# Software Requirements Specification Template

Smart Inventory

Software Requirements Specification

Android Studio Iguana Runtime version: 17.0.9

05-02-2024

Sri Sai Krishna Bathula Sumedh Appanagari

Sai Kumar Thangalla Muni Venkata Bhanu Venkata Siva Kumar Gorantla

Submitted in partial fulfillment Of the requirements of

CSIS 44-691 Graduate Directed Project 1

# Revision History

**Date Description Author Comments**

|  |  |  |  |
| --- | --- | --- | --- |
| <date> | <Version 1> | <Your Name> | <First Revision> |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

**Signature Printed Name Title Date**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Table of Contents

**Table of Contents**

1. Introduction
   1. Purpose
   2. Scope
   3. Definitions, Acronyms, and Abbreviations
   4. References
   5. Overview
2. General Description
   1. Product Perspective
   2. Product Functions
   3. User Characteristics
   4. General Constraints
   5. Assumptions and Dependencies
3. Specific Requirements
   1. External Interface Requirements
      1. User Interfaces
      2. Hardware Interfaces
      3. Software Interfaces
      4. Communications Interface
   2. Functional Requirements
   3. Use Cases
   4. Class/Objects
   5. Non-Functional Requirements
      1. Performance
      2. Reliability
      3. Availability
      4. Security
      5. Portability
   6. Inverse Requirements
   7. Design Constraints
   8. Logical Database Requirements
   9. Other Requirements
   10. Prototypes (for complete project)
   11. Use Case Diagrams
4. Design
   1. ER diagram
   2. GUI
5. Analysis Models
   1. Data Flow Diagram
   2. Sequence Diagram

## Introduction

In today’s fast-paced business environment, efficient inventory management is crucial for maintaining a competitive edge. With rapid technological advancements, smart

inventory management systems have emerged as a powerful tool for optimizing efficiency and streamlining operations.

A smart inventory management system is a software application that streamlines and optimizes inventory management processes. It is designed to help a company have

multiple warehouses or godowns. The system's main feature is the centralized database that allows for real-time monitoring and tracking of inventory level, movement, and

location across all of the company’s storage facilities. With centralized data

management, managers can easily report and analyze their inventory, allowing them to make better decisions about it. The system can also automatically replenish inventory

based on predetermined minimum stock thresholds, helping the company maintain the right inventory levels and reduce stockouts. The goal of the Smart inventory

management system is to: Reduce manual effort; Improve inventory accuracy; Enhance overall operational efficiency; and maintain optimal inventory levels across the

company's multiple locations

### Purpose

The Smart Inventory Project aims to develop a user-friendly and efficient system that:

* + - Enables users to view current inventory levels and request specific products.
    - Empower the owner to manage inventory, update stock levels, fulfill requests, and track historical data.
    - Provides real-time visibility into inventory status for improved decision-making.

### Scope

This project focuses on building a mobile application with the following functionalities:

* + - User App:
      * View current inventory with product details and stock levels.
      * Send requests for specific items to the owner.
      * Track past requests and product movements.
    - Owner App:
      * Add, edit, and delete inventory items.
      * Update stock quantities based on sends and receives.
      * Process user requests and send out products.
      * View and respond to user requests for specific items.

### Definitions, Acronyms, and Abbreviations

* + - **SMART INVENTORY:** Smart Inventory Management System
    - **Android Studio:** Integrated development environment for building Android applications.
    - **Firebase:** Google's mobile platform for developing cloud-based apps.
    - **AWS Cloud:** Amazon Web Services cloud computing platform.
    - **API:** Application Programming Interface, a set of protocols for software components to communicate.
    - **UI:** User Interface, the visual elements users interact with.

### References

* + - Firebase Documentation: <https://firebase.google.com/docs/>
    - AWS Documentation: <https://aws.amazon.com/>
    - Android Studio Development Tutorials: <https://developer.android.com/studio>

### Overview

This document provides a high-level overview of the SMART INVENTORY Project. The following sections will deliver deeper into the specific functionalities, implementation details, and considerations for building a robust and scalable inventory management system using Android Studio.

# General Description

## Product Perspective

1. **User Interface (UI) Design**: Create an intuitive and user-friendly interface for the Android application**.** Ensure that users can easily navigate the application and perform various management tasks such as adding new products**,** updating quantities, and generating reports.
2. **Real-Time Data Syncing**: Enable real-time synchronization of inventory data between Android application and backend database or server. This ensures that users have access to the latest information on inventory, sales, and incoming shipments
3. **Inventory Tracking and Alerts**: Add features to track inventory levels and set alerts for low inventory levels or expiring items. This helps users proactively manage inventory and avoid out-of-stock or overstock situations.
4. **Integration with Other Systems**: Consider integrating your inventory management application with other business systems, such as accounting software. This facilitates the smooth flow of data between different systems and improves overall business efficiency..
5. **Security and Permissions**: Implement security measures to protect sensitive storage data stored in the application. Use authentication mechanisms such as credentials or biometric authentication and enforce role-based access control to restrict user rights based on their role in the organization.
6. **Scalability and Performance**: Design the application architecture to be scalable and capable of handling large amounts of storage data. Optimize performance to ensure smooth operation even under heavy use.

