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

3.5.1. Performance

3.5.2. Reliability

* + 1. Availability
    2. Security
    3. Portability
  1. Inverse Requirements
  2. Design Constraints
  3. Logical Database Requirements
  4. Other Requirements
  5. Prototypes (for the complete project)
  6. Use Case Diagrams

1. Design

4.1. ER diagram

4.2. GUI

1. Analysis Models

4.1. Data Flow Diagram

4.2. Sequence Diagram

1. User Manual

6.1. Introduction

6.2. Warehouse Manager Interface

6.3. User Interface

1. Technical Manual

7.1. Introduction

7.2.

## Introduction

In a dynamic business landscape, efficient inventory management is key to sustaining a competitive advantage. With advancements in technology, smart inventory management systems have become indispensable for streamlining operations, reducing costs, and enhancing overall productivity. These systems are specifically designed to manage inventory across multiple warehouses or storage facilities by leveraging a centralized database, enabling real-time tracking of stock levels, movement, and location. Through data-driven insights and automated replenishment, businesses can maintain optimal inventory levels, prevent stockouts, and make more informed decisions. Ultimately, a Smart Inventory Management System aims to reduce manual workload, improve inventory accuracy, and elevate operational efficiency.

**1.1. Purpose**

The Smart Inventory Project aims to develop a comprehensive, user-friendly mobile application that:

* Allows users to access real-time inventory levels and request specific products.
* Enables owners to manage inventory across multiple locations, update stock, fulfill requests, and monitor historical data trends.
* Provides instantaneous visibility into inventory status, empowering managers with insights for better decision-making and more accurate forecasting.

**1.2. Scope**

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

* User App:
  + Allows users to view current inventory, including product details and stock levels.
  + Enables requests for specific items, sending notifications to the owner.
  + Provides tracking for past requests and product movements.
* Warehouse App:
  + Empower the owner to add, edit, and delete inventory items.
  + Updates stock levels based on received and dispatched items.
  + Facilitates processing of user requests, including product dispatch and user notifications.
  + Enables viewing and responding to user requests to stream

**1.3. 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.

**1.4. References**

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

**1.5. Overview**

This document serves as a high-level outline of the SMART INVENTORY Project, detailing the core functionalities, technical requirements, and strategic considerations for developing a robust and scalable inventory management solution using Android Studio. Subsequent sections will delve into specific features, implementation methodologies, and tools, providing a roadmap for creating an effective and adaptable smart inventory system.

# 2. General Description:

This section outlines the core features and architecture considerations for the Smart Inventory Management System. The product is designed to provide an intuitive, scalable, and secure solution for real-time inventory tracking, multi-location management, and enhanced decision-making.

## 2.1. Product Perspective:

The Smart Inventory Management System is a mobile application optimized for Android that connects multiple warehouses and users through centralized data management. Key features include:

1. **User Interface (UI) Design**: The application prioritizes a clean, user-friendly interface, ensuring that users can intuitively navigate and perform tasks such as viewing product details, adjusting stock levels, and tracking historical data. The UI is designed for accessibility, allowing users across varying technical skills to interact seamlessly with inventory functions and reporting tools.

2. **Real-Time Data Syncing**: Real-time synchronization between the mobile app and the backend database ensures consistent and updated inventory information across all connected warehouses and user applications. This feature allows users to access accurate, up-to-date data on inventory levels, item movement, and incoming shipments, promoting informed decision-making and timely responses.

3. **Inventory Tracking and Alerts**: The system includes proactive inventory tracking and alert notifications. Users receive alerts for low stock levels, nearing expiration dates, and potential stockouts, empowering them to take preemptive action. This feature minimizes the risk of overstock or understock situations, optimizing storage management and preventing revenue loss.

5. **Integration with Other Systems**: Integration with other critical business systems, such as accounting or order management software, facilitates seamless data flow, improving efficiency and reducing manual data entry. The ability to interface with external systems ensures that inventory data aligns with broader business operations, supporting cohesive workflows and financial reporting.

6. **Security and Permissions**: Robust security features protect sensitive inventory and user data, utilizing role-based access control and secure authentication methods (e.g., credentials or biometric authentication). Role-based access limits user permissions based on their organizational roles, ensuring that only authorized personnel can access, modify, or manage specific data.

7. **Scalability and Performance**: The application architecture is designed to be scalable and capable of supporting a growing dataset as business needs expand. The backend structure and data processing functions are optimized to handle high-volume usage, ensuring smooth operations even under heavy load. The system’s modular design facilitates future enhancements, allowing for easy adaptation to additional features or increased data demands.

