocket communication to enable efficient and real-time management of food inventory. At its core, the system integrates hardware components, such as load cell sensors for monitoring food weight and DHT11 sensors for tracking temperature and humidity, to ensure precise data collection from the storage environment. These sensors are connected to an ESP32 microcontroller, which acts as the central node for gathering sensor data and transmitting it to the backend system.

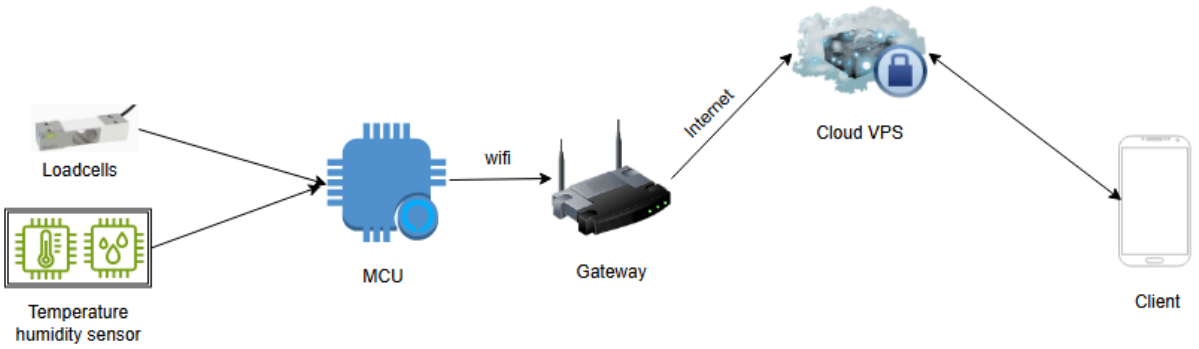


Figure 4: Architecture diagram of a food warehouse management system for restaurants

The backend, developed using Spring Boot, serves as the backbone of the system, providing RESTful APIs for handling standard client requests such as fetching inventory data, updating item information, and managing user configurations. Simultaneously, WebSocket technology enables real-time bi-directional communication between the server and client, facilitating instant notifications for events like low stock alerts, temperature deviations, or food expiration warnings.

For user interaction, a mobile application, developed using Android Java, and a responsive web application provide intuitive interfaces. These applications allow users to monitor inventory levels, configure expiration dates, and analyze food consumption patterns. The applications communicate with the backend via REST API for routine operations, while WebSocket ensures users receive instantaneous updates on critical events, such as real-time changes in environmental conditions or stock levels.

The system also relies on a centralized database to store inventory data, sensor readings, and user configurations. This architecture ensures scalability, allowing the system to handle diverse use cases ranging from small kitchens to large commercial storage facilities. By combining the power of IoT devices with modern communication protocols, the proposed system delivers a comprehensive, real-time solution for food inventory management.

3.2.2 Hardware Structure

Figure 5 demonstrates the components of the SFoodInventory system, which include weight sensors, temperature sensor, microcontroller, display device and power supply. When the microcontroller is powered, it receives data from weight sensors, a temperature sensor and pushes that data to the server. The display device helps to return

notifications from the app, displaying temperature and humidity. The microcontroller is critical for data processing and server export.

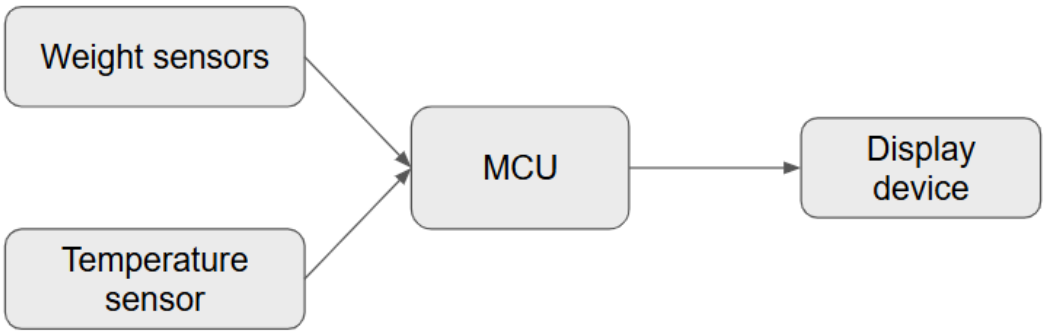


Figure 5: Hardware Diagram

3.2.3 Server

The server plays a pivotal role in the food inventory management system, acting as the central hub for data processing and system coordination. It manages the flow of information between various components, including the hardware (sensors and devices), the database, and the mobile application. Its primary responsibility is to ensure seamless and efficient communication across the system, supporting real-time data collection, processing, and dissemination. A critical function of the server is to manage inventory data dynamically. It continuously monitors stock levels, records food items as they are added, removed, or updated, and ensures that all information remains synchronized across different parts of the system. This real-time data management is essential for providing accurate and up-to-date insights into inventory status, helping users make timely decisions regarding restocking or discarding expired items. The server also plays a key role in ensuring system security and access control. It supports user authentication and manages permissions to ensure that only authorized users can perform specific actions, such as adjusting inventory or accessing sensitive reports. This role-based control framework helps maintain the integrity of the system and protects against unauthorized access or accidental data manipulation. Another important aspect of the server's operation is its ability to generate and send notifications. It actively tracks critical events, such as items nearing their expiration date or sudden changes in storage conditions (e.g., temperature or humidity). When such events are detected, the server promptly sends alerts to relevant users, enabling them to take immediate corrective action. These notifications ensure that potential issues are addressed before they escalate, reducing food waste and maintaining optimal storage conditions. In addition to

real-time monitoring and notifications, the server supports routine operational tasks. It automates processes such as generating daily inventory reports, summarizing stock levels, and providing usage trends. These reports help users gain valuable insights into inventory dynamics, identify patterns of consumption, and plan future procurement effectively. The server acts as the backbone of the system, orchestrating all key operations and ensuring that the food inventory management process is streamlined, responsive, and reliable. Its design supports efficient inventory handling and enhances the overall user experience by providing accurate and actionable information at all times.

3.2.4 Mobile Application Structure

The mobile application for the IoT-based food inventory management system is designed to provide an intuitive and responsive user experience for managing inventory, tracking storage conditions, and receiving real-time alerts. Its primary goal is to enable users to efficiently interact with the system, offering features such as inventory overview, detailed item information, and notification handling.

The application is structured using the Model-View-ViewModel (MVVM) architectural pattern, which ensures a clean separation of concerns, promoting easier maintenance, testing, and scalability. This modular design enhances the application's flexibility, allowing for future feature expansion and integration with various backend services.

- **Model Layer**: Manages core data and business logic. This layer handles interactions with backend services to retrieve data such as inventory levels, expiration dates, and sensor readings. Additionally, it integrates a local database to ensure data availability during offline use.
- **ViewModel Layer**: Acts as a mediator between the Model and View layers. It fetches data from the Model, processes it, and prepares it for presentation. The ViewModel is lifecycle-aware, persisting data through configuration changes such as screen rotations, which prevents unnecessary data reloads and enhances performance. It also leverages LiveData objects to notify the View of any data changes, enabling efficient and dynamic UI updates.
- **View Layer**: Responsible for rendering the user interface and handling user interactions. Activities and Fragments in this layer observe LiveData objects from the ViewModel to display real-time data updates. For instance, the inventory screen dynamically updates the list of food items, showing attributes like weight and expiration date. The View layer delegates user actions, such as adding a new item or removing

expired stock, to the ViewModel, ensuring minimal business logic within the UI components.

By following the MVVM architecture, the application ensures a seamless and reactive user experience. The design not only facilitates real-time data synchronization with backend services but also enhances code maintainability and testability, allowing for robust and scalable application development.

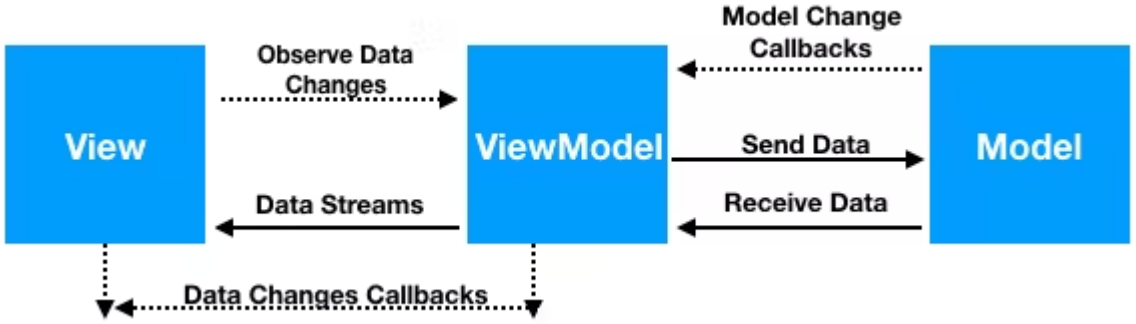


Figure 6: Mobile Application Structure

3.2.5 Database

The database serves as the foundation for managing and storing all critical information within the food inventory management system. It organizes food items by attributes such as type, quantity, expiration date, and storage location, enabling efficient inventory tracking. Each item is linked to specific trays, making it easy to monitor their exact position within the storage system. The database also records environmental conditions, such as temperature and humidity, which are regularly captured to ensure food items are stored under optimal conditions. Additionally, the system maintains detailed logs of food consumption, tracking usage patterns and providing insights for better stock management. User information is stored to support account management, including authentication and tracking individual activities within the system. Historical data is carefully recorded, enabling the generation of reports and statistics that help identify trends, such as frequently used items or periods of high consumption. By structuring this data logically, the system ensures that all relevant information is easily accessible, enabling users to make informed decisions about inventory management. This streamlined approach helps prevent food waste, ensures timely replenishment of stock, and maintains high-quality storage conditions to meet the demands of effective food inventory control.

4 Server

The server is the central component of the IoT-based Food Inventory Management System, ensuring data processing, secure communication, and seamless interaction between IoT devices, the database, and the mobile application. Built with Spring Boot, hosted on a VPS, and managed with Nginx, HTTPS, and Jenkins, the server ensures reliable performance and scalability. Below, the server's components and functionalities are categorized into distinct sections for clarity.

4.1 Server Responsibilities

The server fulfills multiple critical roles in the system. It processes data collected from IoT devices, such as load cells and DHT11 sensors, ensuring accurate inventory updates and monitoring environmental conditions like temperature and humidity. The server also implements business logic for core functionalities, including expiration date tracking, low-stock alerts, and automated notifications for approaching expiration dates. Through its RESTful APIs and WebSocket integration, the server enables real-time communication with the mobile application, ensuring users are promptly informed about critical events. Additionally, the server handles user authentication and authorization, enforcing role-based access control for secure interactions.

4.2 Spring Boot

Spring Boot is the primary framework used for server-side development in this system. After evaluating other frameworks like Django, Flask, and Express.js, Spring Boot was selected due to its flexibility and simplicity in developing web applications and RESTful services in Java. It streamlines application configuration, allowing developers to focus more on functionality rather than setup, and it supports a wide variety of integration options with databases, transaction management, and scheduling. The framework provides default configurations for common tasks and integrates well with Spring-based projects, including Spring Data JPA, Spring Security, and Spring Scheduler.

Table 1: Framework Evaluation Criteria

Criteria	**Spring boot**	**Django**	**Node.js**
Performance	High, optimized for stable and scalable systems	Good, suitable for small to medium applications	High, with non-blocking I/O, but might struggle under heavy load
Scalability	Excellent, supports microservices and distributed architecture	Good, but may face challenges when scaling large systems	Very good, flexible scalability due to non-blocking nature
Security Management	Spring Security, robust and customizable	Django Security, easy to configure	Security libraries need to be integrated manually
Community and Documentation	Large community, comprehensive documentation	Large community, easy to access documentation	Huge community, extensive documentation and library support
Reason for Choosing	Spring Boot is ideal for distributed systems, with high security and excellent scalability, making it a perfect fit for IoT applications where performance and long-term scalability are crucial.	Django is also a good choice for rapid web development but doesn't handle distributed systems as efficiently as Spring Boot.	Node.js is fast and efficient for asynchronous tasks, but Spring Boot offers better security and long-term scalability for large systems.

Spring Boot is particularly advantageous in large, distributed systems like IoT, where high performance, scalability, and maintainability are critical. According to the 2024 JetBrains Developer Ecosystem survey, Spring Boot remains one of the most popular frameworks for backend Java development, with more than 50% of developers using it in their projects [12]. The combination of automatic configuration and extensive support for cloud platforms makes it an ideal choice for modern enterprise applications.

Reason for Choosing Spring Boot:

Spring Boot was chosen due to its high scalability, robust security, and ability to easily integrate with other technologies. Moreover, it has been proven over time in large-scale systems that require stability and performance.

4.3 Nginx

Nginx was selected as the reverse proxy server to distribute load and handle HTTPS connections. When compared to other web servers like Apache HTTP Server and HAProxy, Nginx stands out due to its high concurrency and low resource consumption. Nginx not only handles load balancing but also optimizes caching, which improves the performance of the server, particularly in high-traffic scenarios.

Table 2: Web Server Evaluation Criteria

Criteria	**Nginx**	**Apache**	**Caddy**
Performance	High, optimized for handling large numbers of simultaneous connections with low resource usage	Good, but not optimized for handling large numbers of simultaneous connections	Good, easy to configure but less popular than Nginx
Scalability	Supports SSL/TLS, HTTP2, and secure reverse proxy architecture.	Supports SSL/TLS but more complex configuration.	Built-in SSL support, easy to configure
Ease of Configuration	Easy to configure, but requires some learning	Flexible but more complex to configure	Easy to configure for basic use cases
Load Balancing	Strong and easy to configure	More complex to configure, but still effective	Built-in, easy for simple use cases
Reason for Choosing	Nginx is chosen for its ability to handle millions of	Apache is very flexible but does not perform as	Caddy is a simpler choice for basic web serving, but

	simultaneous connections, optimize web traffic, and provide secure reverse proxying, making it an ideal choice for load balancing and managing secure communications.	efficiently as Nginx under high loads with many concurrent connections.	Nginx has proven superior performance and scalability for handling large-scale web traffic.

Additionally, Nginx provides enhanced security by effectively managing connections and ensuring that HTTPS traffic is encrypted. As reported by Netcraft in 2024, Nginx holds over 30% of the global web server market share, making it one of the most reliable and scalable choices for handling web traffic [13]. Its ability to scale efficiently under heavy traffic loads while maintaining high security is why it was chosen for this system.

Reason for Choosing Nginx:

Nginx was chosen because of its proven ability to efficiently handle large-scale web traffic, perform load balancing, and ensure secure HTTPS communication. This is particularly important in a system with high traffic, such as IoT systems.

4.4 Jenkins

Jenkins was chosen as the continuous integration and continuous deployment (CI/CD) tool for automating the software delivery pipeline. Jenkins was selected after evaluating other CI/CD solutions like GitLab CI and CircleCI, with Jenkins standing out due to its extensibility and strong community support [14]. Jenkins integrates seamlessly with a variety of tools, allowing for automatic testing, building, and deployment of code changes [15].

According to the DevOps Research and Assessment (DORA) group, organizations using Jenkins significantly reduce development and deployment times, improve software quality through continuous testing, and mitigate risks associated with software deployment failures [16]. This automation ensures faster and more reliable updates, making Jenkins an essential tool for maintaining continuous delivery in the system.

Table 3: CI/CD Tool Evaluation Criteria

Criteria	Jenkins	GitLab CI/CD	CircleCI
Community	Large community, extensive plugin support	Strong community within the GitLab ecosystem	Smaller community, but rapidly growing
Scalability	Excellent, supports many plugins and integrations	Good, easy to integrate within GitLab	Good, but requires more configuration for larger projects
Ease of Configuration	Flexible configuration, but requires learning curve	Easier to use if already using GitLab	Easy-to-use interface, good for simple CI/CD pipelines
Features	Automates CI/CD, supports diverse plugin ecosystem	Integrated CI/CD in GitLab, easy to expand	Quick CI/CD with support for external tools
Reason for Choosing	Jenkins was chosen for its high flexibility in automating CI/CD processes, supporting large, complex systems with customizable workflows.	GitLab CI/CD is easy to use if you're already using GitLab, but Jenkins supports more plugins and customization.	CircleCI is good for smaller projects but lacks the depth of customization and plugin support offered by Jenkins.

Reason for Choosing Jenkins:

Jenkins was selected for its extensive plugin ecosystem and flexibility in automating CI/CD pipelines. It supports complex systems and workflows, which is essential for large-scale IoT applications requiring continuous integration and deployment.

4.5 Implementation Details

The server is implemented using Spring Boot, providing a lightweight yet scalable framework for enterprise-level applications. It exposes RESTful APIs for operations like adding, updating, or retrieving inventory data. WebSocket is integrated for real-time notifications, pushing alerts directly to the mobile application whenever a predefined condition, such as a low-stock threshold or an expired item, is met. For database interactions, the server utilizes Spring Data JPA, enabling efficient querying and seamless communication with Azure SQL Server. The business logic is modularized to ensure that operations like expiration tracking and report generation can be maintained or extended easily.