## Product Functions

### Item Management:

* + Add new items to the inventory database.
  + Edit the details of an existing product, such as name, description, category, and price.
  + Remove or deactivate products that are no longer in stock or in production.

### Barcode Scanning:

* + Scan barcodes or QR codes to quickly add or update products in stock.
  + Get product information, such as name, description, and price, from scanned codes.

### Inventory Tracking:

* + Track the inventory of each product in real-time.
  + Track inventory movements, including sales, purchases, and adjustments.
  + Set warnings about low stock levels to avoid running out of stock.

### Order Management:

* + Create purchase orders to replenish inventory.
  + Receive and process incoming shipments based on purchase orders.
  + Create packing lists and shipping labels for outbound orders.

### User Management:

* + Manage user accounts and access rights in the application.
  + Assign roles and rights to users according to their responsibilities (e.g., system administrator, warehouse manager, sales representative).
  + Monitor user activity and audit trails for accountability.

### Integration with External Systems:

* + Integrate with accounting software to synchronize inventory and financial data.
  + Integrate with point-of-sale systems for seamless sales and inventory management.
  + Sync with e-commerce platforms to update real-time product catalogs and inventory..

### Customization and Configuration:

* + Configure settings such as units of measurement, currency, and tax rates.
  + Set alerts and notifications based on user preferences.
  + Customize the application to meet specific industry requirements or business workflows.

## User Characteristics

### Inventory Managers/Operators:

* + These users are responsible for daily warehouse management tasks.
  + They need a user-friendly interface to efficiently add, edit, and track inventory items.
  + They need access to real-time inventory data to make informed decisions about inventory levels and replenishment.

### Warehouse Staff:

* + Storekeepers manage the physical movements of the warehouse, such as receiving, picking, and shipping.
  + They need a mobile-friendly interface that allows them to trade inventory on the go.
  + Barcode scanning is essential to quickly identify and process items in stock.

### Administrators/Managers:

* + Administrators control the entire inventory process and are responsible for configuring system settings and access rights.
  + They need access to advanced reporting and analysis tools to monitor inventory performance and identify areas for improvement.
  + Security features such as role-based access control are critical for managing user access rights and ensuring data integrity.

### Business Owners/Decision Makers:

* + Business owners and decision-makers rely on inventory data to make strategic purchasing, pricing, and resource allocation decisions.
  + These require access to advanced metrics and statistics to assess the overall health of the business and identify growth opportunities.
  + Customizable dashboards and summaries are essential to present key performance indicators (KPIs) clearly and concisely.

### Remote Users:

* + Remote users may need access to inventory data while working in the field or out of the office.
  + They need a mobile-friendly application that allows them to view inventory, place orders, or perform other actions on their smartphone or tablet.
  + Offline usage is essential for productivity, even in areas with limited or no internet access.

## General Constraints

1. **Device Compatibility**: Make sure the app is compatible with multiple Android devices, including smartphones and tablets with different screen sizes, resolutions, and hardware capabilities.
2. **Resource Constraints**: Android devices may have limited resources such as memory, processing power, and battery life. Optimizing your application to minimize resource usage and ensure a smooth user experience is essential.
3. **Network Connectivity**: Design the application with intermittent or unreliable network connections, especially in environments with limited signal strength or data coverage. Enable offline features to allow users to continue working with inventory data even when offline.
4. **Security**: Implement strong security measures to protect sensitive storage data stored on devices and transmitted over networks. Use encryption for data storage and communication, implement secure authentication mechanisms, and enforce access controls to prevent unauthorized access.
5. **Scalability**: Design the application architecture to be scalable and able to handle inventory data and user traffic growth over time. Ensure back-end infrastructure can scale horizontally to meet growing demands without compromising performance or reliability.
6. **Testing and Quality Assurance**: The fragmentation and diversity of Android devices make comprehensive testing difficult. Testing your application across devices, operating system versions, and network conditions to ensure compatibility, reliability, and performance is time-consuming but necessary.
7. **User Experience**: Prioritize user experience (UX) design to create an intuitive and user-friendly interface that enables users to perform inventory management tasks easily and efficiently. Conduct usability tests and collect user feedback to continuously improve applications and performance.

## Assumptions and Dependencies Assumptions:

1. **Data Accuracy**: Assuming inventory data provided by users or imported from external sources is accurate and up-to-date. This includes the assumption that the barcode data scanned by users exactly matches the products in stock.
2. **Network Connectivity**: Assuming users have a reliable internet connection, sync data with the backend server in real time. However, offline functionality should be provided as a backup mechanism in situations where online connectivity is not available.
3. **Device Compatibility**: Assuming that the Android application works correctly on several different devices with different screen sizes, resolutions, and hardware specifications. This includes the assumption that the user interface of the application will adapt to different device configurations.

## Dependencies:

1. **Backend Infrastructure**: Dependence on the availability and reliability of the backend infrastructure (including servers, databases, and APIs) that an Android application relies on to store, retrieve, and synchronize data.
2. **Database**: Reliance on a comprehensive and up-to-date database that accurately identifies products and retrieves relevant information when barcodes are scanned.
3. **API Integration**: Reliance on integration with external systems such as accounting software. This includes ensuring compatibility with the APIs provided by these systems for data exchange.
4. **Security Measures**: Dependency on implementing robust security measures to protect sensitive inventory data stored on devices and transmitted over networks. This includes dependency on encryption algorithms, authentication mechanisms, and secure network protocols.
5. **User Feedback and Testing**: Depending on user feedback, thorough testing is essential for pinpointing and resolving usability issues, bugs, and performance challenges. This encompasses user acceptance testing (UAT) to guarantee the functionality and user-friendliness of an Android app before it goes live.

