# Inventory Management

## *Inventory Management Mobile Application For Restaurant And Similar Businesses*

Washington State University

**Team One**

Amrit Banga

Tucker Brisbois

Zijian Zhang

[05/24/2024]

# TABLE OF CONTENTS

# I. Introduction

The team will be working on creating an all new Restaurant Inventory Management mobile application for iOS. This application will bring an ease of use to managing inventory. User's will be able to track all inventory logged and also mark expiration dates and cost. This will be done on iOS mobile devices and be able to be accessed on multiple devices due to cloud saving for the data.

An application like this will be essential in maximizing efficiency in any restaurant or food based business. By keeping track of inventory along with their expiration dates, business owners can know exactly when to buy more of a certain product and know if anything has expired and needs replacing. This will drastically reduce the possibility of serving expired products to Users or over/under purchasing certain food based products.

# II. System Requirements Specification

Use Case 1: User Registration and Login

A new user registers for an account and logs into the system using two-factor authentication. This involves the restaurant manager or staff opening the application and being greeted with a welcome page offering sign-in or sign-up options. Upon selecting "Sign Up," the user inputs a username and password, which the system securely stores, returning the user to the main page. When the user selects "Sign In," they enter their credentials, which the system verifies, then directs the user to the two-factor authentication page. The user receives a code from another source (such as email or SMS), enters it, and upon verification, gains access to the main dashboard. This use case addresses the requirement for an accounts-based access control system (R1) and a two-factor authentication system (R2).

Use Case 2: Adding a New Inventory Item

This scenario involves a user adding a new food item to the inventory, including details such as expiration date, quantity, and price. The restaurant staff, already logged into the system, navigates to the inventory management section from the main dashboard. They select the option to add a new inventory item and enter relevant details like the name, expiration date, quantity, and price. The system stores these details in the database and confirms successful addition to the user. This use case pertains to the requirement for integration with databases for storing inventory details (R3).

Use Case 3: Viewing and Editing Inventory

In this use case, a user views the current inventory and edits the details of an existing item. The restaurant manager or staff, who has inventory management permissions, logs into the system and navigates to the inventory management section from the main dashboard. They select an item from the inventory list to view its details and edit necessary information such as quantity or price. The system updates the item details in the database and confirms the update to the user. This use case supports the requirement for database integration for storing inventory details (R3).

Related Requirements

- R1: An accounts-based access control system

- R2: Two-factor authentication system

- R3: Integration with databases for storing inventory details

## II.1. Functional Requirements

### Login System

**Implement 2FA:** The system must support two-factor authentication to enhance the security of user logins.
**Source:** Users need to go through two-factor authentication to ensure the security of the information
**Priority:** Level 2

**User Registration:** The system needs to allow new users to create accounts, requiring necessary information like username and password. And the second way to authenticate their identity.
**Source:** Users need account creation and management.
**Priority:** Level 2

**User Authentication:** The system needs to authenticate users using a secure login process that requires a username and password and another way to authenticate them twice.
**Source:** Users need to use this way to get their data without leaking.
**Priority:** Level 2

### User Interface

**UI:** The system needs to create a UI for managers and staff to use the system.
**Source:** Users need a UI to use.
**Priority:** Level 2

**Friendly UI:** The system needs to create a friendly UI for managers and staff which could be easy to use.
**Source:** Users need an easy UI to use to ensure widespread adoption.
**Priority:** Level 2

### Inventory Logging System

**Register Inventory:** The system needs to allow users to log inventory items, including details such as item type, quantity, and date received.
**Source:** Users need to manage varied food.
**Priority:** Level 3.

**Delete Inventory:** The system needs to allow users to delete inventory items, including details such as item type, quantity, and date received.
**Source:** Users need to manage varied food.
**Priority:** Level 3.

**Update Inventory:** The system needs to allow users to change inventory items, including details such as item type, quantity, and date received.
**Source:** Users need to manage varied food.
**Priority:** Level 3.

**Track Inventory:** The system needs to allow users to search inventory items, showing details such as item type, quantity, and date received.
**Source:** Users need to track everything in their restaurant.
**Priority:** Level 3.

**Expiration Showing System**
**Show Expiration:** The system needs to show all expired items in the restaurant today.
**Source:** Users need to ensure everything in the restaurant is good.
**Priority:** Level 1.

**Cost Showing System**
**Show Costs:** The system needs to allow users to see the cost of inventory items to track financial data**.**
**Source:** Users need financial tracking and reporting.
**Priority:** Level 1.

**History of Everything System**
**Maintain Logs:** The system needs to maintain an audit log of all user activities to track changes.
**Source:** Users need to track every action in the system.
**Priority:** Level 2.

## II.2. Non-Functional Requirements

Performance Requirements

- Scalability:

The system shall be able to handle simultaneous access by multiple users without performance degradation, allowing for expansion as the user base grows.

- Data Integrity:

The system shall ensure that all data entered and stored in the database is accurate and consistent, preventing data corruption and loss.

Security Requirements

- Data Encryption:

The system shall use TLS encryption for all data transmissions to ensure the security and privacy of user information.

- Access Control:

The system shall implement role-based access control, ensuring that only authorized users can access functionalities and data within the system.

Usability Requirements

- User Interface:

The system shall have an intuitive and user-friendly interface that requires minimal training for new users to navigate and utilize its features effectively.

- Accessibility:

The system shall be accessible on both iOS and web platforms, ensuring that users can access the system from a variety of devices.

Maintainability Requirements

- Modular Design:

The system shall be designed in a modular manner, allowing for easy updates and maintenance without affecting the entire system.

- Documentation:

The system shall include comprehensive documentation for both users and developers, facilitating ease of use and future development.

Process Requirements

- Programming Language:

The system shall be developed using Swift for iOS development, ensuring compatibility and optimal performance on Apple devices.

- Development Methodology:

The system shall follow Agile development practices, allowing for iterative development, continuous feedback, and adaptive planning.

Testing Requirements

- Unit Testing:

The system shall include comprehensive unit tests for all major functionalities to ensure individual components work as intended.

- Integration Testing:

The system shall undergo integration testing to ensure that different modules and components work together seamlessly.

## III. System Evolution

In terms of system evolution, one of the biggest cases for evolution will be due to the evolution of iOS as well. Every month or so iOS is updated with bigger updates coming once a year. In order to stay functional on newly updated devices, system changes may need to be developed in order for the application to function properly.

Another source of system evolution will be on the hardware side. New iOS devices are released each year and eventually certain devices will be running older and older versions of iOS. The decision on how old of an operating system the application will be compatible with will also be a source of evolution.

A way the system will need to be updated drastically, would be in the potential port to android devices. The current version of the application being developed is only compatible with iOS devices as it is being built using Swift. Swift is the language used to develop on iOS usually, and is not compatible with Android. To port the application to Android, the application will need to be rebuilt using Java.

Minor changes need to be made in the future with the help of user feedback. These changes include UI changes, bug fixes, and/or new features. Changing UI will be relatively simple using the Swift platform, as will bug fixes. New features can vary drastically in terms of difficulty.

## IV. Glossary

iOS: The operating system on iPhones

Java: A coding language used to built applications

UI (User Interface): relates to the interface the user will interact with during the application

## V. References