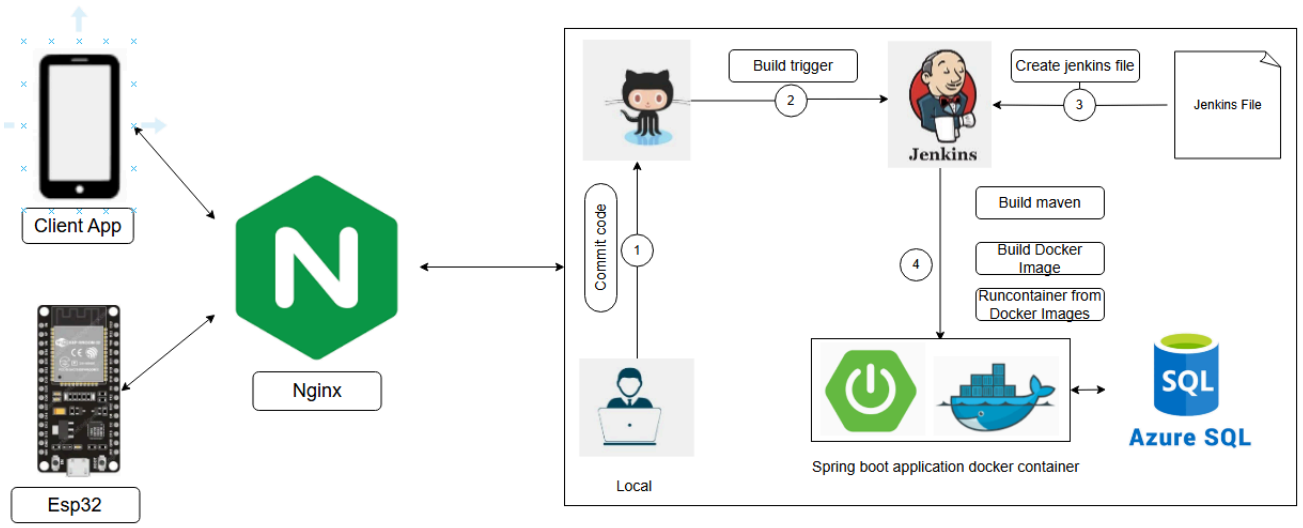


Figure 7: Server architecture

4.6 Server Deployment Workflow with CI/CD

The deployment and maintenance of the server are streamlined using Jenkins for continuous integration and deployment. Jenkins automates the process of building the Spring Boot application from the source code, running tests, and deploying the server to the VPS. Automated testing ensures that each build functions correctly, while error alerts and rollback mechanisms maintain system stability. Additionally, the server automates recurring tasks, such as generating inventory reports and sending scheduled notifications, using Spring Boot's task scheduling framework. The server deployment process follows a streamlined Continuous Integration/Continuous Deployment (CI/CD) pipeline managed by Jenkins, ensuring that new features or updates are deployed efficiently with minimal downtime. The workflow involves the following steps:

Step 1 : Code Commit

Developers write and test code locally before committing it to a GitHub repository. This serves as the central version control system where the server code is maintained.

Step 2: Build Trigger

Jenkins monitors the GitHub repository for new commits. When a change is detected, Jenkins triggers a build process automatically.

Step 3: Jenkins Build Process

Jenkins starts by fetching the latest code from the GitHub repository. The application is built using Maven, ensuring all dependencies are resolved, and the code is compiled into an executable artifact.

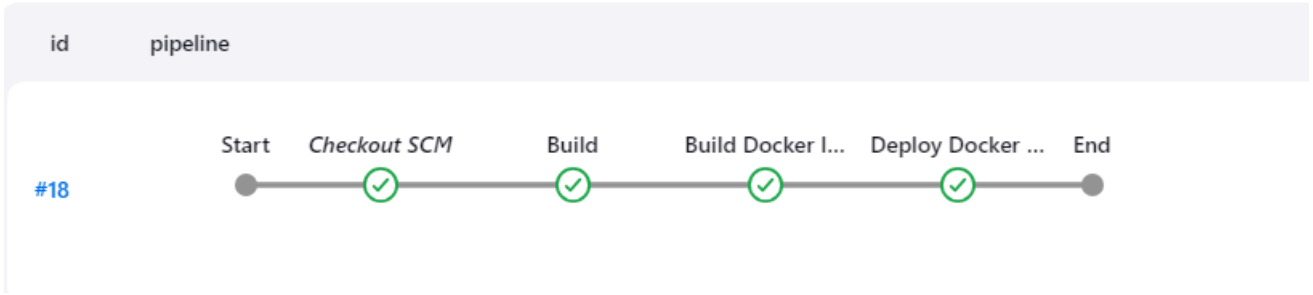


Figure 8: Jenkins build process

Step 4: Docker Image Creation

Jenkins packages the Spring Boot application into a Docker image. This containerized image ensures consistent behavior across development, testing, and production environments.

Step 5: Deployment

The Docker container is deployed to the VPS, replacing the previous version of the application without downtime. Jenkins verifies the deployment, ensuring that the new application version is running correctly.

4.7 Server Security and Communication

The server integrates multiple layers of security to protect sensitive data and ensure secure communication between clients and devices. All data exchanged between the mobile application, IoT devices, and the server is encrypted using HTTPS, enabled

through SSL/TLS certificates managed by Nginx. User authentication is handled via Google OAuth2, ensuring that only verified users can access the system. Role-based access control is implemented to restrict access to sensitive operations based on user roles, such as admin or regular users. Additionally, firewall configurations on the VPS prevent unauthorized access by allowing only trusted IPs to interact with the server.

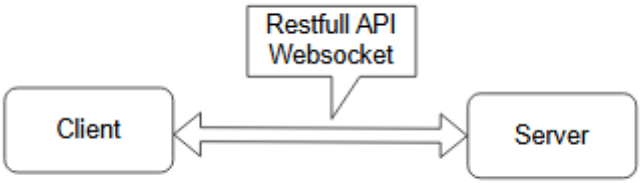


Figure 9: Client - Server Communication

Real-time communication is facilitated by WebSocket, which establishes a persistent connection between the server and the mobile application. This connection allows the server to push notifications instantly to users, ensuring timely updates about critical events such as expiring food items or low inventory levels. The use of WebSocket reduces latency compared to traditional HTTP requests, improving the overall user experience.

4.8 Google OAuth2

Google OAuth2 was chosen for user authentication and authorization, ensuring that only authenticated users can access the system. OAuth2 provides a secure way to manage access permissions by using a third-party authentication provider like Google. This reduces the risk of password-related security breaches and simplifies the login process for users.

OAuth2 is widely used and trusted in the industry, as it follows the highest security standards for authentication. According to the OpenID Foundation, Google OAuth2 provides a robust mechanism for managing user identity and permissions, making it an ideal solution for systems requiring secure and simple login flows [17]. By using Google OAuth2, the system ensures that user credentials are kept secure while streamlining the authentication process.

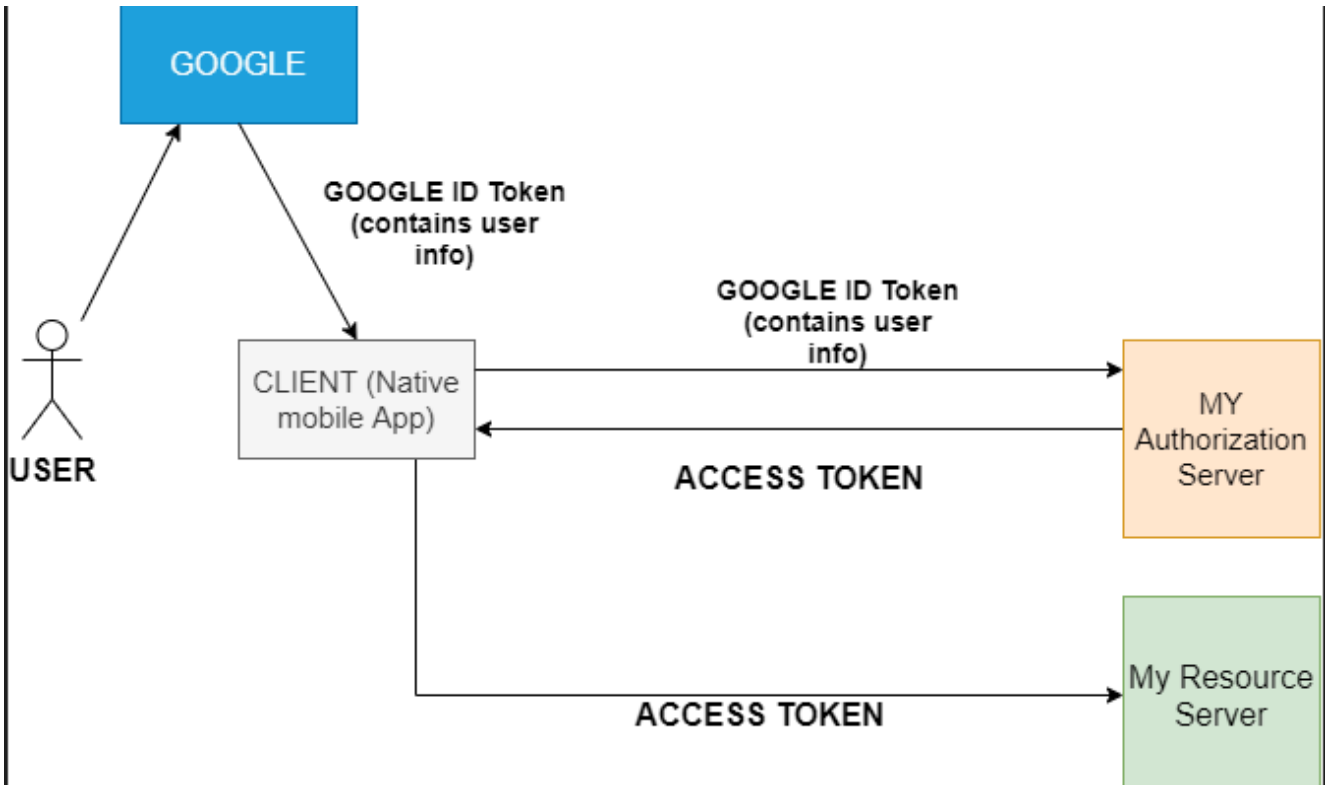


Figure 10: Google OAuth 2.0 Authentication Flow [18]

The Authorization Code flow is as follows:

User Authentication with Google:

The user initiates the login process through the client application (native mobile app).

The client app redirects the user to Google's OAuth 2.0 endpoint for authentication.

Upon successful authentication, Google issues an ID Token containing the user's information.

Client Validation:

The client app receives the Google ID Token and sends it to the custom Authorization Server.

Authorization Server Token Exchange:

The Authorization Server validates the received Google ID Token by verifying its signature and checking the claims (e.g., audience, expiration). After validation, the Authorization Server issues an Access Token for the client.

Access to Resource Server:

The client app uses the issued Access Token to make authorized requests to the Resource Server (e.g., to access protected resources or APIs).

4.9 Integration with Azure SQL Server

The server interacts seamlessly with Azure SQL Server, which serves as the central database for the system. Through Spring Data JPA, the server executes CRUD operations efficiently, enabling real-time updates to the inventory, user data, and environmental readings. For example, when an IoT device sends updated sensor data, the server processes it and updates the corresponding entries in the database. Optimized queries and indexing ensure that data retrieval and reporting are fast, even as the system scales to handle larger volumes of data.

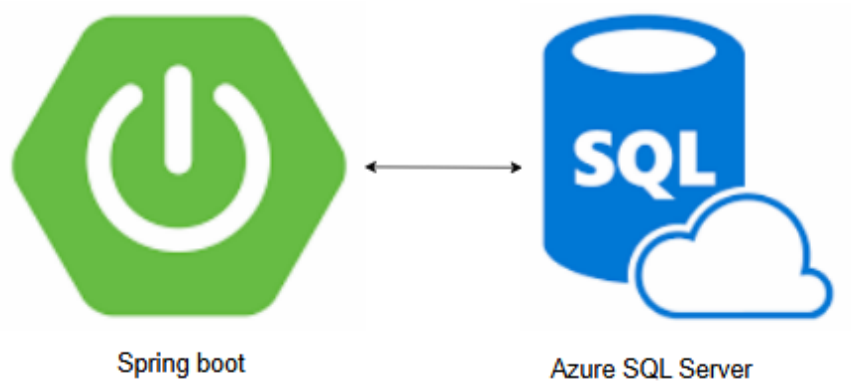


Figure 11: Integrate SQL Server with Spring boot

5 Hardware

5.1 Objectives

Restaurants need scales to weigh the amount of food in their warehouses. And managers need to control and monitor the temperature of the cold storage to avoid food spoilage. Therefore, the load cell sensor and DHT22 are very suitable for controlling the amount of food as well as the ambient temperature of the food storage. In addition, the adapter contributes to the convenience of providing power for the hardware devices to operate effectively.

For the SFoodInventory project, the hardware devices are divided into main boxes and food trays to suit each arrangement of the cold storage. In which, the main box includes an esp32 microcontroller, a Led Matrix display screen. This box is placed where the manager wants, where they can observe the warnings from the Led Matrix. In addition, there are 2 important components of the project that must be placed in the cold storage, which are the food trays and the DHT22 sensor. And these 2 components are wired to the main box. In the food trays include load cell sensors and HX711 modules to measure the food weight of the tray. And this tray can have many different types such as 1kg, 5kg, 10kg, 20kg, they are placed anywhere in the cold storage because the wire can be changed according to the user's wishes. And there is also a DHT22 module to measure the temperature and humidity of the cold storage. Just like the food tray, it can be extended anywhere in the cold storage with the right position to measure the most accurate temperature and humidity. The figure 12 below shows the structure of the device as well as the selected components.

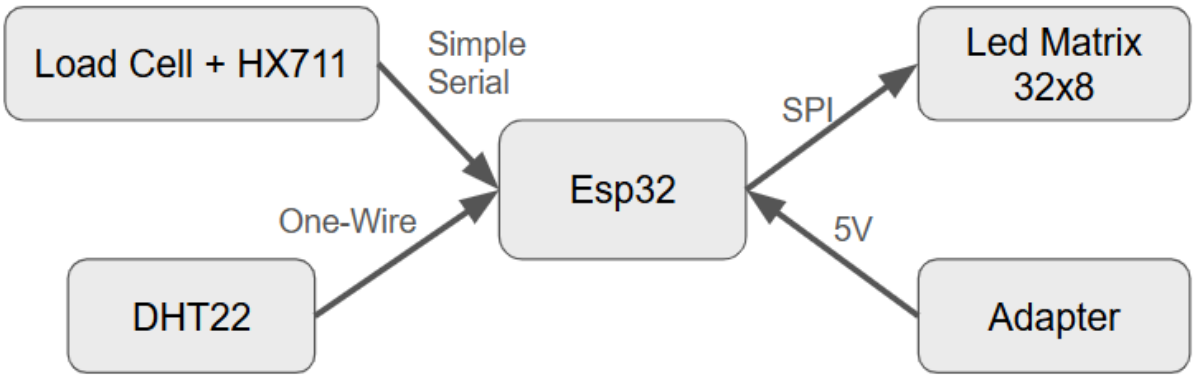


Figure 12: Device Structure

To determine which module has the best quality, reasonable price and ease of use, we conducted some research to evaluate the benefits and limitations of these modules. Finally, the hardware of SFoodInventory is Load cell, DHT22 and Led matrix were selected. Based on the sensors, the main box will collect data, process that data and

transfer them via Websocket protocol to the Server.

Design a scalable hardware architecture, allowing to add more sensors and modules as needed, to meet more complex monitoring requirements and larger inventories.

5.2 Hardware Study

5.2.1 Hardware Components

Weight sensor

Weight sensors are important devices in many measurement applications, especially in the SFoodInventory project to track food volume. And below are the popular types of weight sensors on the market.

Table 4: Types of Weight Sensors

Sensor Type	**Operating Principle**	**Advantages**	**Disadvantages**	**Common Applications**
Load Cell (Strain Gauge)	Measures changes in resistance of strain gauges when force is applied.	High accuracy. Durable, widely used, and cost-effective.	Requires a signal amplifier (e.g., HX711). Needs calibration before use.	Electronic scales, weight monitoring in IoT applications.
Pressure Sensors	Measures the pressure of liquids or gases and converts it to weight values.	Does not require direct contact with the object being measured (in some cases).	Lower accuracy compared to load cells for weight measurement.	Monitoring liquid levels, tank weighing.
Piezoelectric Sensors	Converts mechanical force into electrical signals using the piezoelectric effect.	Quick response to force changes.	Poor accuracy for static weight measurement.	Collision force detection, dynamic weighing.
Hydraulic Load Cells	Uses hydraulic pressure changes due to the applied load to calculate	Handles heavy loads, stable in harsh environments.	Bulky and requires high maintenance.	Industrial scales, heavy load measurements

	weight.			.
Magnetostrictive Sensors	Measures force or weight based on changes in magnetic fields caused by applied load.	No physical contact with the load required.	High cost, requires specific operating environments.	Non-contact force measurement, industrial applications.

Reasons for choosing Load Cell:

In the SFoodInventory project, choosing a load cell is highly appropriate due to its key advantages. Load Cell, based on strain gauge technology, offer high accuracy, making them ideal for measuring food weight with minimal error. They integrate easily with microcontrollers like ESP32 through signal amplifiers such as the HX711, which converts analog signals to digital, ensuring seamless compatibility with IoT systems for transmitting weight data to servers. Load Cells are also cost-effective compared to other sensors like magnetostrictive or piezoelectric types, making them suitable for small to medium-scale projects. Additionally, load cells are durable and reliable, capable of stable long-term operation under typical kitchen or storage conditions. Their versatility further enhances their value, allowing applications in food weighing, inventory monitoring, and alerting when weight thresholds are exceeded.

A similar approach is highlighted in the research presented in the paper "IoT-Based Food Inventory Tracking System for Domestic and Commercial Kitchens".[19] This study demonstrates the use of load cell sensors to monitor food weight in both domestic and commercial kitchen environments. The paper shows how load cells provide high accuracy in tracking food volume and seamlessly integrate with IoT systems, making them an ideal choice for real-time inventory management in food-related applications.

Table 5: Comparison of Weight Sensors

Criteria	**Load Cell**	**Pressure Sensors**	**Piezoelectric Sensors**	**Hydraulic Load Cells**	**Magnetostrictive Sensors**
Accuracy	High	Medium	Low for static weight	Medium	High
Durability	High	High	Low	Very High	High

IoT Integration	Easy	Limited	Limited	Limited	Limited
Cost	Affordable	Medium	High	High	Very High
Best Applications	Food scales, electronic scales	Liquid level monitoring	Collision force detection	Industrial load measurement	Non-contact force measurement

In conclusion, the SFoodInventory project leverages load cells for their high accuracy, cost-effectiveness, and ease of integration with IoT systems like ESP32. Their durability and versatility make them an optimal choice for reliably monitoring food weight, managing inventory, and providing timely alerts, ensuring the system operates efficiently and meets the project's objectives.

Temperature sensor:

In food management systems such as SFoodInventory, maintaining temperature and humidity within the optimal range is a core factor to ensure food quality and safety. Temperature not only directly affects the storage time but also helps prevent the risk of spoilage and bacterial growth. Therefore, choosing the right temperature sensor, ensuring high accuracy, stability and the ability to integrate with IoT systems, is extremely important to meet strict monitoring requirements and ensure the efficiency of the system.