# Specific Requirements

## External Interface Requirements

### User Interfaces

* + - * Intuitive and user-friendly interface for easy navigation.
      * Mobile-optimized design for seamless use on Android devices.
      * These interfaces will allow users to place orders, check inventory levels, and view order status.

### Hardware Interfaces

* + - * The app can run on Android smartphones with specifications like a 5.0-inch screen, 2GB of RAM, and 16GB of storage.

### Software Interfaces

* + - * Integration with existing ERP systems for data exchange and synchronization.
      * APIs will be used to communicate with shipping carriers for order fulfillment and tracking.
      * Import/export inventory data in common formats like CSV, and Excel.

### Communications Interface

* + - * It uses a client-server (C/S) three-tier architecture with database, service, and user (Android app) layers.
      * Web services technology is used for data exchange between Android apps and databases.

## Functional Requirements

* User authentication and access controls.
* Add, update, or delete inventory items with details like cost, quantity, and description.
* Track inward and outward stock movements.
* View real-time inventory levels and stock availability.
* Generate reports on stock levels, sales trends, and order history.

## Use Cases

* User Order Placement: The customer submits an order, triggering an inventory update.
* Users can log into the system, place the request as per requirement, track the request, and receive the product.
* Inventory Replenishment: low stock alert prompts automatic reorders from suppliers.
* Warehouse Management can log in to the system and perform CRUD operations like adding, removing, updating, and viewing products.
* Warehouse Management: Warehouse staff receive order details and fulfill shipments.

## Class/Objects

* Product: attributes include ID, name, description, price, and quantity.
* Order: Contains details like order ID, customer information, and items ordered.
* Warehouse: represents physical storage locations, with attributes such as location and capacity.
* Supplier: Manages details such as supplier ID, contact information, and delivery schedules.

## Non-Functional Requirements

### Performance

* + - * Fat response times for inventory queries and updates.
      * Handle large datasets without performance degradation.

### Reliability

* + - * Data integrity checks to prevent incorrect/duplicate entries.
      * Robust error handling and input validation.
      * The system should have an uptime of at least 99.9% and should be able to recover from failures within 5 minutes.

### Availability

* + - * Offline functionality for basic operations when the server is unavailable.
      * The system should be accessible 24/7 except during scheduled maintenance windows.

### Security

* + - * User authentication and authorization mechanisms should be implemented to ensure that only authorized users can access sensitive data.
      * Role-based access controls for different user types.
      * Data encryption should be used to protect data both in transit and at rest.

### Portability

* + - * Support for multiple Android versions and device types.
      * The system should be designed to run on both desktop and mobile platforms, with a responsive design for different screen sizes.

## Inverse Requirements

* + - The system should not allow unauthorized users to access sensitive inventory data.
    - The system should not delete inventory items without proper authorization.
    - The system should not allow negative inventory quantities.

## Design Constraints

* + - The application must comply with Android design guidelines for consistent user experience.
    - The application should be developed using Java or Kotlin programming languages for Android compatibility.
    - The application's design should accommodate varying screen sizes and resolutions.