## 2.2 Product Functions

1. **Item Management:**

* Add new products to the inventory database with detailed information, including name, description, category, and pricing.
* Edit and update product details to reflect any changes in specifications, descriptions, or categories.
* Remove or deactivate products that are discontinued or out of stock to keep inventory data accurate and current.

2. **Barcode Scanning:**

* Utilize barcode and QR code scanning to efficiently add or update products in the inventory system, streamlining data entry and reducing errors.
* Instantly retrieve product information (name, description, price, etc.) by scanning codes, enabling quick and easy updates.

3. **Inventory Tracking:**

* Monitor each product’s inventory in real-time across multiple warehouses, tracking changes from sales, purchases, or adjustments.
* Generate alerts for low stock levels, enabling proactive inventory management and preventing potential stockouts.
* Record detailed inventory movement history for analysis, including item transfers, adjustments, and returns.

4. **Order Management**:

* Create and manage purchase orders to ensure timely replenishment of inventory based on minimum stock thresholds.
* Process incoming shipments against purchase orders, updating stock levels automatically upon receipt.
* Prepare packing lists and generate shipping labels for outbound orders, streamlining dispatch operations.

5. **User Management**:

* Manage user accounts within the application and define access levels based on organizational roles (e.g., admin, warehouse manager, store manager).
* Implement role-based permissions to control who can view, edit, or approve various operations within the system.
* Track user activity and maintain audit trails for accountability, ensuring transparent and secure management of inventory actions.

6. **Integration with External Systems:**

* Seamlessly integrate with accounting software to synchronize inventory and financial data, simplifying reconciliation and reporting.
* Connect with point-of-sale systems to automatically update inventory levels with each sale, ensuring accuracy.
* Sync with e-commerce platforms for real-time inventory updates, providing accurate product availability across sales channels.

7. **Customization and Configuration**:

* Configure units of measurement, currency settings, tax rates, and other preferences to align with business requirements.
* Set up custom alerts and notifications, such as low stock warnings or order fulfillment reminders, based on user preferences.
* Adapt the application to meet specific industry standards or company workflows, with customization options to suit unique operational needs.

## 2.3 User Characteristics

1. **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.

2. **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.

3. **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.

4. **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.

5. **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.

## 2.4 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.

## 2.5 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.

# 3. Specific Requirements

## 3.1 External Interface Requirements

**3.1.1 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.

**3.1.2 Hardware Interfaces**

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

**3.1.3 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.

**3.1.4 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.

## 3.2 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.

## 3.3 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.

## 3.4 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.

## 3.5 Non-Functional Requirements

3.5.1. **Performance**

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

3.5.2. **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.
  + 1. **Availability**
* Offline functionality for basic operations when the server is unavailable.
* The system should be accessible 24/7 except during scheduled maintenance windows.

* + 1. **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.