Table 6: Types of Temperature Sensors

Sensor Type	**Operating Principle**	**Advantages**	**Disadvantages**	**Common Applications**
DHT11/ DHT22	Uses a thermistor and an integrated microcontroller to measure temperature and humidity.	Easy to use, affordable (DHT11). DHT22 offers higher accuracy and a wider temperature range.	Slower response time compared to other sensors. Suitable only for applications requiring moderate accuracy.	Monitoring temperature and humidity indoors, simple IoT applications.
DS18B20	Uses a digital 1-Wire interface to transmit temperature data.	High accuracy, allows multiple sensors to connect on a	Requires more complex programming for 1-Wire	Measuring temperature in liquids, cold storage,

		single wire.	communication.	industrial applications.
BME280	Based on MEMS technology, it measures temperature, humidity, and atmospheric pressure.	Multifunctional (temperature, humidity, pressure), high accuracy.	More expensive compared to single-purpose temperature sensors.	Environmental monitoring indoors, high-end IoT devices.
LM35	Uses an analog voltage signal for temperature measurement.	Simple, easy-to-read analog output.	Requires an ADC if used with microcontrollers lacking analog inputs.	Monitoring ambient temperature, industrial devices.
Thermocouple	Measures temperature using the thermoelectric effect of two dissimilar metals in contact.	Extremely wide temperature range, works well in harsh environments.	Requires signal amplification, accuracy depends on calibration.	Industrial applications, high-temperature measurements in furnaces, engines, and harsh environments.

Reasons for choosing DHT22:

The DHT22 sensor is a logical choice due to its key advantages. With higher accuracy (±0.5°C) and a broader temperature range (-20°C to +80°C) compared to the DHT11, it ensures precise temperature data suitable for food storage requirements. The DHT22 is easy to integrate with ESP32 through simple GPIO communication, requiring no complex circuitry, and is supported by readily available libraries, reducing development time. Additionally, it measures both temperature and humidity, a critical factor for food preservation, eliminating the need for multiple sensors. Its reasonable cost, compared to multifunctional sensors like the BME280, makes it ideal for medium-sized projects. Moreover, the DHT22 is well-suited for standard kitchen or storage environments, without needing the extreme durability required by sensors like thermocouples.

A study titled "An IoT-based Real-time Intelligent Monitoring and Notification System of Cold Storage" [20] highlights the application of the DHT22 sensor for

monitoring temperature and humidity in cold storage environments, ensuring optimal conditions for food preservation. This system not only uses the DHT22 for accurate temperature and humidity measurements but also integrates it with an IoT network to provide real-time monitoring and notifications to administrators, further ensuring food safety and preventing spoilage. The findings from this research align with the requirements of the SFoodInventory project, demonstrating the DHT22's suitability in practical food storage applications.

Table 7: Comparison of Temperature Sensors

Criteria	**DHT22/DHT11**	**Pressure Sensors**	**Piezoelectric Sensors**	**Hydraulic Load Cells**	**Magnetostrictive Sensors**
Accuracy	Moderate (DHT11), high (DHT22)	High	High	Moderate	Moderate
Temperature Range	-20°C to +80°C (DHT22), 0°C to +50°C (DHT11)	-55°C to +125°C	-40°C to +85°C	-55°C to +150°C	Very wide (-200°C to +1250°C)
Response Speed	Slow	Fast	Fast	Moderate	Slow (requires signal processing)
Communication Complexity	Low	Moderate (1-Wire)	High (I2C/SPI)	Low (analog)	High (signal amplification)
Cost	Affordable (DHT11), reasonable (DHT22)	Reasonable	High	Affordable	High
Best Applications	Simple IoT, indoor environments	Industrial, environmental monitoring	High-end IoT, climate monitoring	Simple ambient monitoring	Industrial, harsh environments

DHT22 is the right choice for the SFoodInventory project thanks to its ability to accurately measure temperature and humidity, easy integration with ESP32, reasonable price, and suitable for usage conditions. Compared with other types of sensors, DHT22 has a good balance between performance and cost, fully meeting the requirements of food management systems.

Visual Display Hardware:

In food management systems like SFoodInventory, displaying messages clearly and comprehensively is an important factor for users to quickly grasp information. Notifications such as food expiration warnings, environmental status, or food inventory levels need to be conveyed visually and easily readable in all conditions. Therefore, choosing the right type of display, ensuring brightness, accuracy, and the ability to integrate with IoT systems, is an indispensable step to optimise the system's operational efficiency.

Table 8: Types of Display Screens

Display Type	**Key Features**	**Advantages**	**Disadvantages**	**Common Applications**
LED Matrix	Uses LEDs arranged in a matrix to display simple characters or images.	Affordable, easy to extend by chaining multiple modules.	Limited to displaying basic characters and simple images.	Signboards, device status notifications, displaying basic information.
LCD 16x2/20 x4	Liquid crystal display that supports static or basic scrolling text and numbers.	Easy to use, low power consumption.	Does not support complex graphics or dynamic content.	Measurement devices, ATMs, compact IoT systems.
OLED	Uses organic light-emitting diodes, providing crisp displays and basic graphics support.	High brightness, clear visibility even in low-light conditions.	Higher cost compared to LCD and LED Matrix.	Wearable devices, IoT applications requiring simple graphics.
TFT LCD	A colored liquid crystal display supporting detailed graphic	Capable of displaying complex graphics and	High power consumption, expensive.	IoT control screens, devices requiring

	rendering.	supporting user interfaces.		aesthetically pleasing interfaces.
E-Ink	Uses electronic ink technology, resembling printed paper.	Extremely low power consumption, easy to read under bright light.	Slow refresh rate, unsuitable for dynamic displays.	E-readers, fixed status notification boards.

Reasons for choosing LED Matrix 32x8:

LED Matrix is highly suitable for several reasons. It effectively displays essential information such as alerts for expiring or expired food and environmental status like temperature and humidity, with its 32x8 resolution offering sufficient space for simple and readable text. Compared to alternatives like OLED or TFT LCD, the LED Matrix is more cost-effective, making it ideal for medium-scale projects with limited budgets. Its moderate power consumption ensures continuous operation in warehouse environments without requiring high-capacity power sources. Additionally, the LED Matrix is easy to expand by chaining modules, and its compatibility with widely available ESP32 libraries simplifies programming and reduces development time. With high brightness, it provides excellent visibility even from a distance, making it well-suited for warehouses or kitchens where clear notifications are critical.

Table 9: Comparison of Display Screens

Criteria	**LED Matrix**	**LCD 16x2/20x4**	**OLED**	**TFT LCD**	**E-Ink**
Display Complexity	Low	Moderate	High	Very High	Low
Graphic Capabilities	Limited	None	Basic	Full	None
Brightness	Very High	Low	High	High	Not bright
Power Consumption	Moderate	Low	Low	High	Very Low
Cost	Affordable	Affordable	Moderate	Expensive	Moderate

Best Applications	Basic notifications	Small measurement devices	IoT with simple graphics	User interfaces	Fixed status updates

LED Matrix 32x8 is the optimal choice for the SFoodInventory project thanks to its simple information display, low cost, moderate energy consumption, and easy integration with IoT systems. This type of display fully meets the requirements for status and warning notifications, while ensuring cost-effectiveness and flexibility in deployment.

5.2.2 Center Control Unit

The project aims to develop an IoT system that helps manage food in terms of expiration date, quantity and preservation. At the same time, analyze and calculate food consumption needs. Therefore, ESP32 was chosen as the main microcontroller for this development project. Because The ESP32 is a low-cost, low-power system on a chip series of microcontrollers with Wi-Fi and Bluetooth capabilities and a highly integrated structure powered by a dual-core Tensilica Xtensa LX6 microprocessor.[21] Its versatility makes it a go-to choice for a wide range of IoT applications, from smart home devices to industrial sensors. [22].

Table 10: CCU comparison

	ESP8266	**ESP32**	**Arduino Uno (ATmega328P)**	**Raspberry Pi (Model B)**	**STM32 (STM32F4)**
Announcement Date	2014	2016	2010	2019	2011
Main processor	32-bit Tensilica L106	Dual-core 32-bit Xtensa LX6	8-bit AVR (ATmega328P)	Quad-core ARM Cortex-A72	32-bit ARM Cortex-M4
SRAM	160KB	520 KB	2 KB	2 GB - 8 GB (LPDDR4)	192 KB - 256 KB
ROM	64 KB	448 KB (boot ROM)	1 KB (bootloader)	N/A (uses storage)	512 KB
JTAG	X	✓	X	✓	✓
Cache	16 KB	16 KB (I/D cache)	X	1 MB L2 Cache	Optional

WiFi	802.11 b/g/n	802.11 b/g/n/ac	X	Optional (via USB dongle)	X
Bluetooth	X	BLE + Classic	X	BLE + Classic	Optional (via module)
Ethernet	X	Optional (via PHY)	X	✓	Optional
SPI	1	4	1	1	3
I2C	1	2	1	1	3
I2S	1	2	X	1	2
UART	2 (one TX only)	3	1	4	2+

Advantages of ESP32 over other MCUs

- Comparison with Arduino Uno: Arduino Uno does not have WiFi and Bluetooth, requiring additional external modules, increasing costs and complicating the design. The 8-bit AVR processor is not powerful enough to handle IoT tasks and data analysis at the same time as the ESP32.
- Comparison with Raspberry Pi 4: Raspberry Pi 4 has high performance but consumes a lot of power, is more expensive, and requires complex software. ESP32 is low cost and is more optimized for energy-saving IoT applications.
- Comparison with STM32: STM32 is strong in signal control capabilities, but WiFi and Bluetooth implementation requires external modules. ESP32 integrates WiFi and Bluetooth, simplifying the design and reducing costs.
- Comparison with ESP8266: ESP32 has added Bluetooth and Dual-core processor, more powerful than ESP8266 in complex applications.

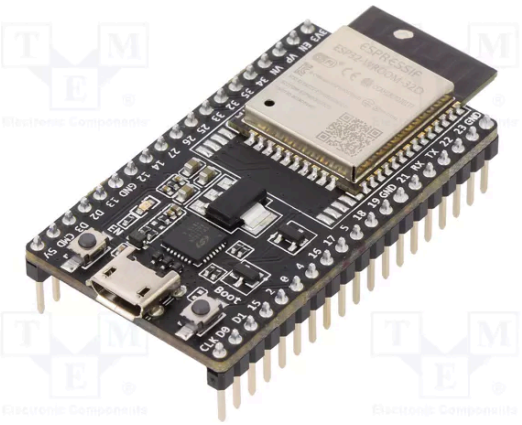


Figure 13: ESP-32 ESP-WROOM-32D

ESP32 stands out thanks to its integrated WiFi, Bluetooth, powerful processor,

and support for multiple communication protocols, fully meeting the requirements of IoT applications. Compared with other MCUs, ESP32 is the optimal choice due to its high performance, flexibility, and cost-effectiveness, which helps to simplify system design and meet project requirements well.

Table 11: SFoodInventory Requirement

SFoodInventory Requirement	**Feature-equipped ESP32**
Wireless connectivity (WiFi, Bluetooth)	Supports WiFi 802.11 b/g/n/ac and Bluetooth BLE + Classic, ideal for IoT applications.
Sensor data processing	Dual-core processor and 520 KB RAM ensure high performance for real-time tasks.
Expiration date management and notifications	Built-in timers and protocols support automatic notifications.
Integration with temperature and humidity sensors	I2C and SPI interfaces compatible with popular sensors like DHT22.
Data analysis and statistics	Sufficient hardware resources to handle on-device data analysis algorithms.
Low power consumption	Low-power mode suitable for continuous system operation.

ESP32 is an ideal choice for IoT Application Used in Food Inventory Management project due to its integration of many important features, high performance, and reasonable cost. Compared with other MCUs, ESP32 helps to optimise the design and ensure the system operates effectively under real conditions.

ESP32 Development Boards

ESP32 development boards are versatile tools for IoT applications, offering integrated WiFi, Bluetooth, and multiple interfaces (UART, SPI, I2C, etc.). They feature dual-core processors, low power consumption, and compatibility with platforms like Arduino IDE and MicroPython. Their small form factor and cost-effectiveness make them ideal for rapid prototyping and deployment.

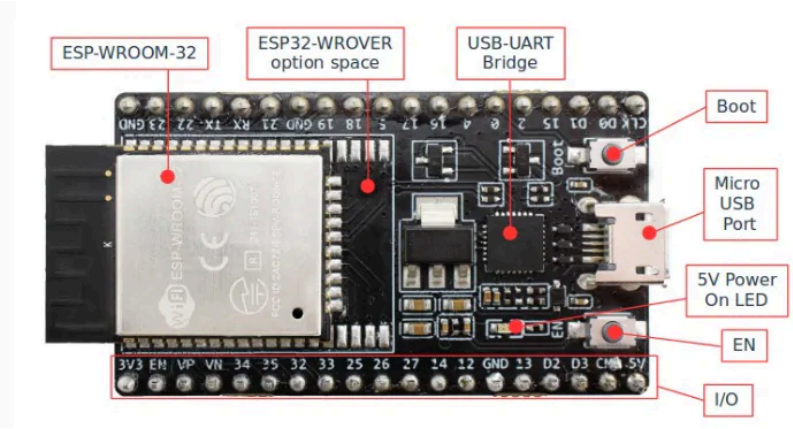


Figure 14: DOIT Esp32 DevKit v1

DOIT ESP32 DevKit

The DOIT Esp32 DevKit v1 is one of the development boards created by DOIT to evaluate the ESP-WROOM-32 module. It is based on the ESP32 microcontroller that has support for Wifi, Bluetooth, Ethernet, and Low power, all in one chip.[23]. Key features include:

- Wireless Connectivity: Built-in WiFi and Bluetooth.
- Rich Interfaces: Supports GPIO, ADC, DAC, UART, SPI, I2C, and PWM.
- Compact and Efficient: Operates at 3.3V with a USB connection and includes low-power modes.

Specifications:

- Processor: Dual-core Xtensa LX6.
- Memory: 520 KB SRAM, 4MB Flash.
- Interfaces: 3x UART, 4x SPI, 2x I2C, 16x ADC, 2x DAC.

Use Cases: Widely used in smart home automation, environmental monitoring, wearable devices, and industrial IoT, the DOIT ESP32 DevKit is reliable, user-friendly, and ideal for IoT development.

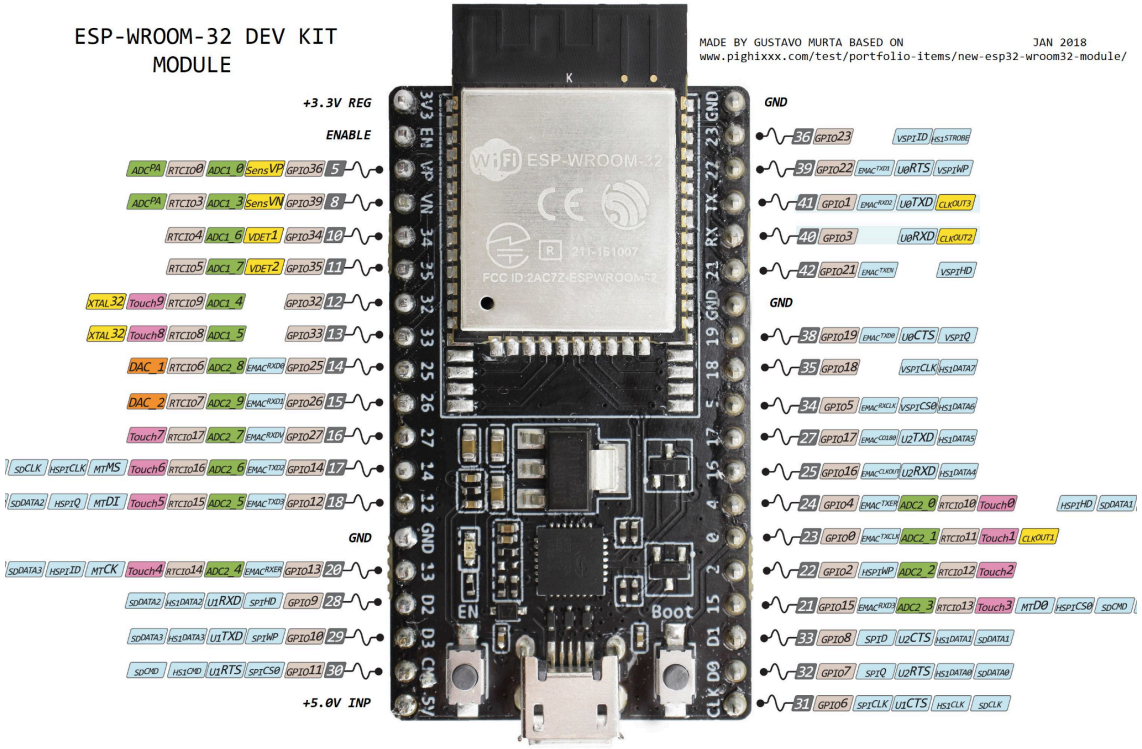


Figure 15: DOIT Esp32 DevKit pinout diagram

5.2.3 Battery

The SFoodInventory project uses ESP32, devices such as load cell, LED matrix, DHT22, and WebSocket protocol to communicate with the server. Therefore, the system requires a stable power source with enough capacity to ensure continuous operation of the device and avoid interruption.

Table 12: Compare power supplies for projects

Power Supply Method	**Stability**	**Power Output**	**Suitability for the Project**	**Reason**
Adapter 5 volt	High	Sufficient	Very suitable	Ensures stable power for ESP32 and peripherals (LED matrix, load cell).
Rechargeable Battery (Lithium)	Medium	Potentially sufficient	Suitable as backup	Can be used during power outages but requires charging and protection circuits.
Solar Power	Medium	Potentially sufficient	Less suitable	Depends on sunlight conditions, unstable without backup batteries.

Alkaline Batteries	Low	Insufficient	Not suitable	Low capacity, cannot meet the power needs of ESP32 and peripherals.
Computer USB Power	High	Limited	Suitable for testing	Cannot provide sufficient power for high-consumption devices in the system.

The adapter 5 volt is the current optimal choice for the SFoodInventory project, ensuring stable power supply, easy deployment, and sufficient capacity for the entire system.