## Logical Database Requirements

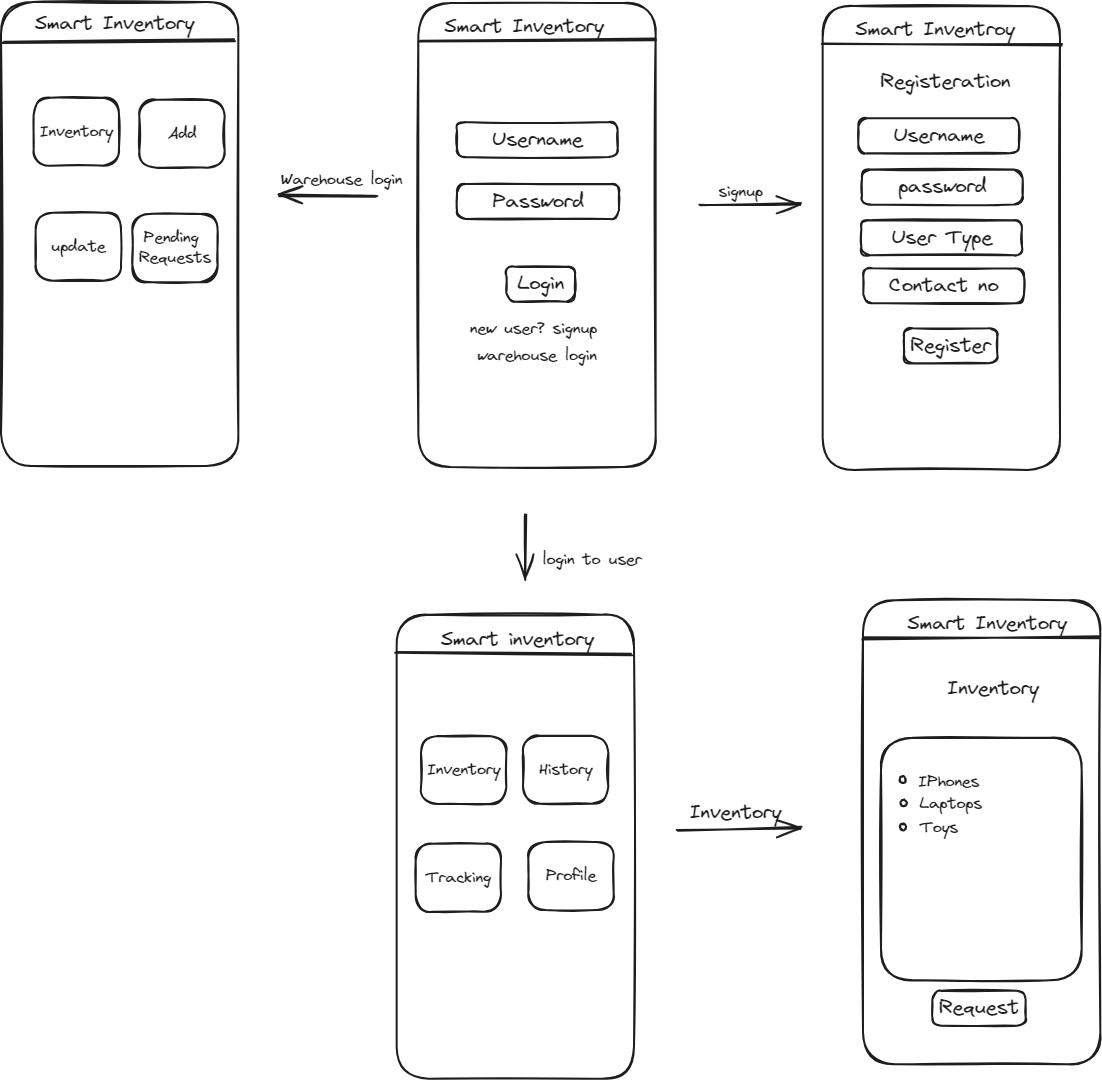
* + - The database should have tables for storing information about inventory items, orders, users, and transactions.
    - Relationships between tables should be defined to enforce data integrity.
    - Indexes should be created for efficient querying of large datasets.

## Other Requirements

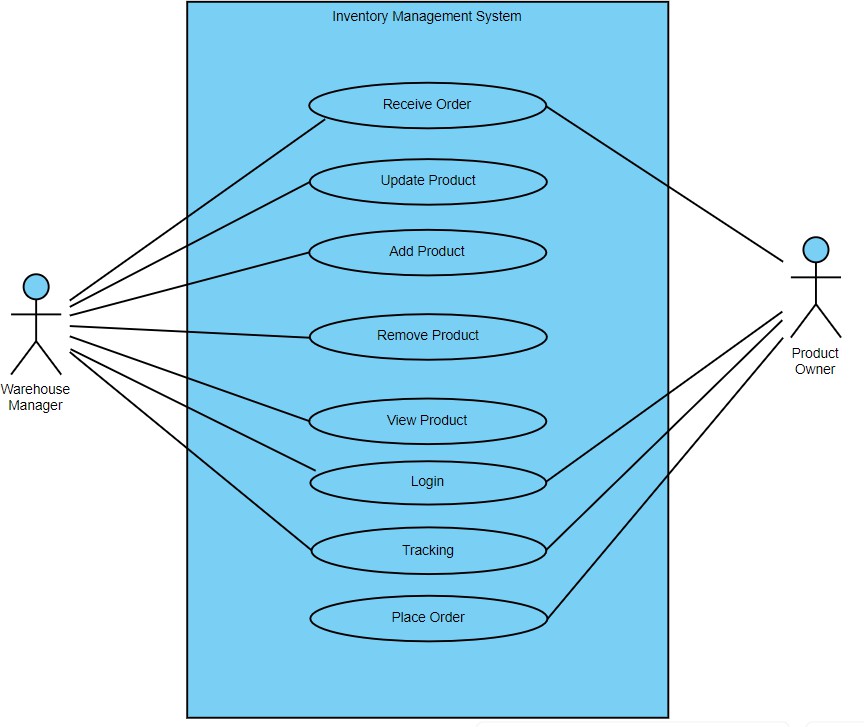
* + - The application should have a user-friendly onboarding process for new users.
    - The application should provide notifications for low inventory levels or pending orders.

## Prototypes (for the complete project)

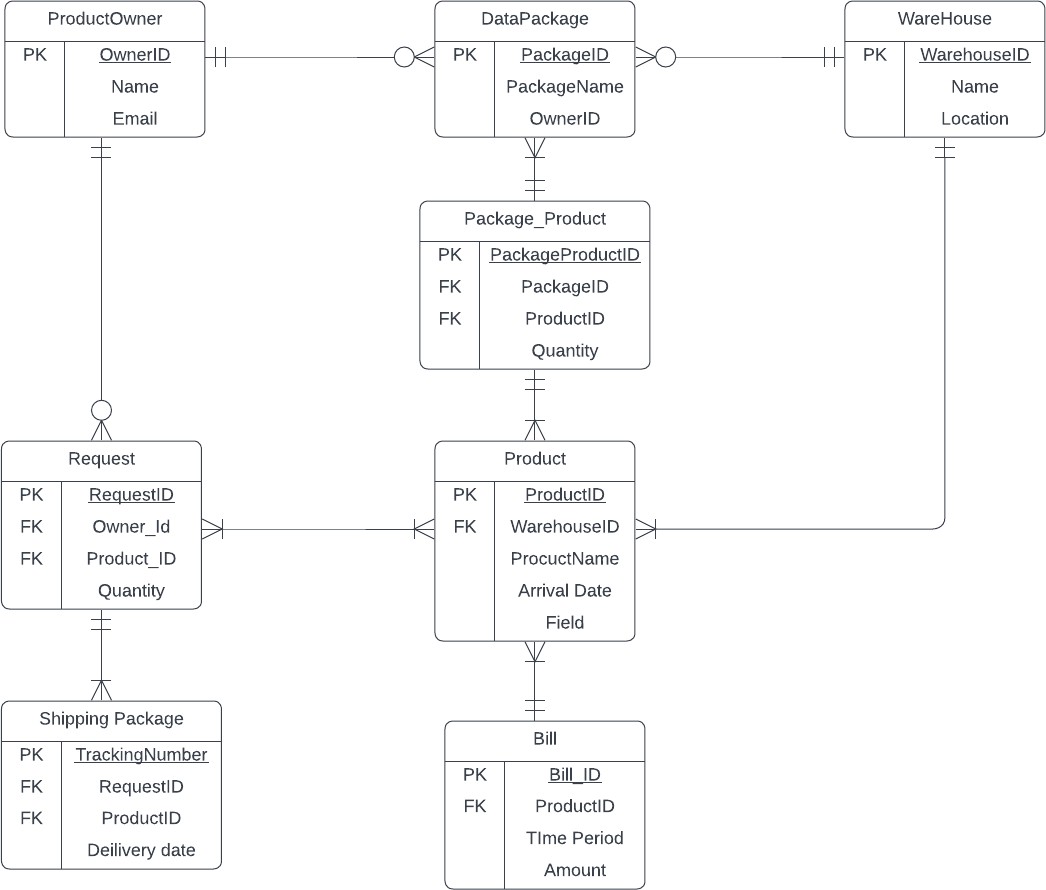
* + - Visualize UI and workflow via wireframes, illustrating layout, navigation, and functionality.