- Stability: The adapter provides a constant and stable current (5V, usually 2A or more), which meets the requirements of ESP32 and devices such as LED matrix or sensor.
- Power response: LED matrix and load cell consume a lot of power, and the adapter can provide enough power without additional supporting devices.
- Ease of implementation: The adapter is popular, easy to find, and does not require complex circuit design like rechargeable batteries or solar energy.

5.3 Principle Electrical Circuit and Assembly

5.3.1 Principle electrical circuit

The main box consists of 8 main components: ESP32, load cell, DHT22, led matrix, button, adapter 5V.

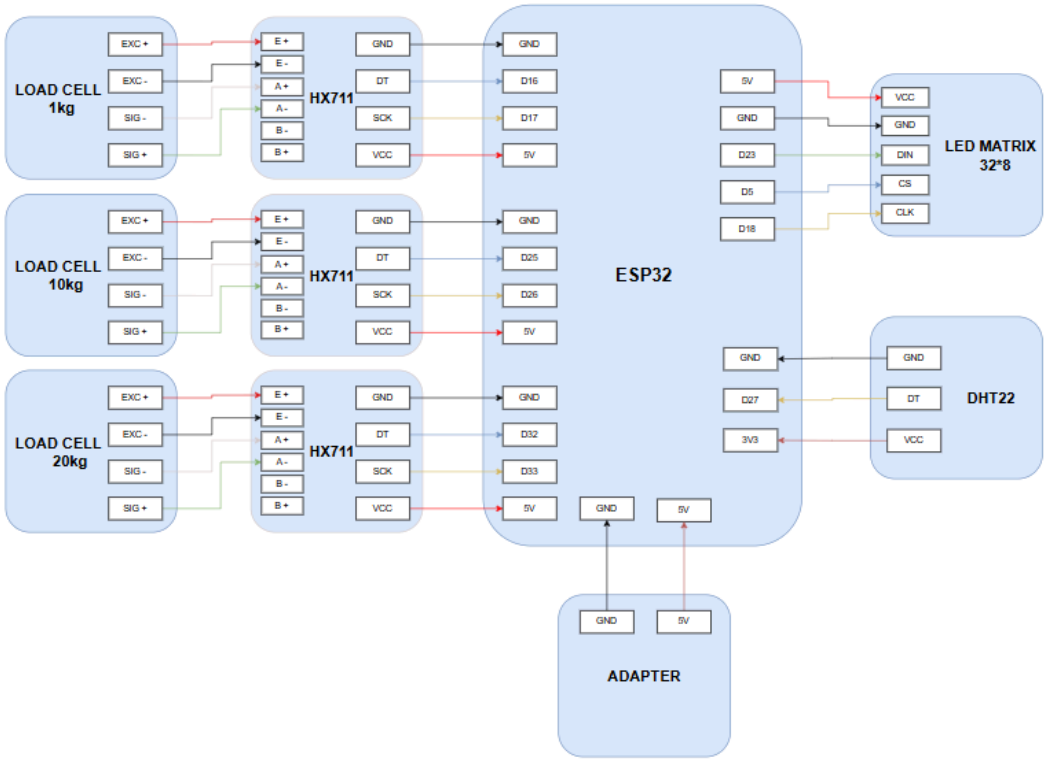


Figure 16: Principle electrical circuit

The project's wiring system design needs to ensure aesthetics, safety and performance. ESP32 and LED matrix are placed in the same box to reduce wire length, increase stability and keep the system neat. The connecting wires are carefully fixed to avoid tangles and save space.

The load cell is located in a separate box, connected to HX711 with a high-quality 4-core signal wire, threaded through a drilled hole with a rubber gasket to prevent dust and moisture. HX711 connects to ESP32 with a 4-core wire and a 4-pin SM jack, making it easy to disassemble for maintenance.

The DHT22 sensor is placed outside, connected to ESP32 through a drilled hole protected by a rubber gasket, preventing dust and water from entering. The power adapter for ESP32 is plugged through a dedicated hole, which helps to fix and protect the power connection, ensuring the power wire is separated from the signal wire to avoid interference.

This design helps the system operate stably, neatly, easy to maintain and increase the durability of hardware components.

5.3.2 Assembly

The following are the packaged prototypes. The prototype was constructed on a plastic box to facilitate experimentation and ease of usage.

Figure 17: Hardware model image

The project has packaged all the hardware components in a sealed box to protect them from external factors such as water, dust, and humidity, ensuring their safety and long-term preservation. This sealed box design not only protects the electronic components from environmental impacts, but also maintains the stability of the system under all operating conditions. Careful storage by sealing reduces the risk of damage, short circuits due to humidity, and protects the electrical connections from negative impacts. This also helps prolong the life of the components and ensures the system operates at peak performance for a long time.

5.4 Gateway in SFoodInventory

The Gateway in the SFoodInventory system plays an important role as a bridge between IoT (Internet of Things) devices and the central management system. It is responsible for collecting data from sensors and terminals, pre-processing this data before transmitting it to the server for storage and analysis.

First, ESP32 acts as a gateway, it receives data from load cells and DHT22. Then, calculates the remaining food amount, and predicts the expiration date based on the collected data. When the data is obtained, the WebSocket protocol is used to transmit real-time data to the server. And the Gateway receives commands from the central system to adjust the configuration or perform necessary operations on the IoT device.

The Gateway is not simply a data transmission device, but also the brain that processes and manages information between IoT devices and the central management system. Designing and implementing an effective Gateway will ensure the system operates stably, securely and is highly scalable, contributing to improving the efficiency of food warehouse management.

5.5 Protocol in SFoodInventory

The table below will help clarify why WebSocket, HTTPS, and BLE protocols are used in the SFoodInventory project.

Table 13: Protocols used in SFoodInventory

Protocol	**Advantages**	**Suitability for the project**	**Reason for Use**
WebSocket	- Continuous two-way communication. - Real-time data transmission. - Low latency, instant response.	Suitable for real-time notifications and data updates.	Real-time updates to provide users with instant information on changes in the warehouse or food status.

HTTPS	- High security (SSL/TLS). - Stable connection, easy to implement. - Ensures security during transmission.	Suitable for transmitting data between IoT devices and the backend.	Ensures secure transmission of sensitive information regarding food status and warehouse conditions.
BLE	- Energy-efficient. - Short-range, low-latency communication. - Supports multiple devices simultaneously.	Suitable for nearby IoT sensors like load cells and environmental sensors in the warehouse.	Saves energy and maintains stable connectivity within the warehouse without draining battery life.

The SFoodInventory project uses WebSocket, HTTPS, and BLE protocols to provide real-time updates, ensure data security, and save energy. These protocols support an efficient and flexible IoT system for monitoring and managing food inventory.

5.6 Firmware

Main function of SFoodInventory

- Read sensor value from weight sensors and temperature sensor: The sensors will be initialised after checking conditions.
- Send message to server through gateway: Prepare message with the weight of the load cells, temperature and humidity of the DHT22 to send to server along with the companyID to know where the data comes from.
- Push data to the mobile app to display information: the mobile app will display the actual data

5.6.1 Flow Chart

The Flow Chart provides an overview of the entire firmware operation process in the SFoodInventory system. The process begins with the system booting, after which the ESP32 connects to the configured WiFi network and establishes a WebSocket connection with the central server. Next, the Load Cell and DHT22 sensors are initialized to start collecting data. The firmware continues by collecting data from the sensors, pre-processing it by filtering and cleaning the data, and then sending the processed data to the server via WebSocket. At the same time, information about temperature and humidity is displayed on the LED matrix screen. Finally, the firmware receives control commands from the server to adjust the system configuration if necessary, ensuring that the system always operates efficiently and stably.

Below is the flowchart diagram of hardware in figure 18.

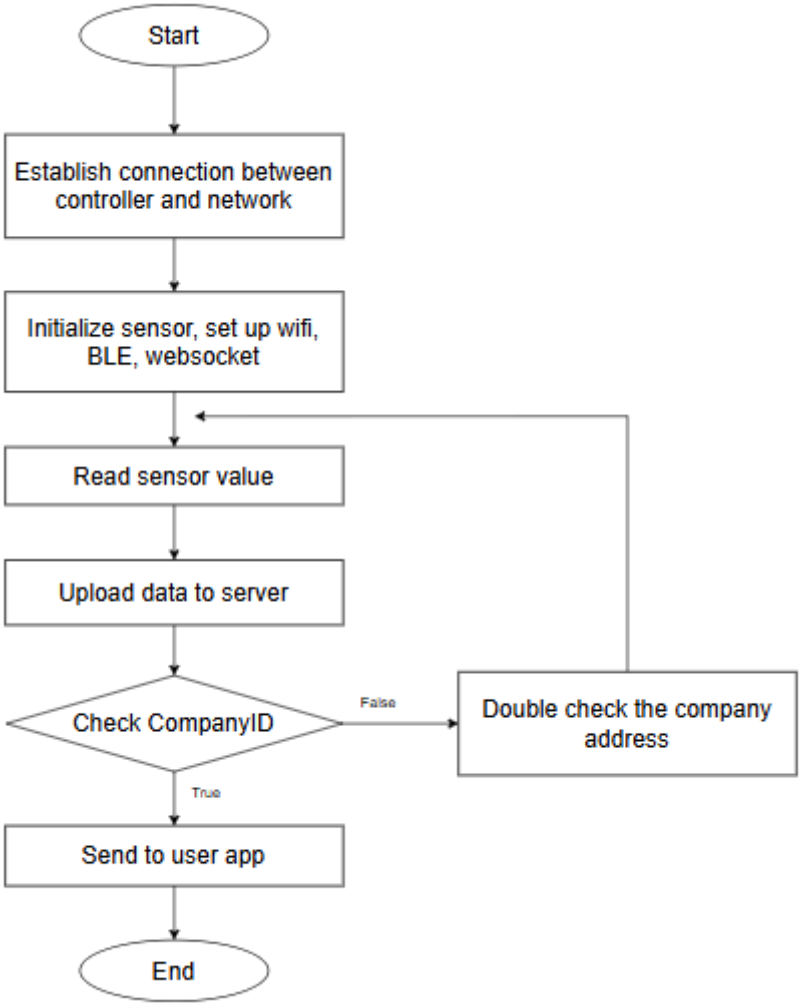


Figure 18: Main functions flowchart

5.6.2 Checking data from sensor task

Below is the flowchart diagram of checking data from sensor tasks in figure 19. The process starts by initiating the data check, then the firmware proceeds to read data from the Load Cells and DHT22 sensor to get information about weight, temperature and humidity. After collecting the data, the firmware checks the validity of the data by verifying that there are invalid values. If the data is valid, the firmware continues processing by calculating the amount of food based on the measured weight and storing the processed data or error information for use in subsequent tasks. If the data is invalid, the firmware logs the error and reports it via Serial, ensuring that only correct data is used in the system.

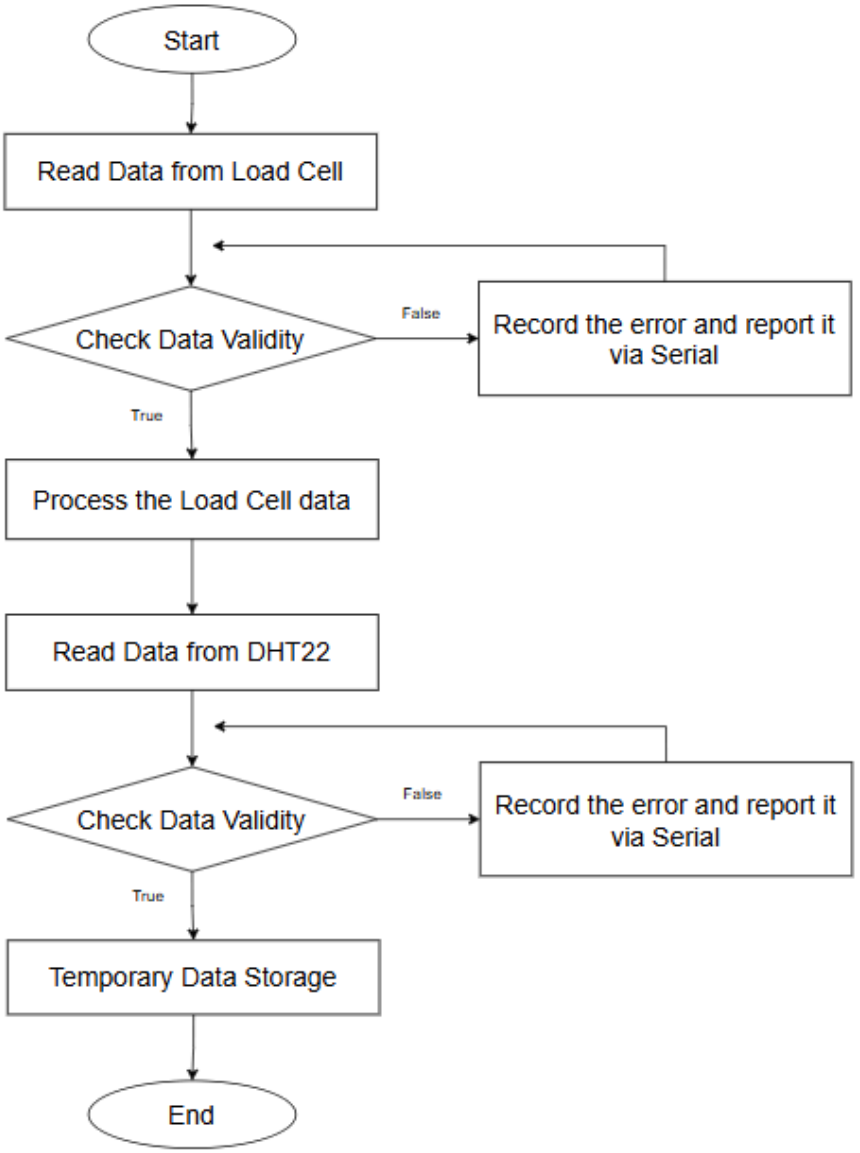


Figure 19: Checking data flowchart

5.6.3 Sending data to server task

Below is the flowchart diagram of sending data to server tasks in Figure 20 . The process starts by generating a JSON string from the weight and sensor data using the ArduinoJson library. The firmware then checks the WebSocket connection to ensure that the ESP32 has successfully connected to the server. If the connection is established, the firmware sends the JSON data to the WebSocket server. After sending the data, the firmware checks the response from the server to determine whether the data was received successfully. If successful, the firmware waits for a specified amount of time (e.g. 2 seconds) before sending the data again. If unsuccessful, the firmware attempts to reconnect to the WebSocket server. This process is repeated continuously to ensure that the data is always updated to the server efficiently and reliably.

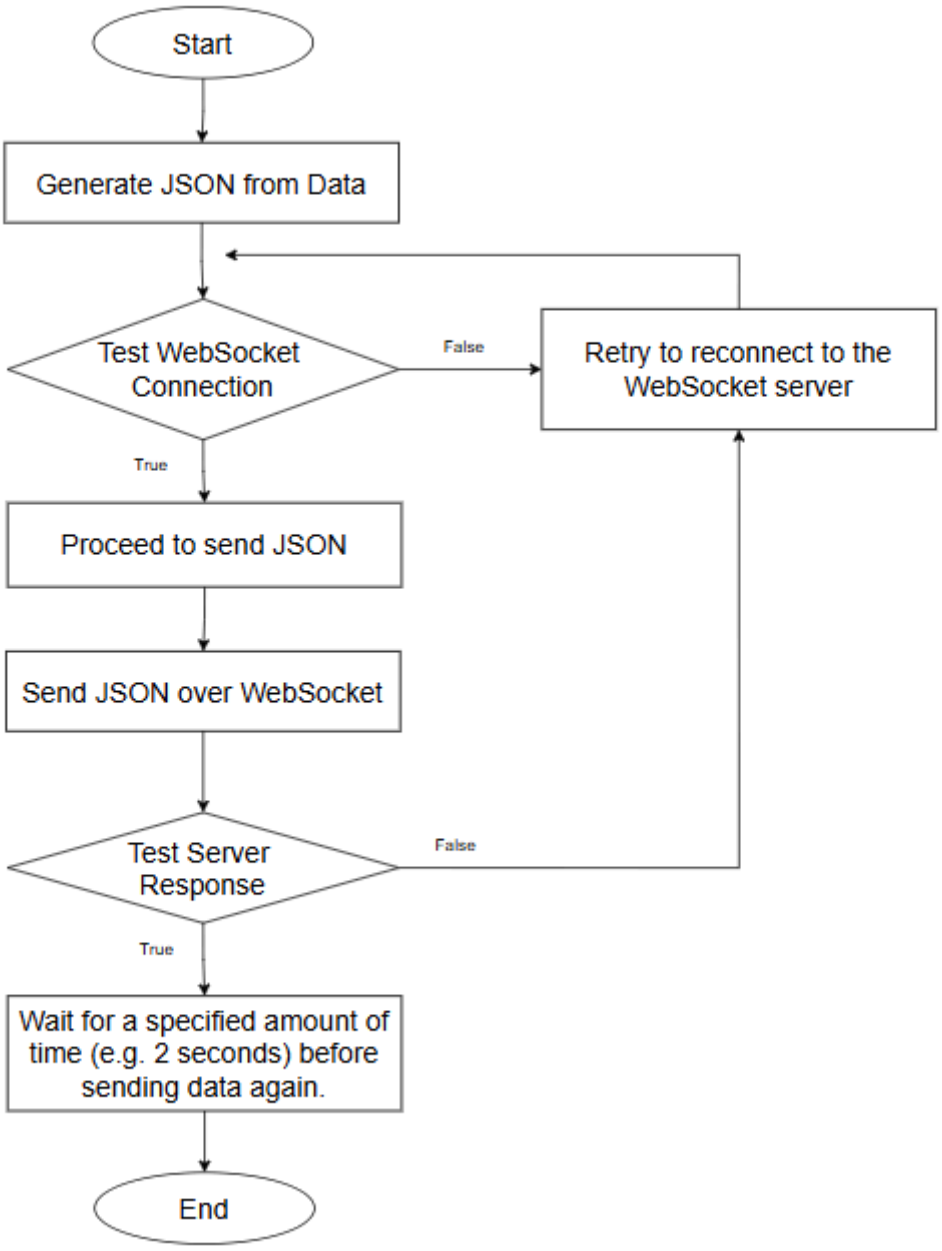


Figure 20: Sending data to server flowchart

5.6.4 Showing real data task

Below is the flowchart diagram showing real data tasks in Figure 21. The process starts by initiating the data display on the LED screen. The firmware reads the temperature and humidity values from the DHT22 sensor, then formats this data into appropriate character strings using the printf function. The temperature data (e.g. "25.3°C") is displayed on the LED matrix, then the system pauses for a specified period of time (e.g. 2 seconds) before continuing. Next, the humidity data (e.g. "60.5%") is formatted and displayed on the LED screen, then pauses again before repeating the process. This ensures that the temperature and humidity information is constantly

updated and easily visible to the user, allowing them to effectively monitor the food preservation status.

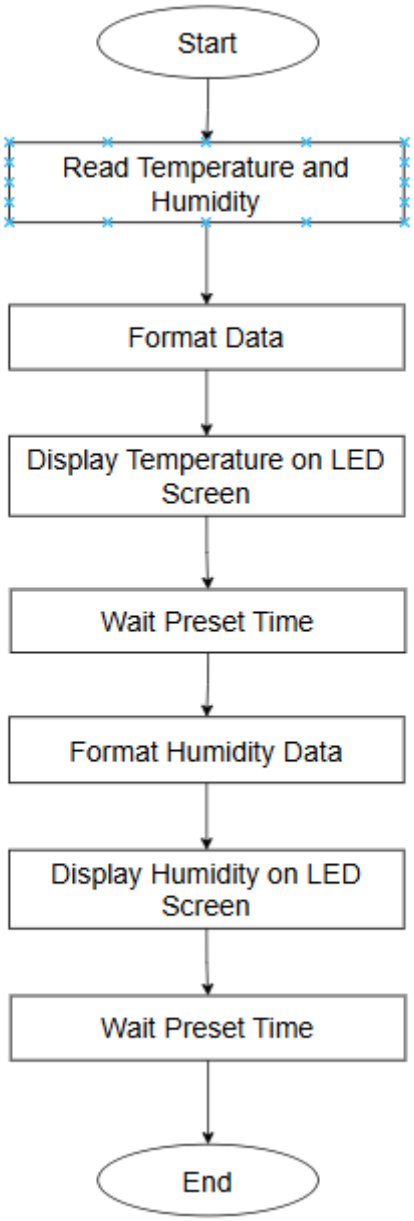


Figure 21: Showing real data flowchart

6 Database

6.1 Overview

The database plays a crucial role in the Food Inventory Management System Based on IoT, being responsible for storing and managing all essential data for the system. Based on the project's requirements, Azure SQL Server was chosen as the database platform due to its scalability, security, and smooth integration with the Spring Boot backend [24]. The database serves not only as a storage space but also ensures synchronization and real-time data retrieval [25].

With automatic scaling capabilities and built-in disaster recovery, Azure SQL Server offers a reliable and efficient solution that can handle the system's growing data needs. Additionally, the pay-as-you-go pricing model of Azure optimizes costs, providing a balanced trade-off between functionality and cost for this IoT-based application [26].

The database architecture facilitates easy communication between the Spring Boot backend and the system, ensuring real-time data storage and updates. This is critical for accurately managing food inventory, monitoring environmental conditions, and sending timely notifications to users [27].

Table 14: On-Demand Pricing

Provider	Service Name	Pricing model	Estimated Cost (Hourly)
Microsoft Azure	Azure SQL Database	vCore-based (Serverless or Provisioned)	~$0.37/hour
AWS	Amazon RDS for SQL Server	On-Demand Instances (Single-AZ)	~$0.504/hour
Google Cloud	Cloud SQL for SQL Server	Per-use pricing (Enterprise Editions)	~$0.5712/hour

6.2 Azure SQL Server

Azure SQL Server was selected after evaluating various database options based on critical factors for the success of the project. The auto-scaling capabilities of Azure SQL Server ensure that the system can handle increasing data as IoT devices provide real-time data. Its seamless integration with the Spring Boot framework simplifies

development and enhances the communication between the server and the database. Additionally, Azure SQL Server provides high availability and reliability through built-in failover and disaster recovery features, ensuring minimal downtime [28]. Furthermore, Azure's pay-as-you-go pricing model is a cost-effective choice for this IoT application, providing a good balance between functionality and cost. In this system, the database serves a supporting but essential role in efficiently organizing and storing data. It maintains records of the food inventory, including names, quantities, expiration dates, and food types, ensuring accurate tracking of the inventory. User information, including roles and permissions, is securely stored to ensure proper access control. Environmental sensor data, such as temperature and humidity, is logged to monitor storage conditions, while notifications are recorded to provide timely information to users. Although the database does not perform complex analytics or processing, it ensures that data is readily available for real-time server and mobile app operations. By choosing Azure SQL Server, the database meets the reliability, scalability, and security requirements of the project. Its integration with other system components ensures seamless data storage and transfer, making it an indispensable part of the Food Inventory Management System Based on IoT.

Figure 22: Azure Microsoft SQL Server

6.3 Database Schema

6.3.1 Device

Purpose**:** Manages information about IoT devices used in the system.

Key Columns	**Function**

id	Unique identifier for each device.
name	Name of the device.
company_id	Links the device to a specific company.
mac_address	MAC address of the device for network identification.
created_at, updated_at, deleted_at	Tracks when the device was created, updated, or deleted.

6.3.2 Company

Purpose: Stores information about companies using the system.

Key Columns	**Function**
id	Unique identifier for each company.
name	Name of the company.
created_at, updated_at, deleted_at	Tracks the creation, update, and deletion times of the company record.

6.3.3 Food

Purpose: Manages information about different types of food.

Key Columns	**Function**

id	Unique identifier for each food item.
name	Name of the food.
expired_date	Expiration date of the food.
category_id	Links the food to a category in the FoodCategory table.
created_at, updated_at, deleted_at	Tracks when the food record was created, updated, or deleted.

6.3.4 FoodCategory

Purpose: Categorizes food into different types.

Key Columns	**Function**
id	Unique identifier for each food category.
category_name	Name of the category (e.g., Vegetables, Meat, Seafood).
created_at, updated_at, deleted_at	Tracks creation, update, and deletion times for the category.

6.3.5 FoodItem

Purpose: Stores detailed information about specific food items.

Key Columns	Function
food_id	Links to the Food table to specify the food.
quantity	Quantity of the food item.
expiration_date	Specific expiration date for the food item.
last_checked_expired_date	The last date when the expiration was checked.
created_at, updated_at, deleted_at	Tracks when the food item record was created, updated, or deleted.

6.3.6 TemperatureHumidity

Purpose: Monitors temperature and humidity data for a specific company.

Key Columns	Function
id	Unique identifier for the temperature and humidity record.
company_id	Links the data to a company.
temperature, humidity	Recorded temperature and humidity values.
created_at	Timestamp when the data was recorded.

6.3.7 Notification

Purpose: Manages notifications sent to users or companies.

Key Columns	**Function**
id	Unique identifier for the notification.
message	Content of the notification.
type_notification	Type of the notification (e.g., Temperature Alert, Expiry Alert).
created_at, deleted_at	Tracks creation and deletion times of the notification.
status	Status of the notification (e.g., unread, read).

6.3.8 Users

Purpose: Manages user information within the system.

Key Columns	**Function**
id	Unique identifier for each user.
username, password	User credentials for login.
company_id	Links the user to a company.

role	Defines the user's role (e.g., admin, staff).
created_at, updated_at, deleted_at	Tracks the creation, update, and deletion of user accounts.

6.3.9 Token

Purpose: Manages user authentication sessions.

Key Columns	**Function**
id	Unique identifier for each token.
user_id	Links the token to a user in the Users table.
expired	Expiration time of the token.
created_at, deleted_at	Tracks the creation and deletion of tokens.

Figure 23: Entity Relationship Diagram

7 Mobile Application

7.1 Mobile Application Introduction

Worldwide, mobile application development relies on a variety of programming languages, each supported by large, active communities that help streamline the development process. Among these, Java for Android continues to be one of the most popular and trusted languages for mobile app development.

From the very beginning, Google selected Java as the primary language for Android app development, and it remains integral to the Android ecosystem. With robust community support and an extensive range of tools and libraries, Java allows developers to create powerful, adaptable, and reliable applications for Android devices. According to the 2023 Stack Overflow Developer Survey [29], Java continues to rank among the top 5 most widely used programming languages, especially in the domain of mobile development.

A Statista report from 2023 reveals that 71% of Android apps are built using either Java or Kotlin, with Java holding around 50% of the market share in Android development [30]. This underscores Java's enduring importance and leadership in the mobile app development industry. Moreover, Java provides developers with access to a vast array of libraries and frameworks that accelerate the development process, helping to reduce both time and costs.

Given these benefits, the development team has selected Java for Android as the primary programming language for the mobile application. This choice guarantees high performance, wide compatibility with different versions of Android, and long-term stability throughout the app's development and maintenance life cycle.

7.2 Objectives

The mobile application for the food inventory management system aims to provide real-time monitoring of food storage conditions, including temperature, humidity, and expiration dates, to reduce spoilage and waste. It will enable users to track food weight, monitor stock levels, and receive notifications about low stock or critical conditions. Additionally, the app will offer data analytics to help businesses optimize inventory, improve restocking decisions, and enhance operational efficiency, ultimately supporting sustainability by minimizing food waste.By implementing these features, the mobile application will support the broader goal of optimizing food inventory management, improving operational efficiency, and promoting sustainability by reducing food waste.

7.3 Food Inventory App Features

The Food Inventory Management System mobile application is designed to streamline food inventory tracking, monitor storage conditions, and manage product expiration dates. The system features three primary roles: Managers, Staff, and Administrators. Both Managers and Staff share similar responsibilities, such as monitoring inventory levels, tracking expiration dates, and receiving alerts for low stock or products approaching expiration. They also oversee the storage conditions, with IoT sensors tracking temperature, humidity, and weight. The key difference is that Managers have the added responsibility of managing personnel, including assigning tasks and overseeing team activities. Administrators, on the other hand, focus on system settings and user role management. The app provides real-time updates on inventory status and storage conditions, supporting businesses in maintaining food safety and reducing waste. The context diagram in Figure 24 outlines the external entities and system interfaces.

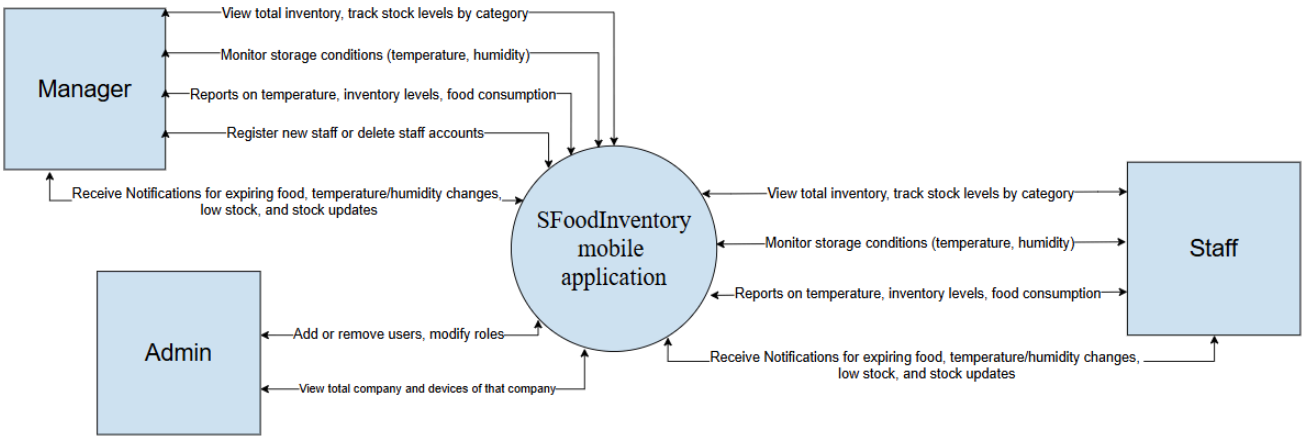


Figure 24: Mobile application system context diagram

7.4 User Requirements

7.4.1 User Roles and Permissions

The system is composed of three primary user roles: Managers, Staff, and Administrators, each with specific responsibilities to ensure smooth operation.

Table 15: User groups

#	Actor	Description
1	Managers	Managers are responsible for overseeing inventory management, setting alerts for low stock or approaching expiration dates, and generating reports on inventory trends and waste. They also manage staff tasks and ensure proper food storage conditions are maintained.

2	Staff	Staff track food stock, update inventory levels, monitor expiration dates, and ensure real-time data is accurately reflected in the system. They also handle alerts related to stock shortages and environmental deviations.
3	Administrators	Administrators configure the system, manage user access and roles, ensure the smooth operation of the application, view the total company and devices of that company, and oversee the overall functionality to maintain efficiency, and user satisfaction.

7.4.2 Use Cases

Diagram

These use cases diagrams illustrate the functions of three different actors: Manager in Figure 25, Staff in Figure 26 and Administrators in Figure 27.

Figure 25: Use cases diagram - Manager use cases

The main functions of the Manager account include logging in with username and password or logging in with Google, logging out, managing staff profiles by viewing, registering new staff, and deleting staff, viewing statistics, receiving notifications, monitoring temperature and humidity, adding devices, viewing food data by viewing real-time weight, editing food tray information, and clearing food tray information, changing the password, and logging out.

Figure 26: Use cases diagram - Staff use cases

The main functions of the Staff account include logging in with username and password or logging in with Google, editing the personal profile, viewing statistics, receiving notifications, viewing food data, adding devices, monitoring temperature and humidity, changing the password, and logging out. For food data management, staff can perform additional actions such as viewing real-time weight, editing food tray information, and clearing tray information.

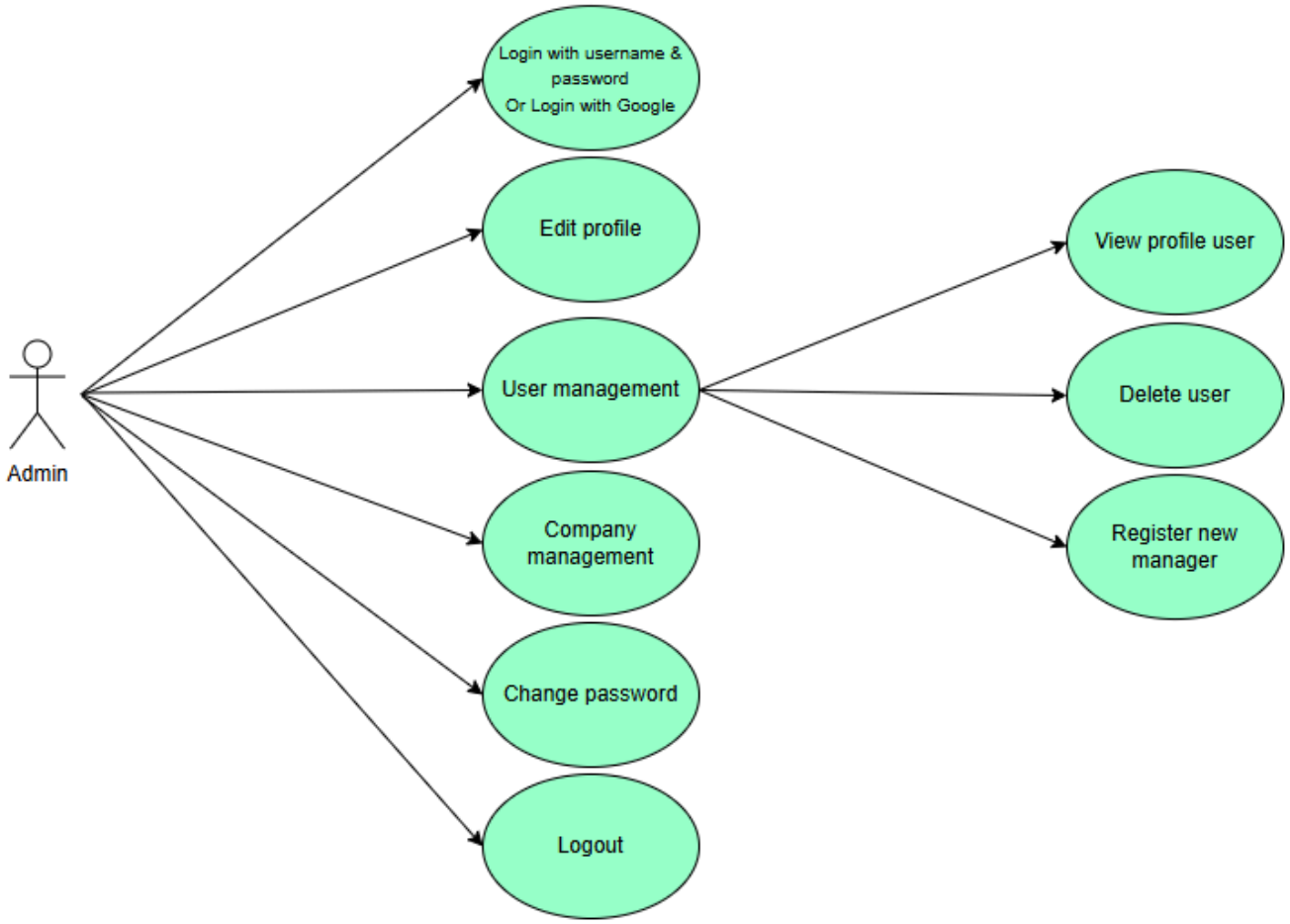


Figure 27: Use cases diagram - Administrators use cases

The main functions of the Admin account include logging in with username and password or logging in with Google, managing user profiles by viewing, registering new user , and deleting user, managing company, changing the password, and logging out.

Description

Detailed descriptions of the functions are presented in Table 16.