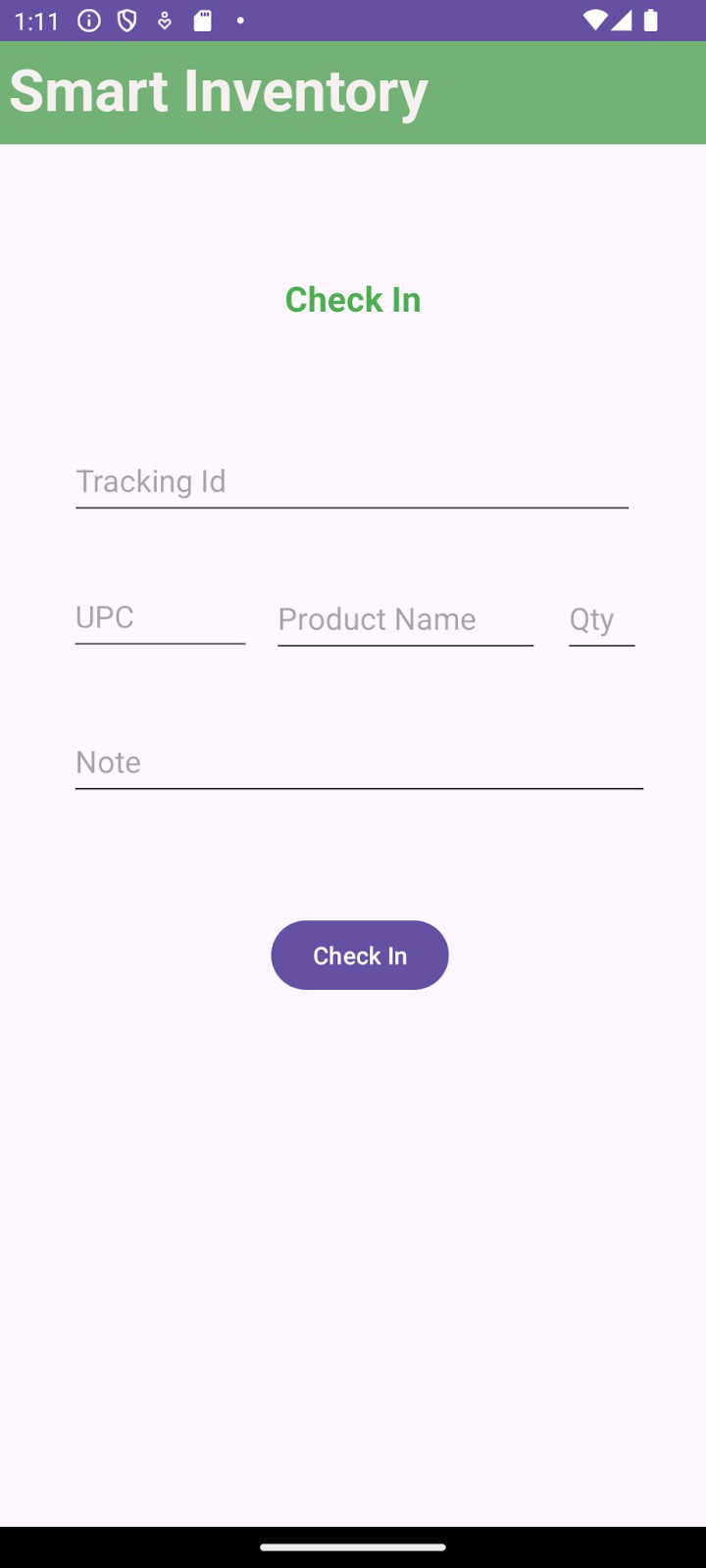
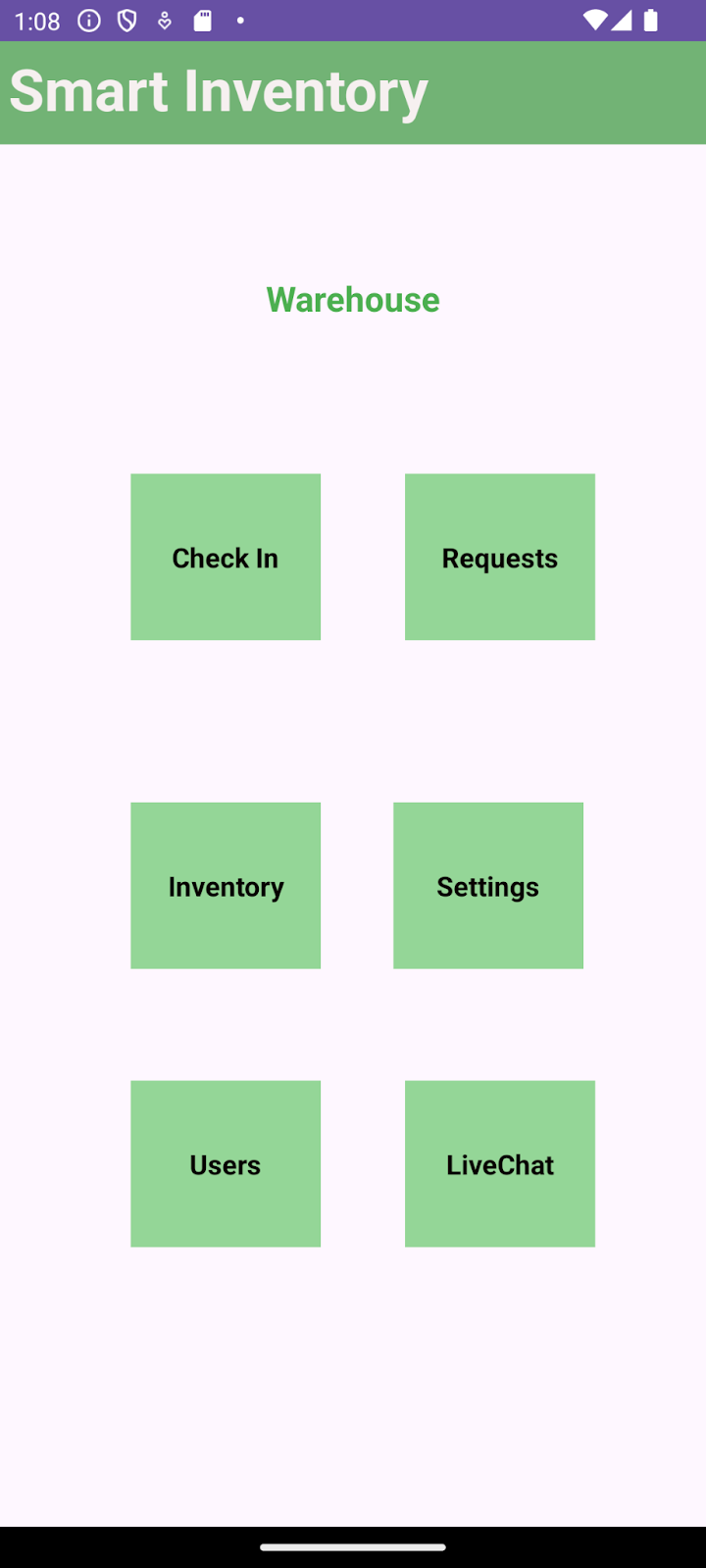
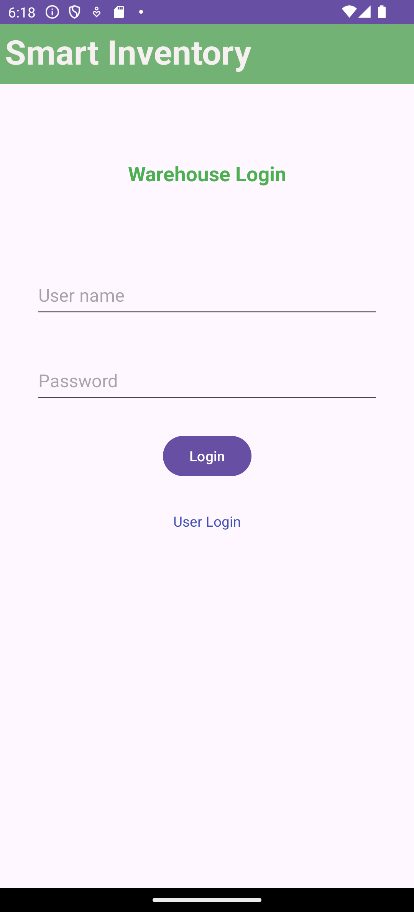
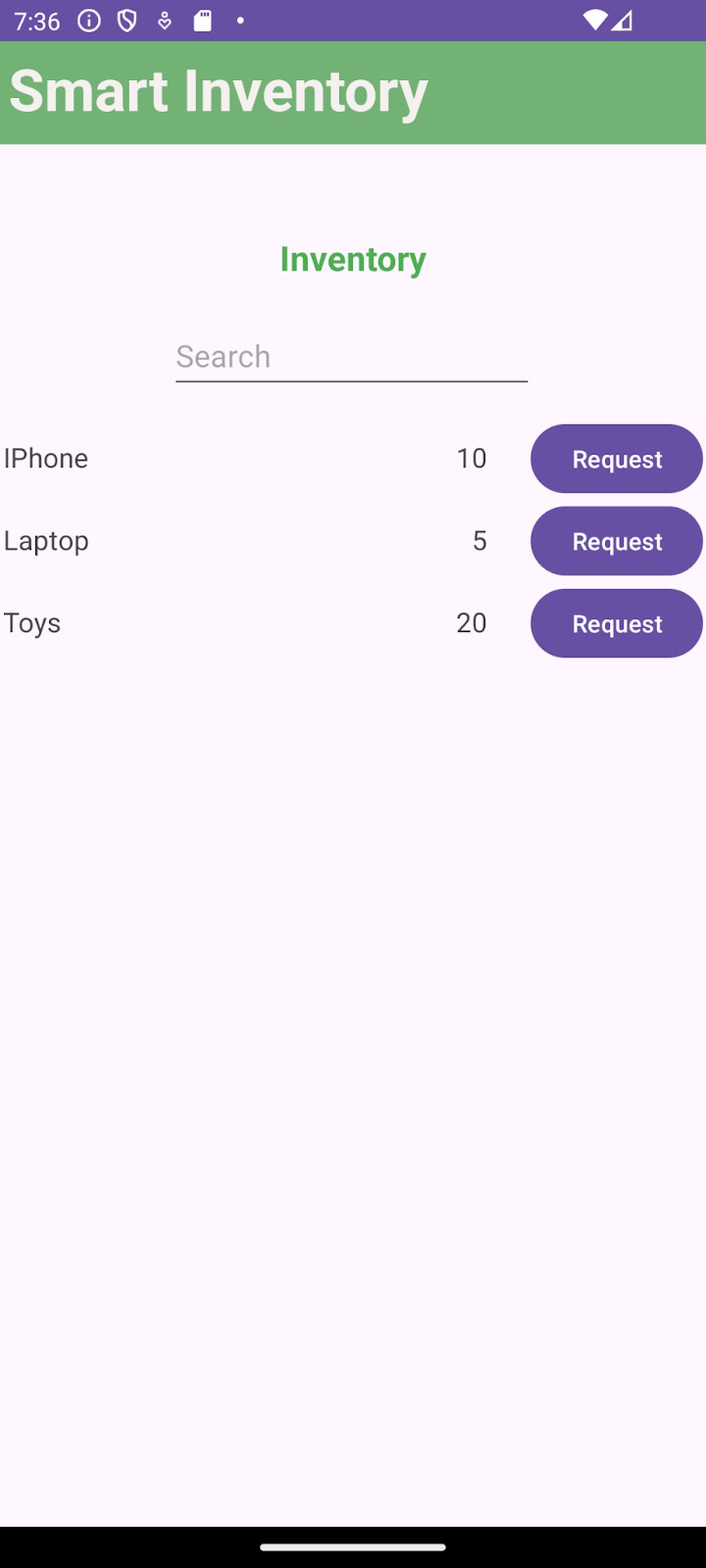
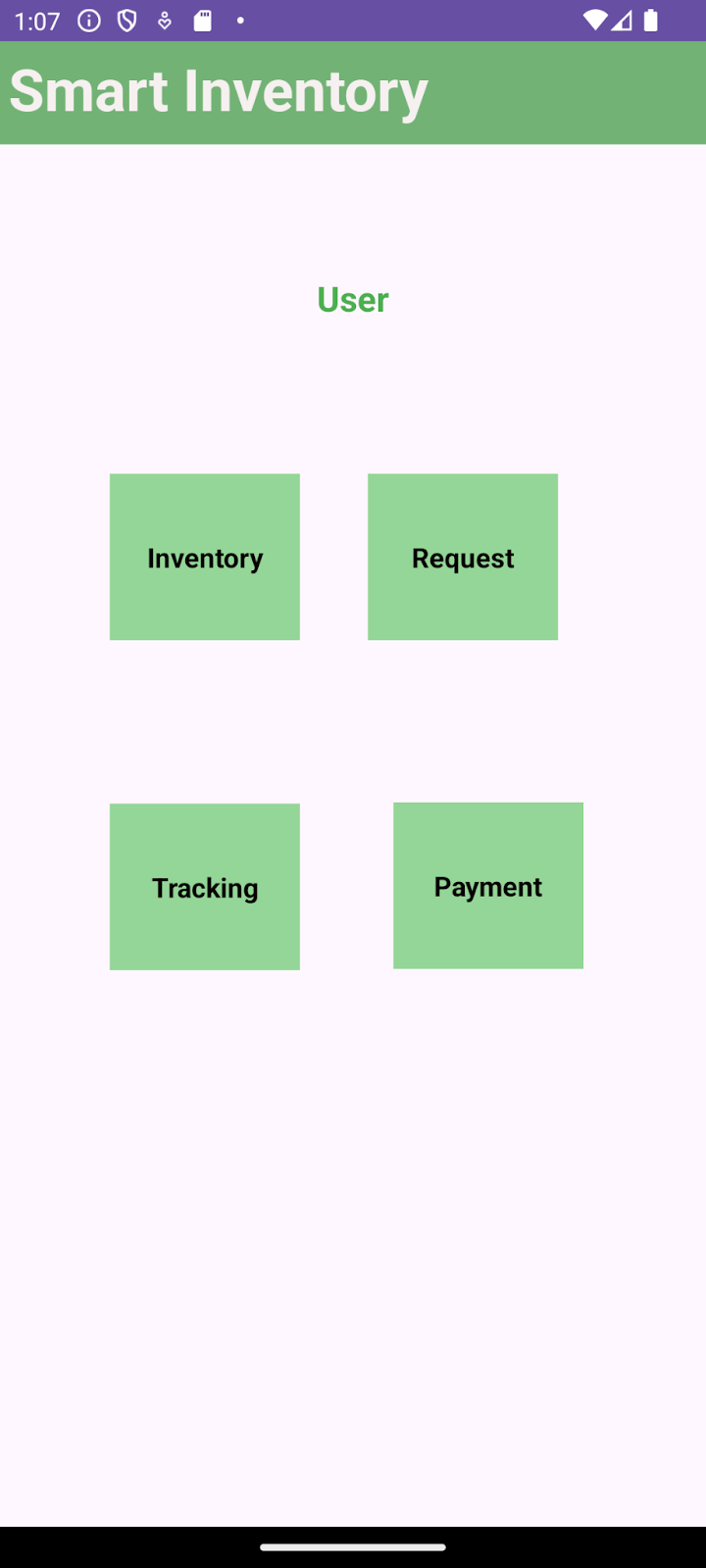
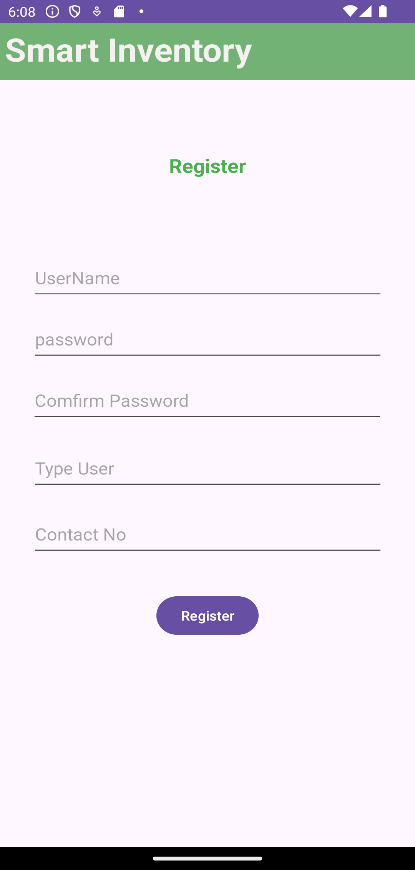
## Use Case Diagrams