Table 16: Function descriptions

ID	Use cases	Actors	Description
1	Login with username and password Or login with Google	Manager, Staff, Admin	Perform authentication for manager, staff, admin
2	Logout	Manager, Staff, Admin	Logout system
3	Edit profile	Manager, Staff, Admin	Allows the user to update personal information such as name, contact details, and other profile settings.
4	Staff management	Manager	Enables the manager to view staff profiles, register new staff, and delete existing staff to ensure efficient team management and operations.
5	View statistic	Manager, Staff	Allows both the manager and staff to view daily statistics, including temperature, humidity, food usage per day, and the amount of food in stock each day. This feature helps track environmental conditions and food inventory, ensuring smooth operations and accurate records.
6	View notifications	Manager, Staff	The feature allows both the manager and staff to receive alerts about food expiration, temperature and humidity issues, low food levels in trays, end-of-day inventory updates, and expiration reminders for newly added food. These notifications help maintain food quality and inventory accuracy.
7	View temperature & humidity	Manager, Staff	The feature allows both the manager and staff to monitor real-time temperature and humidity levels.This helps ensure that environmental conditions meet the required standards for food storage and quality.

8	Change password	Manager, Staff, Admin	Allows the user to update their account password for security purposes.
9	Logout	Manager, Staff, Admin	Allows the user to securely exit their account, ensuring that no unauthorized users can access it after they finish using the system.
10	View profile staff	Manager	Allows the manager to access and review the details of an staff's profile
11	Delete staff	Manager	Allows the manager to remove a staff's account from the system.
12	Register new staff	Manager	Allows the manager to add a new staff to the system by entering their personal Email.
13	View food details	Manager, Staff	Allows both the manager and staff to view specific information about the food trays, including the food name, weight, and expiration date.
14	Edit Food Tray Information	Manager, Staff	Allows the manager and staff to update the details of a food tray, including changing the food name or updating the expiration date.
15	Clear tray information	Manager, Staff	Allows the manager and staff to remove or reset the details of a food tray, such as food name, weight, and expiration date. This ensures that outdated or incorrect data is cleared, maintaining accurate and current inventory records.
16	Company Management & Add Device	Admin	Admin to register a new device for the company by entering its details into the system. This ensures that the new

			device is properly integrated and can be used for monitoring or managing relevant processes within the company's operations.
17	User Management	Admin	Admin to manage user accounts within the system. This includes registering new managers and deleting users as needed.
18	View profile user	Admin	Allows the admin to access and review detailed information about a user's profile, including personal details
19	Delete user	Admin	Allows the admin to remove a user's account from the system. This action is typically performed when a user no longer requires access or has left the organization
20	Register new manager	Admin	Allows the admin to add a new manager to the system by entering the company's name and the manager's email address.

7.5 Non-Functional Requirements

7.5.1 Quality Attributes

Usability

USB-1: The interface is friendly, simple, and easy to use for management purposes.

USB-2: Main functions are placed outside the dashboard for easy and quick access.

USB-3: Links, buttons, and checkboxes are easily clickable.

Performance

PE-1: Any pages in the system shall fully load in at maximum of 30 seconds in normal condition.

Security

SC-1: The user password will be securely hashed using a password encoder function. When a user creates or updates their password, the password is passed through an encoding algorithm. This algorithm generates a hashed version of the password, which is then stored in the database. The hashed password is one-way and cannot be easily reversed, ensuring that user passwords are kept secure.

SC-2: The system supports multiple types of users, each with specific access constraints based on their roles and permissions. Additionally, JWT (JSON Web Token) will be used to securely store and transmit user information. The token allows the system to authenticate and authorize users without exposing sensitive data, providing a secure means of managing user sessions.

Maintainability and extensibility

MTE-1: The source code must adhere to coding conventions to ensure it is easy for developers to read, understand, and maintain or modify as needed.

7.6 System Software Design

The system is a mobile application based on the MVVM (Model-View-ViewModel) architecture, as outlined in Figure 28 . The MVVM pattern divides the application into three core components:

Model: The Model represents the data and business logic of the application. It encapsulates the data and provides methods to access and manipulate it. In mobile applications, the Model typically interacts with databases, web services, or other data sources. It is responsible for managing the application's state and ensuring data integrity and consistency.

View: The View is responsible for displaying the user interface (UI) and presenting the data to users. It captures user interactions and displays the updated UI elements. Views in mobile apps include UI components like buttons, text fields, and images. The View is designed to be intuitive and visually appealing, providing a user-friendly interface.

ViewModel: The ViewModel acts as an intermediary between the Model and the View. It holds the logic to manipulate the data from the Model and formats it for the View. The ViewModel does not directly interact with the UI but instead updates the View through data bindings. It processes user input, manages state, and ensures that the Model is updated based on interactions with the View. In mobile apps, the ViewModel is crucial for separating the UI logic from business logic and ensuring that changes in data are reflected in the View.

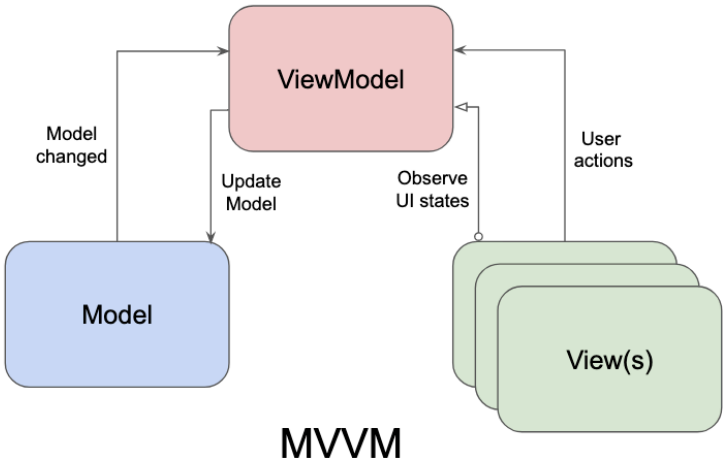


Figure 28: MVVM model

The MVVM architecture is widely used in mobile application development for its distinct advantages in terms of maintainability, scalability, and UI responsiveness. One of the main benefits of MVVM is the clear separation of concerns between the Model, View, and ViewModel. This separation allows the application to be more maintainable and scalable because the user interface can be modified independently from the business logic and data handling logic without negatively impacting the other components. Such a structure facilitates the modification of one layer without disturbing the other, ensuring better code reusability and ease of testing [31].

Another key benefit is the ViewModel's ability to support data binding. By enabling two-way data binding, MVVM simplifies the process of updating the UI in response to changes in the data model. This reduces the need for manual intervention and enhances the dynamism and responsiveness of the application, which is particularly important in modern mobile applications where UI changes are frequent [32]. The ViewModel acts as a bridge between the View and the Model, ensuring that data transformations and UI updates happen automatically, resulting in smoother and more interactive user experiences.

7.7 Detailed Functional Design

7.7.1 Authentication

Login with username and password

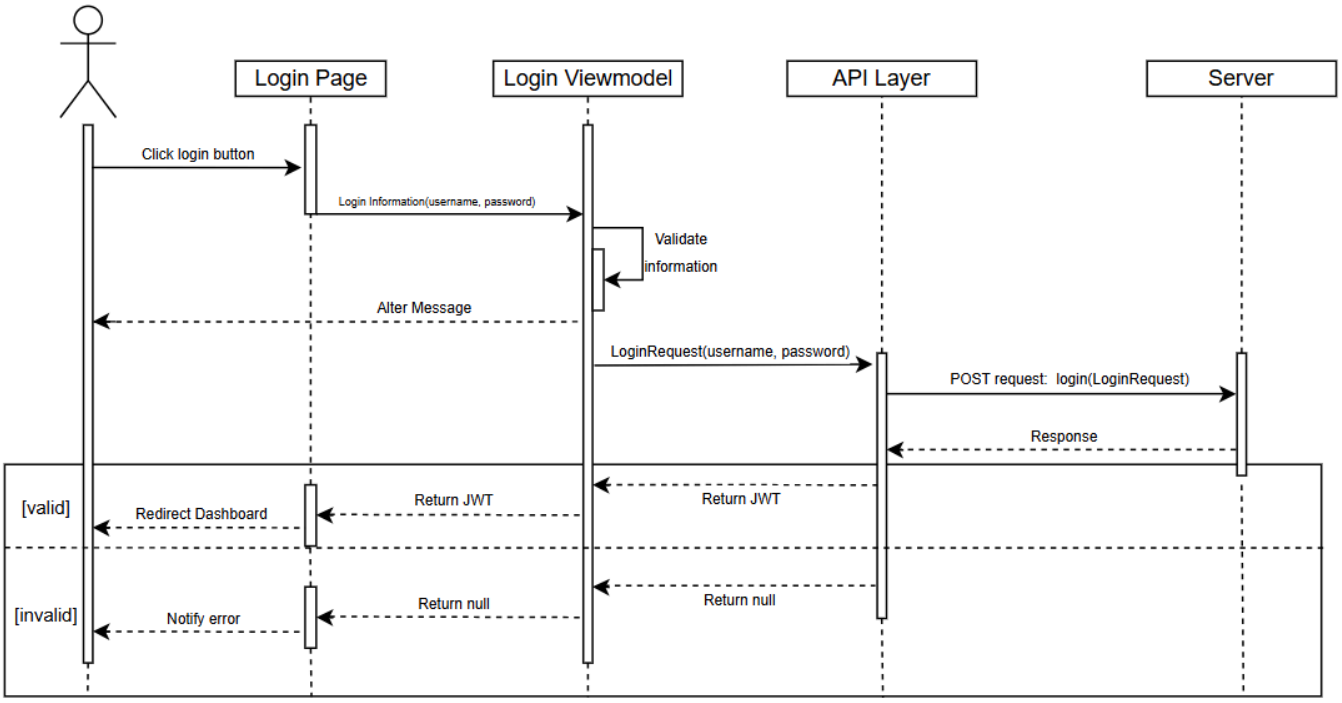


Figure 29: Login with username and password sequence diagram

Login with Google

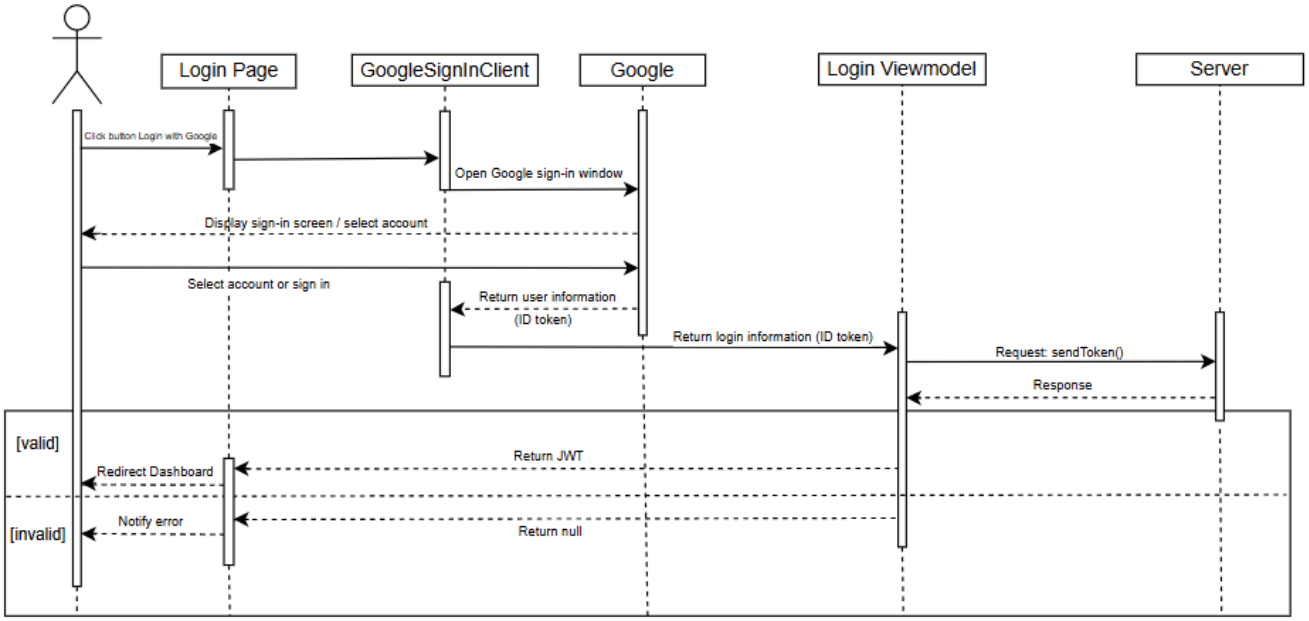


Figure 30: Login with Google sequence diagram

7.7.2 Inventory Management

View food inventory list

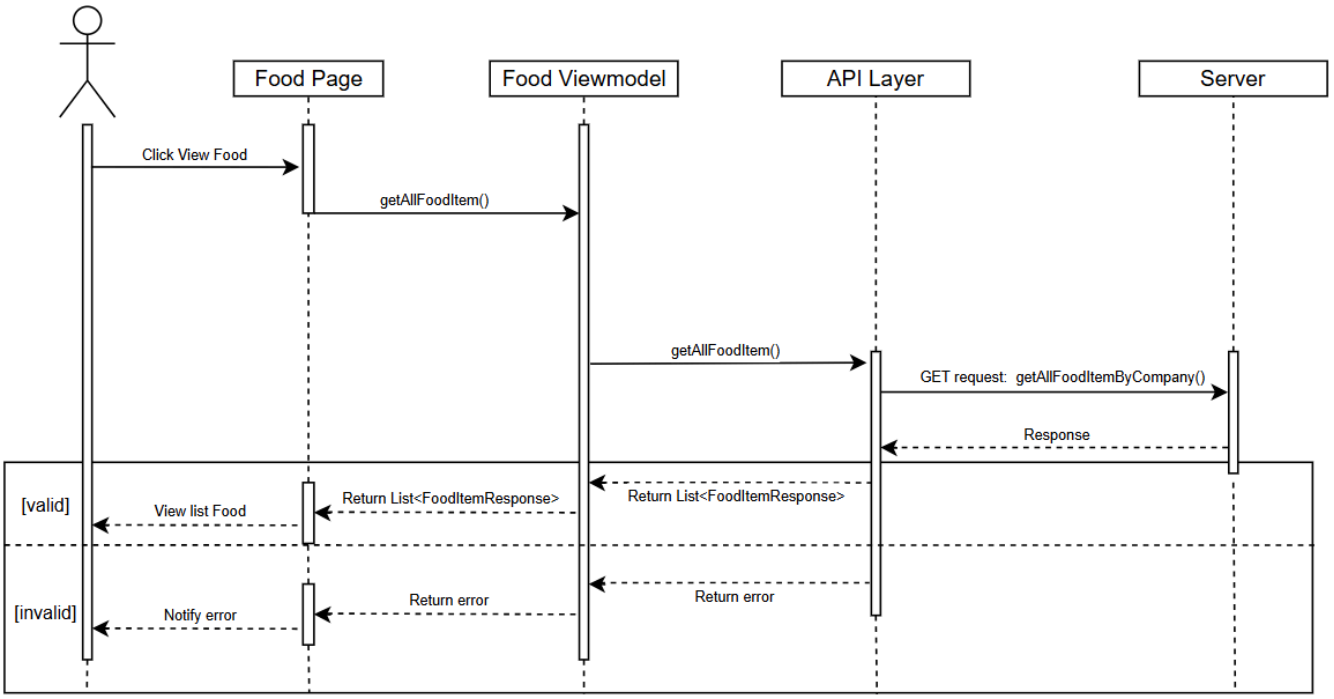


Figure 31: View Food sequence diagram

Real-Time Weight View

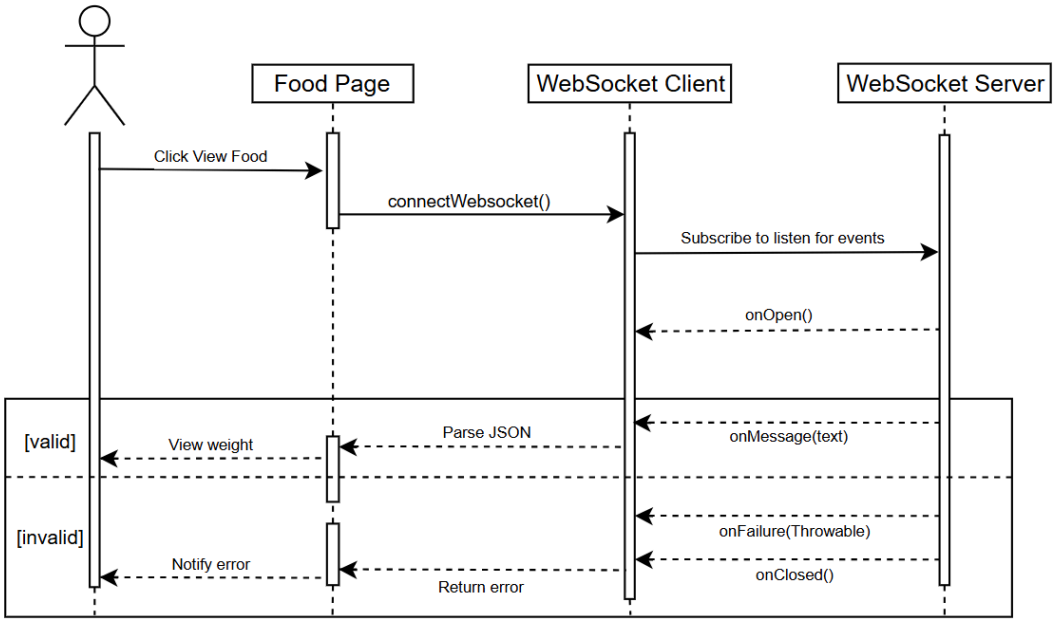


Figure 32: Real-Time Weight View sequence diagram

7.7.3 Notifications and Alerts

Notifications and Alerts

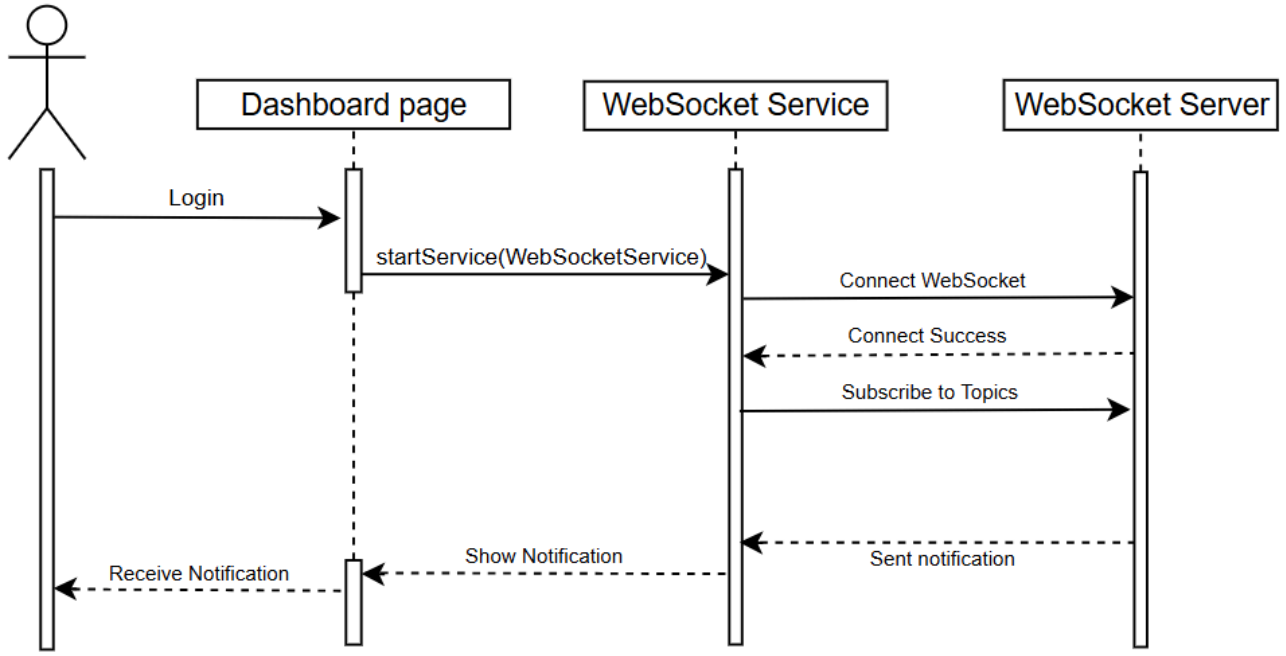


Figure 33: Real-Time notifications and alerts sequence diagram

7.8 Testing

7.8.1 Overview

Manual testing is critical for IoT software testing because some things need a human touch. Only a person can truly judge if a smart home app is easy to use. Manual testers can also spot odd behaviors that machines might miss, like how a device acts when you use it in an unexpected way.[33] In addition, the SFoodInventory project also applies Unit Testing and Integration Testing methods to ensure the quality and stability of the system. Unit Testing helps test each small component of the software independently, ensuring that each module functions properly. Integration Testing combines modules together to test the interactions between them, ensuring that the components work harmoniously and do not cause errors when integrated.