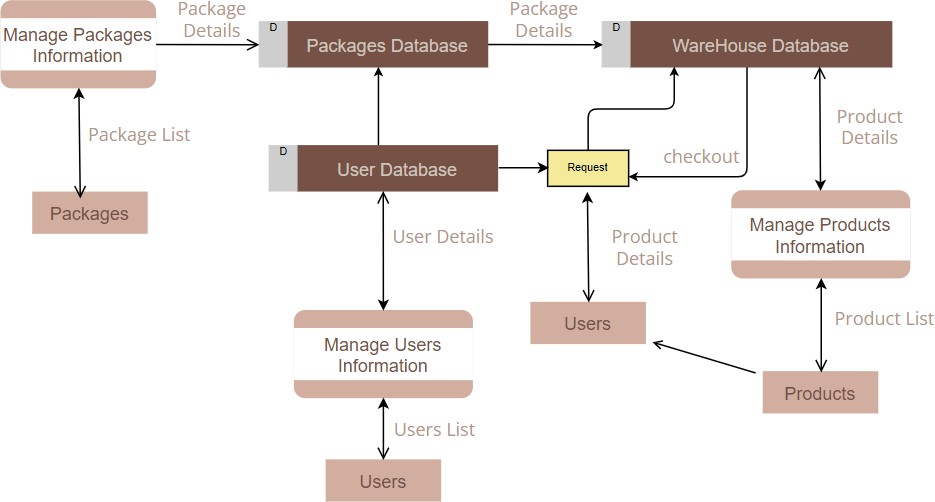
1. **Design**
   1. **ER diagram**



* 1. **GUI**



1. **Analysis Models**
   1. **Data Flow Diagram**



* 1. **Sequence Diagram**