In the project, the team conducts two testing phases: Unit Testing and Integration Testing. The specific description of these two testing phases is explained in Table ??? below.

Table 17: Table phase testing

Testing Methods	**Advantages**
Unit Testing	- Ensures that each module functions properly. - Easily identifies and fixes errors. - Increases the reliability of the source code.
Integration Testing	- Checks the interactions between modules. - Detects problems that arise when components work together. - Ensures that the system operates seamlessly.

The combination of Unit Testing and Integration Testing helps ensure that the SFoodInventory system not only works correctly in each individual part, but also works effectively when these parts are integrated together. This contributes to improving product quality, minimizing errors and enhancing user experience.

7.8.2 Test Cases

Unit testing

Unit testing is the process of testing the smallest functional unit of code. Software testing helps ensure code quality and is an integral part of software development. It is a software development best practice to write software in small functional units and then write unit tests for each unit of code.[34] Unit testing is cost-effective, time-saving, and easy to maintain, giving developers a sense of security.

Detail Unit test will be described in IOP490 G2 UnitTestReport file

Integration testing

Integration testing, also known as integration and testing (I&T), is a type of software testing where various units, modules, or components of a software application are tested together as a unified system.[35] This type of testing ensures that all parts of the application work together seamlessly and that data is accurately transmitted and displayed on the user's mobile application interface. By focusing on the integration points, this testing phase helps identify and resolve any issues that may arise from the interaction between various system components, thereby ensuring the overall stability and reliability of the application.

Detail Unit test will be described in IOP490 G2 IntegrationTestReport file

8 Deliverables

8.1 Overview

The SFoodInventory system consists of three main components: hardware, software, and data management infrastructure. The software includes a mobile application, a cloud-based server. After an intensive period of research and development, the development team created the final version of the mobile application and built the server infrastructure to support the system. The team also set up a robust database to store and manage inventory data, ensuring real-time updates and efficient data retrieval. The entire SFoodInventory system is managed through an online server, which facilitates seamless operation and allows the team to gather valuable insights through usage analytics and system performance metrics.

8.2 Hardware

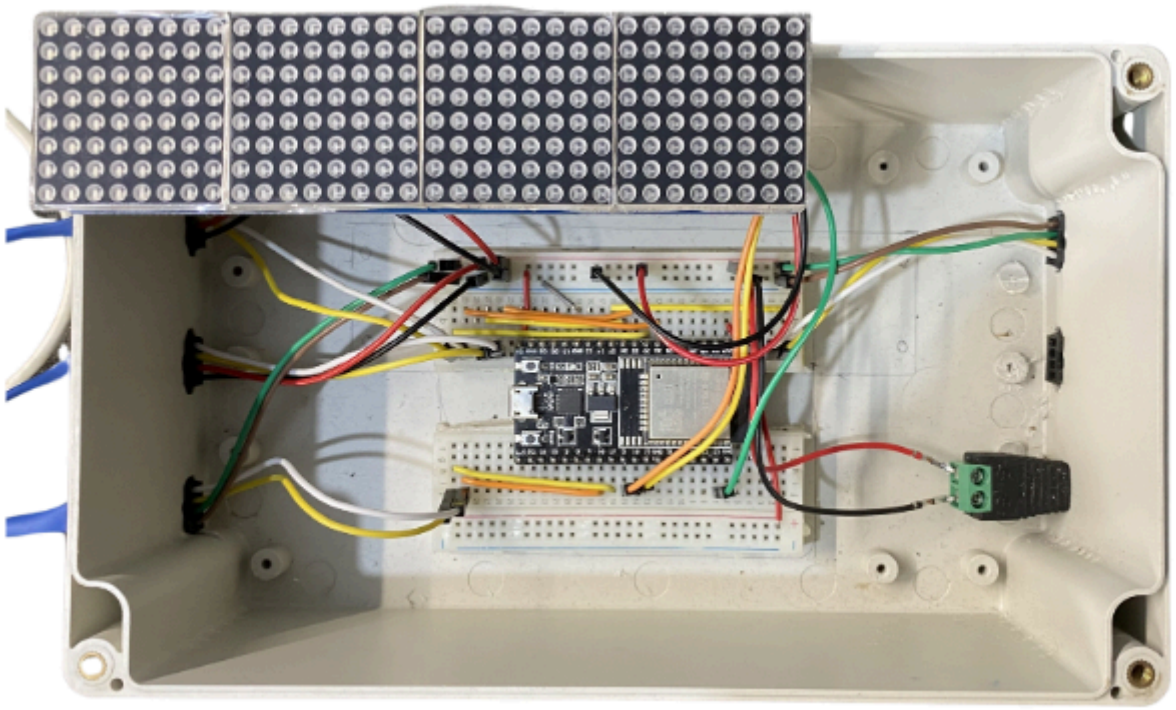


Figure 34: Image of main box

About the main box, all equipment is also packed in a plastic box for protection as well as aesthetic compatibility. All wires are run under a circuit board and sensors are connected via jumper pins or manually designed SM jacks. Disassembly is also more convenient. A button is added for more convenient error handling.

Figure 35: Image of load cells and DHT22 sensor

Regarding these two sensors, they are arranged in the cold storage, anywhere suitable for each different restaurant's cold storage. For Load Cell, it has been packaged in a plastic box with a tray on top, turning it into a smart food tray that shows the actual amount of food. There is only 1 wire coming out with a neat SM Jack, easy to disassemble or move position. As for DHT22, the project is not framed in a box, because each restaurant has a different location design, so we need to survey in advance to be able to place DHT22 appropriately.

8.3 Mobile Application Deliverables

8.3.1 User login with username and password

Role: All roles

When the user opens the application, the login screen will appear. The user clicks the Login button to start using the app. The login process requires entering a username and password. If the user enters an incorrect username or password, an error message will be displayed as shown in Figure 36.

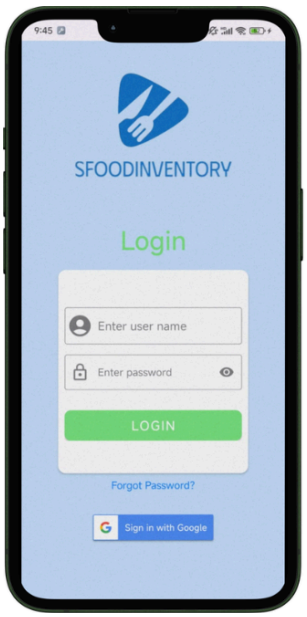


Figure 36: Login with username and password screen

8.3.2 User login with Google

Role: All roles

Login with Google allows users to quickly and securely log into the application using their Google account credentials. By selecting the "Login with Google" option, users are redirected to a Google authentication page, where they can grant permission for the app to access their account information. This method eliminates the need to remember separate usernames and passwords, providing a seamless and efficient login experience. It also ensures a high level of security, as Google's authentication system includes robust protection features like two-factor authentication and encryption. Displayed as shown in Figure 37.

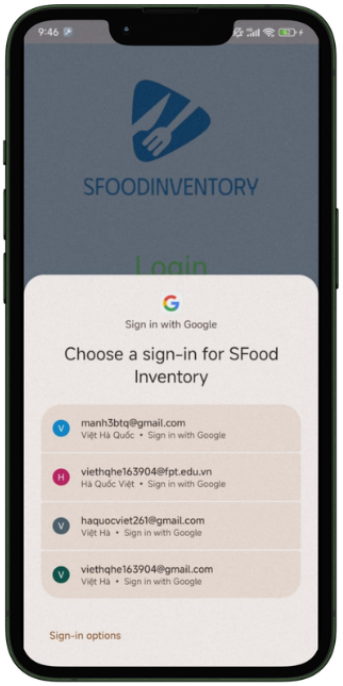
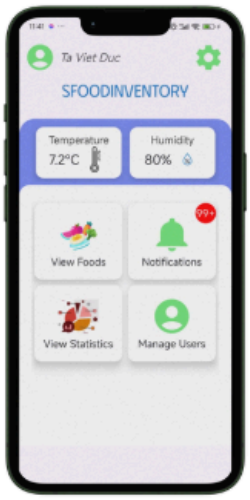


Figure 37: Login with Google screen

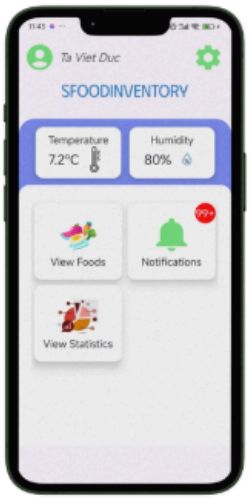
8.3.3 Dashboard

Role: All roles

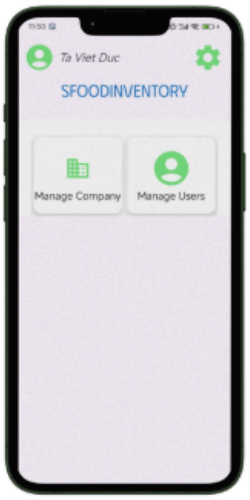
After clicking the LOGIN button, if authentication is successful, the main screen will display the Dashboard. Depending on the user's role, the Dashboard screen will vary: the Manager Dashboard screen, the Staff Dashboard screen, and the Admin Dashboard screen as shown in Figure 38.



Manager dashboard screen



Staff dashboard screen



Admin dashboard screen

Figure 38: Dashboard screen

8.3.4 View and edit profile

Role: All roles

The Edit Profile screen allows users to update their personal information within the application. Here, users can modify details such as their name, email, profile picture, password, and other relevant settings. The interface typically includes fields for each editable piece of information, along with buttons to save changes or cancel the updates. This screen provides a user-friendly way to keep profile information up to date, ensuring a personalized and secure experience within the app. Displayed as shown in Figure 39.

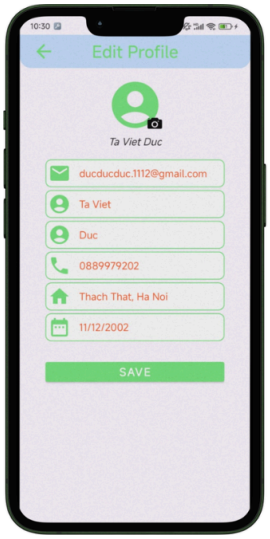
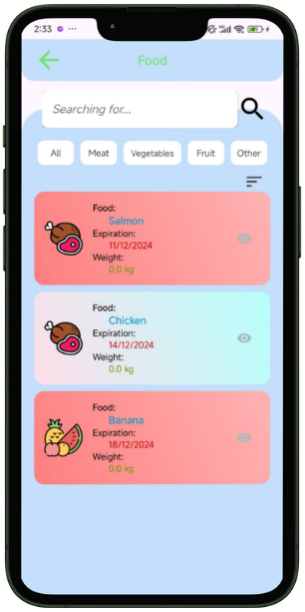
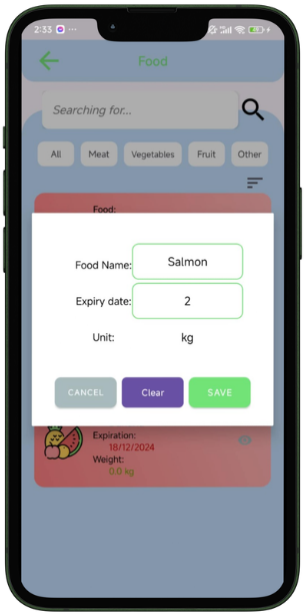


Figure 39: Edit profile screen

8.3.5 View Foods

Role: Manager and Staff

The View Foods screen displays a list of food items available in the application. Each food item is shown with key details, including its name, expiration date, and actual weight. This screen allows users to easily browse and access important information about the food items, helping them keep track of expiration dates and manage inventory efficiently. The layout is designed for quick scanning, ensuring users can identify the relevant details at a glance. Users may also be able to sort or filter the list based on expiration date or weight for better organization. Displayed as shown in Figure 40.	 Figure 40: View Foods screen

When a user taps the View button next to a food item, they can edit the name and expiration date of that item. The application provides a list of available foods along with their corresponding expiration dates. This feature helps users update food details to ensure the information remains accurate and up to date, ultimately making it easier to track and manage their food supply effectively. Displayed as shown in Figure 41.	 Figure 41: Add Food screen

8.3.6 Notifications

Role: Manager and Staff

The Notification Screen provides users with essential alerts and updates regarding their food inventory and environmental conditions. It includes various types of notifications, such as alerts for food items that are nearing or have passed their expiration dates, warnings about temperature and humidity levels falling outside acceptable ranges, and notifications when the actual weight of a food item is too low. Additionally, the screen offers end-of-day stock reports summarizing the inventory status and provides expiration warnings when new food items are added, ensuring users are informed of any critical issues. These notifications help users maintain optimal food management by keeping them updated on inventory levels, food quality, and storage conditions. Displayed as shown in Figure 42.

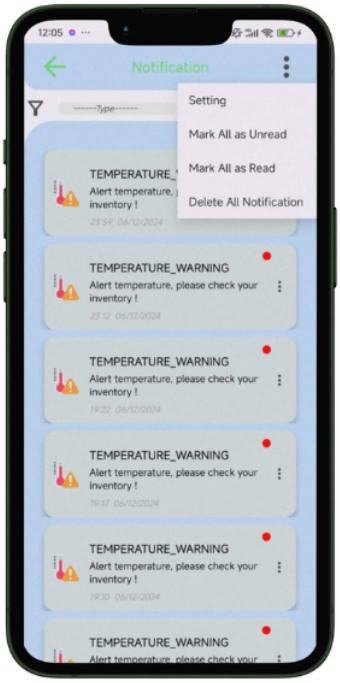


Figure 42: Notifications screen

8.3.7 View Statistics

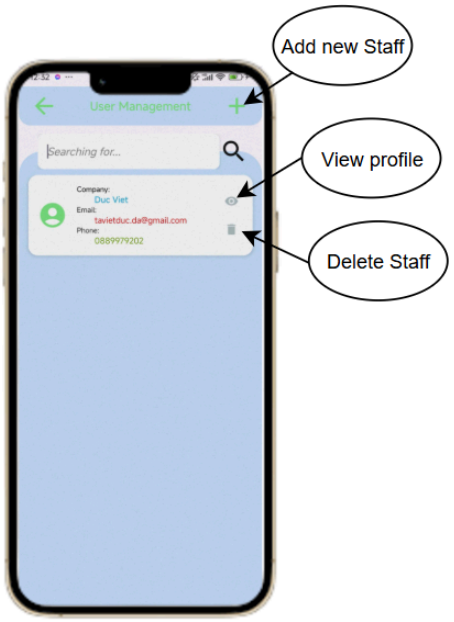
Role: Manager and Staff

The Statistic Screen provides users with detailed charts and reports to analyze various aspects of their food inventory and storage conditions. It includes visual representations of temperature and humidity levels over time, allowing users to track environmental conditions and identify any potential issues. Additionally, the screen presents statistics on the food items used on specific days, offering insights into consumption patterns. It also displays the end-of-day inventory levels, providing a snapshot of the remaining stock for each food item. Lastly, the screen includes an analysis of notifications, helping users understand trends or recurring issues related to food expiration, stock levels, or environmental factors. This screen is designed to provide a comprehensive overview, enabling users to make informed decisions about inventory management and storage practices. Displayed as shown in Figure 43.

Figure 43: Statistics screen

8.3.8 Manager Users

Role: Manager and Administrator

The Manager Users screen is designed for managers to view and manage staff information within the company (Figure 44). It displays a list of staff, showcasing key details such as their name and other relevant information.	 Figure 44: User Management Screen

Managers have the ability to delete a staff member by clicking the Delete button next to the staff's entry. Additionally, the screen allows managers to add new staff by pressing the Add button, which opens a form for entering the new staff member's details. This functionality provides a simple yet effective way for managers to maintain an up-to-date roster of staff and manage the workforce efficiently. Displayed as shown in Figure 45.

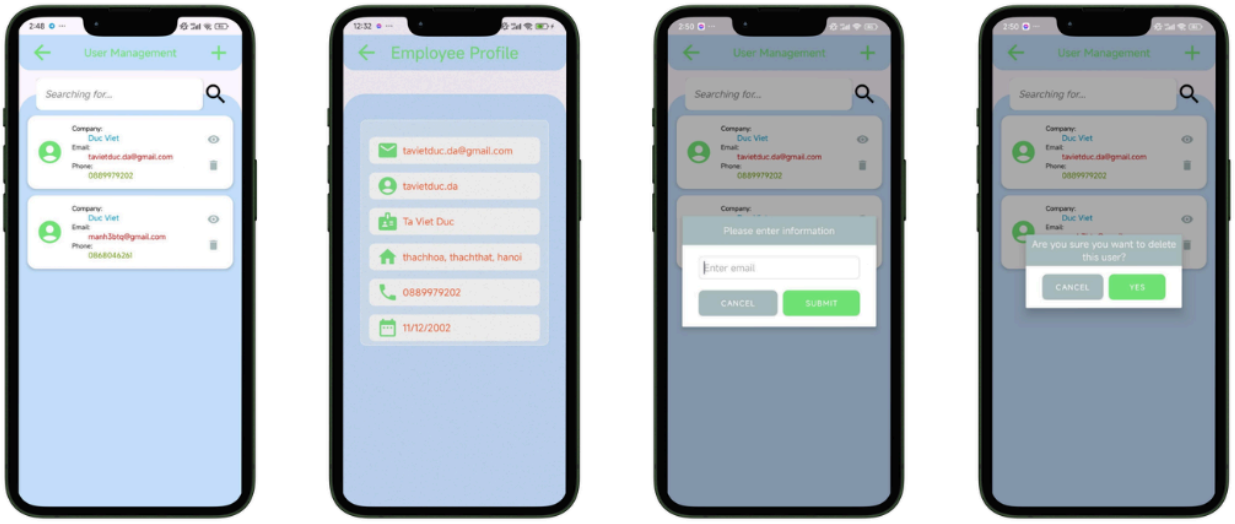


Figure 45: CRUD Staff Screen

8.3.9 Manager company

Role: Administrator

The Manager Company screen for admins displays a list of all companies along with the number of devices associated with each company. Admins can add devices to a company by clicking the Add button next to the company. After clicking the button, the admin will scan for available Bluetooth devices. Once the device is detected, the admin will enter the SSID and Wi-Fi password to configure the device. This functionality enables efficient device management and network configuration for each company. Displayed as shown in Figure 46.

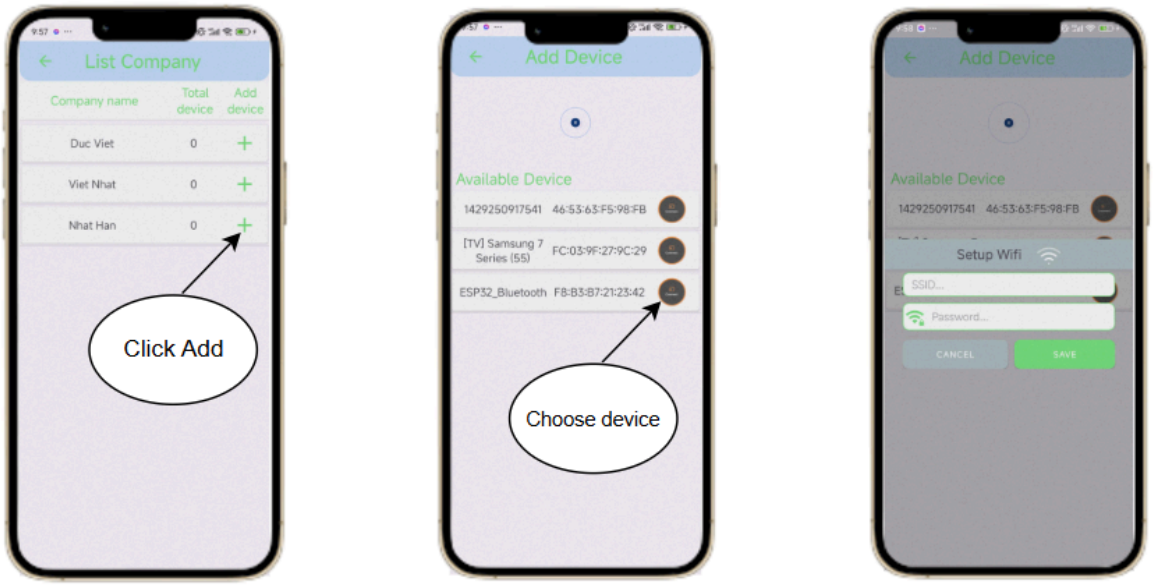
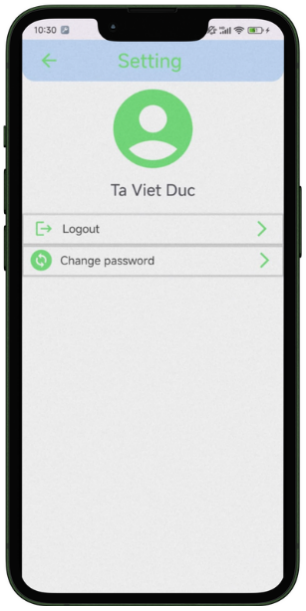


Figure 46: Manager company Screen

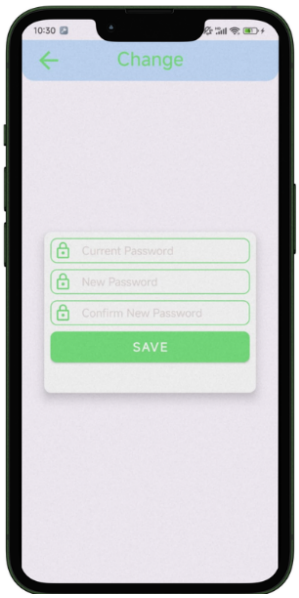
8.3.10 Setting

Role: All roles

The Setting screen allows users to manage their account preferences. It provides the option to change the password, enabling users to update their login credentials for security purposes. Additionally, the screen includes a Logout button, allowing users to sign out of their account when they are finished using the application. This screen is designed to give users control over their account settings, ensuring ease of access and security. Displayed as shown in Figure 47.	 Figure 47: Setting Screen

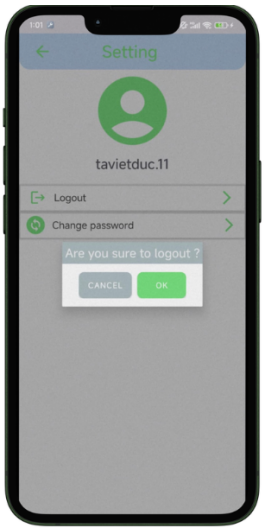
8.3.11 Change password

Role: All roles

The Change Password screen allows users to update their current password for enhanced security. To change the password, users are required to enter their current password, followed by the new password they wish to set, and then confirm the new password. Once the details are submitted, the system will verify the information and update the password if all requirements are met. This feature helps ensure that users can keep their accounts secure by regularly updating their credentials. Displayed as shown in Figure 48.	 Figure 48: Change Password Screen

8.3.12 Logout

Role: All roles

The Logout feature allows users to sign out of their account, ensuring that their session is securely closed. By clicking the Logout button, the user will be logged out of the application, and any active sessions will be terminated. This action helps protect the user's account from unauthorized access, especially when using shared or public devices. Once logged out, users can return to the login screen to sign in again if needed. Displayed as shown in Figure 49.	 Figure 49: Logout Screen

9 Conclusion

Effective food inventory management is a cornerstone of successful restaurant operations, as it directly impacts food safety, cost control, and customer satisfaction. Traditional methods of managing perishable inventory in restaurants often lack the precision and real-time capabilities required to prevent spoilage and optimize stock levels. To address these issues, we developed SFoodInventory, an IoT-powered system designed specifically for restaurant environments, combining advanced monitoring technologies with intuitive management tools.

SFoodInventory integrates weight sensors, temperature-humidity sensors, and a centralized cloud system to provide real-time tracking of critical storage conditions. The system is complemented by a user-friendly mobile application, which enables restaurant staff to monitor inventory levels, track expiration dates, and receive alerts when stock is low or nearing expiration. These features help restaurants proactively manage their inventory, reduce food waste, and ensure that food remains safe and fresh for customers.

In addition to its monitoring capabilities, SFoodInventory offers powerful data analytics tools, allowing restaurant managers to analyze food consumption trends, forecast stock requirements, and plan restocking more efficiently. By using these insights, restaurants can make data-driven decisions that minimize waste, improve operational efficiency, and optimize costs. The system not only enhances day-to-day inventory management but also supports long-term planning to align with business goals.

During testing in real-world restaurant environments, SFoodInventory demonstrated its ability to adapt to the dynamic needs of restaurant operations. Its durability, reliability, and ease of use make it an ideal solution for managing perishable inventory, even in high-pressure settings. The intuitive mobile interface ensures that staff can quickly access critical information and respond promptly to inventory needs, reducing human error and improving workflow efficiency.

Looking ahead, future enhancements to SFoodInventory will focus on scaling its capabilities to support larger restaurant chains and more complex operations. Planned updates include AI-driven features such as automated menu planning based on inventory availability, cooking recommendations tailored to existing stock, and nutritional analysis for menu optimization. These features aim to provide restaurants with smarter tools for improving efficiency and offering a better dining experience to customers.

In conclusion, SFoodInventory delivers an innovative, reliable, and scalable solution for managing restaurant food inventory. By combining IoT technology, real-time monitoring, and data analytics, the system helps restaurants reduce food waste, ensure food safety, and streamline operations. As a cost-effective and user-friendly tool, SFoodInventory empowers restaurants to achieve greater sustainability, efficiency, and customer satisfaction, making it a vital asset in the modern foodservice industry.

References

[1] FAO, "The State of Food Security and Nutrition in the World 2021," *Food and Agriculture Organization of the United Nations*, [Online]. Available: https://www.fao.org/publications/sofi/2021/en/. [Accessed: 07 Dec. 2024].

[2] K. W. Lee, S. H. Lee, and J. H. Lee, "A review of the IoT applications in foodservice management," *Food Control*, vol. 106, 2020, pp. 106759, [Online]. Available: https://doi.org/10.1016/j.foodcont.2019.106759. [Accessed: 07 Dec. 2024].

[3] X. Li, Y. Zhang, and M. Chen, "IoT-based smart inventory system for reducing food waste in restaurants," *Sustainability*, vol. 13, no. 18, 2021, pp. 1008-1019, [Online]. Available: https://doi.org/10.3390/su13181008. [Accessed: 07 Dec. 2024].

[4] T. T. H. Nguyen, "Leveraging mobile applications for food inventory optimization in small businesses," *Journal of Mobile Technology in Foodservice*, vol. 15, no. 4, 2022, pp. 202-213, [Online]. Available: https://jmtf.example.com/leveraging-mobile-applications. [Accessed: 07 Dec. 2024].

[5] Liang, H.-C., "Smart Inventory Management System of Food-Processing-and-Distribution Industry," *Procedia Computer Science*, vol. 17, pp. 373-378, 2013, doi: 10.1016/j.procs.2013.05.048.

[6] Kumar, S., Raut, R. D., Agrawal, N., Cheikhrouhou, N., Sharma, M., and Daim, T., "Integrated blockchain and internet of things in the food supply chain: Adoption barriers," *Technovation*, vol. 118, p. 102589, Jul. 2022, doi: 10.1016/j.technovation.2022.102589.

[7] Sultana, A., Billah, M. M., Ahmed, M. M., Aftab, R. S., Kaosar, M., and Uddin, M. S., "Applications of IoT-Enabled Smart Model: A Model For Enhancing Food Service Operation in Developing Countries," *Journal of Applied Engineering and Technological Science (JAETS)*, vol. 5, no. 2, pp. 1123-1141, Jun. 2024, doi: 10.37385/jaets.v5i2.4937.

[8] Desingh, V., and Baskaran, R., "Internet of Things adoption barriers in the Indian healthcare supply chain: An ISM‐fuzzy MICMAC approach,"

International Journal of Health Planning and Management, vol. 37, no. 1, Sep. 2021, doi: 10.1002/hpm.3331.

[9] O. Mulay, M. Bhalerao, S. Bhamare, V. Gaikwad, and K. Nalavade, "IOT Based Food Inventory Tracking System for Domestic and Commercial Kitchens," International Journal of Innovations in Engineering Research and Technology [IJIERT], vol. 7, no. 4, Apr. 2020, ISSN: 2394-3696.

[10] Rana, M. E., Shanmugam, K., and Hang, C. Y., "Recommendations for Implementing an IoT based Inventory Tracking and Monitoring System," in *Emerging Technologies for Digital Infrastructure Development*, Bentham Science Publishers, 2023, pp. 24-35.

[11] Mulay, O., Bhalerao, M., Bhamare, S., Gaikwad, V., and Sabri, M. S., "IOT BASED FOOD INVENTORY TRACKING SYSTEM FOR DOMESTIC AND COMMERCIAL KITCHENS," *SSRN Electronic Journal*, vol. 7, no. 4, pp. 13-19, Apr. 2020.

[12] JetBrains, "State of Developer Ecosystem 2024," [Online]. Available: https://blog.jetbrains.com/team/2024/07/15/help-shape-the-future-of-tech-participate-in-the-developer-ecosystem-survey-2024/. [Accessed: 07 12 2024].

[13] Netcraft, "Web Server Market Share April 2024," [Online]. Available: https://www.netcraft.com/blog/april-2024-web-server-survey/. [Accessed: 07 12 2024].

[14] Jenkins Documentation, "Why Choose Jenkins for CI/CD?," [Online]. Available: https://www.jenkins.io. [Accessed: 07 12 2024].

[15] Jenkins Blog, "Extensibility and Automation with Jenkins," [Online]. Available: https://blog.jenkins.io. [Accessed: 07 12 2024].

[16] DevOps Research and Assessment (DORA), "The State of DevOps Report," [Online]. Available: https://dora.dev. [Accessed: 07 12 2024].

[17] DevOps Research and Assessment (DORA), "Accelerate: State of DevOps Report," [Online]. Available: https://cloud.google.com/resources/devops/state-of-devops?. [Accessed: 07 12 2024].

[18] Stack Overflow, "How to build your OAuth flow when using Google ID Token," [Online]. Available: https://stackoverflow.com/questions/64607865/how-to-build-your-oauth-flow-when-using-google-id-token. [Accessed: 07 12 2024].

[19] O. Mulay, M. Bhalerao, S. Bhamare, V. Gaikwad, and K. Nalavade, "IOT Based Food Inventory Tracking System for Domestic and Commercial Kitchens," International Journal of Innovations in Engineering Research and Technology [IJIERT], vol. 7, no. 4, Apr. 2020, ISSN: 2394-3696.

[20] Afreen, H., and Bajwa, I. S., "An IoT-Based Real-Time Intelligent Monitoring and Notification System of Cold Storage," *IEEE Access*, vol. PP, no. 99, p. 1, Feb. 2021, doi: 10.1109/ACCESS.2021.3056672.

[21] Maier, A., Sharp, A., and Vagapov, Y., "Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things," in *Proc. of the 2017 IEEE ITECHA Conference*, Sep. 2017, doi: 10.1109/ITECHA.2017.8101926.

[22] EVELTA, "ESP32: The Ultimate Guide to the Versatile IoT Microcontroller," [Online]. Available: https://evelta.com/blog/esp32-the-ultimate-guide-to-the-versatile-iot-microcontroller/#:~:text=The%20ESP32%20stands%20out%20in,home%20devices%20to%20industrial%20sensors. [Accessed: 07 12 2024].

[23] MMicrosoft Azure, "Azure SQL Database Overview," [Online]. Available: https://azure.microsoft.com/services/sql-database. [Accessed: 07 12 2024].

[24] Spring Documentation, "Integration with Azure SQL Server," [Online]. Available: https://spring.io/guides. [Accessed: 07 12 2024].

[25] Microsoft Azure Blog, "Scalability and Disaster Recovery in Azure SQL," [Online]. Available: https://techcommunity.microsoft.com. [Accessed: 07 12 2024].

[26] AWS Documentation, "Amazon RDS Pricing," [Online]. Available: https://aws.amazon.com/rds. [Accessed: 07 12 2024].

[27] Google Cloud Documentation, "Cloud SQL for SQL Server Pricing," [Online]. Available: https://cloud.google.com/sql. [Accessed: 07 12 2024].

[28] Maier, A., Sharp, A., and Vagapov, Y., "Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things," *Proc. of the 7th Int. Conf. on Internet Technologies and Applications*, 2017, pp. 1-6.

[29] Stack Overflow, "Stack Overflow Developer Survey 2023," Stack Overflow, 2023. [Online]. Available: https://survey.stackoverflow.co/2023. [Accessed: 07 12 2024].

[30] Statista, "Market share of programming languages in mobile app development," Statista, 2023. [Online]. Available: https://www.statista.com/statistics/1124754/global-market-share-of-programming-languages-in-mobile-app-development/. [Accessed: 07 12 2024].

[31] Leung, B., and Wong, K., "Benefits of MVVM in mobile application development: A comparative study," *Journal of Mobile Development and Design*, vol. 23, no. 4, pp. 45-59, 2020.

[32] Carvalho, J., Lima, M., and Ferreira, P., "Exploring the MVVM pattern for responsive and scalable mobile apps," *International Journal of Software Architecture*, vol. 15, no. 3, pp. 198-213, 2021.

[33] "IoT Testing: How to Test Internet of Things Solutions," [Online]. Available: https://testfort.com/blog/iot-testing-how-it-works-why-you-cant-do-without-it. [Accessed: 07 12 2024].

[34] "What is Unit Testing? - AWS," [Online]. Available: https://aws.amazon.com/what-is/unit-testing/#:~:text=Unit%20testing%20is%20the%20process,test%20for%20each%20code%20unit. [Accessed: 07 12 2024].

[35] "What is Integration Testing? Definition, How-to, Examples," [Online]. Available: https://katalon.com/resources-center/blog/integration-testing. [Accessed: 07 12 2024].