* + 1. **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

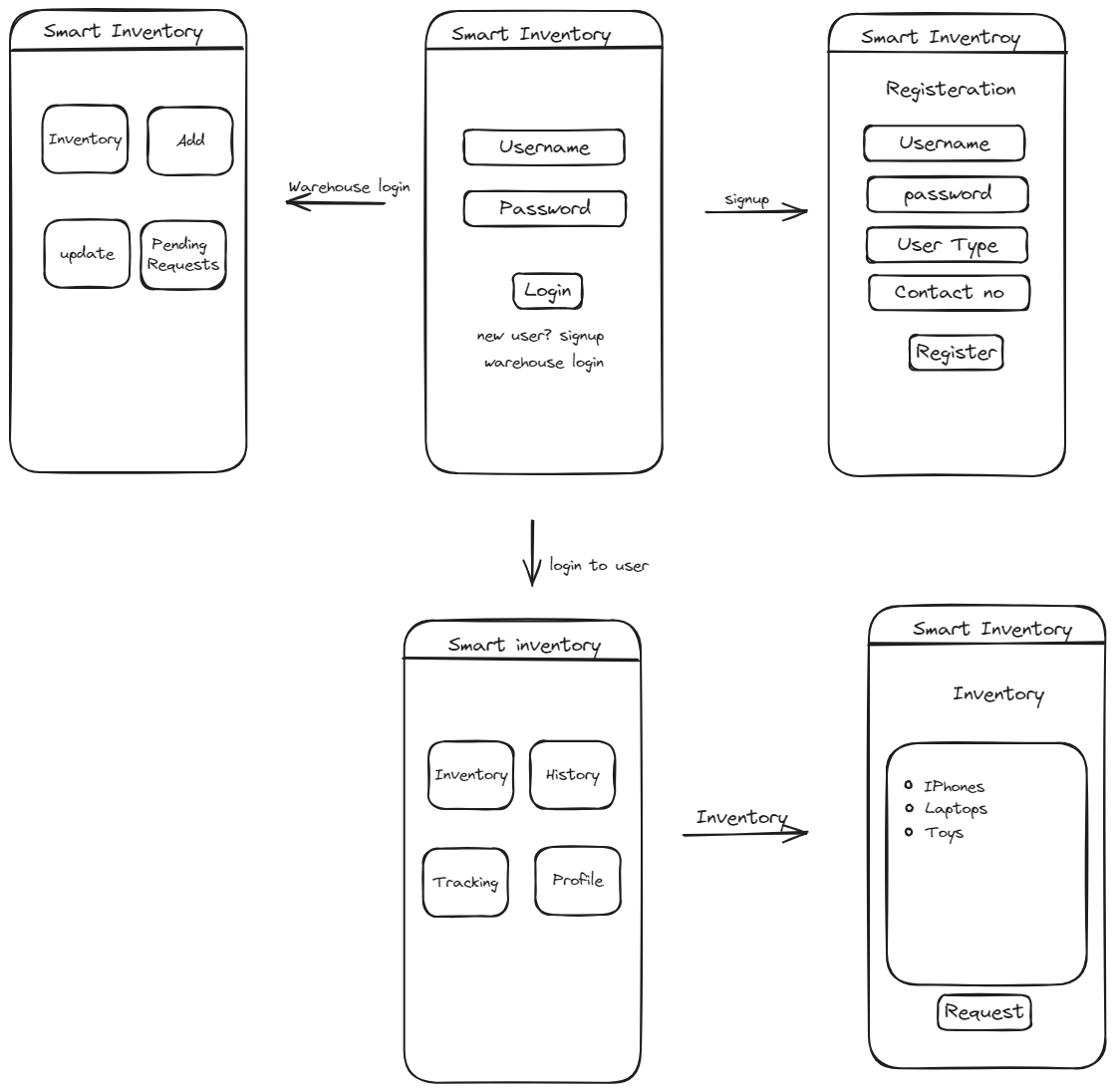
* The database should have tables for storing information about inventory items, orders, users, and transactions.
* Relationships between tables should be defined to enforce data integrity.
* Indexes should be created for efficient querying of large datasets.

## Other Requirements

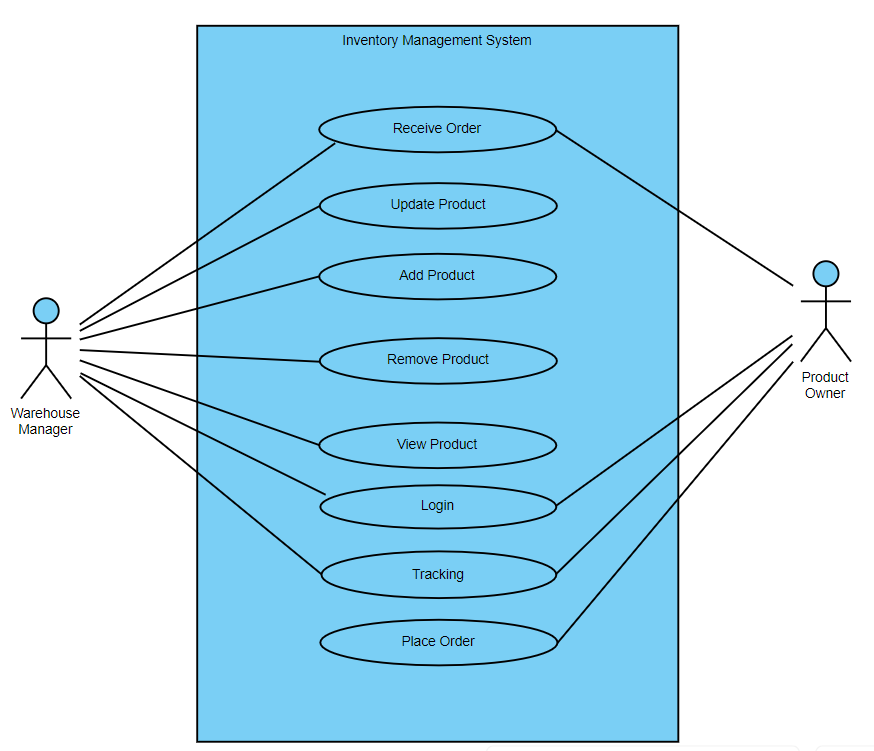
* The application should have a user-friendly onboarding process for new users.
* The application should provide notifications for low inventory levels or pending orders.

## Prototypes (for the complete project)

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

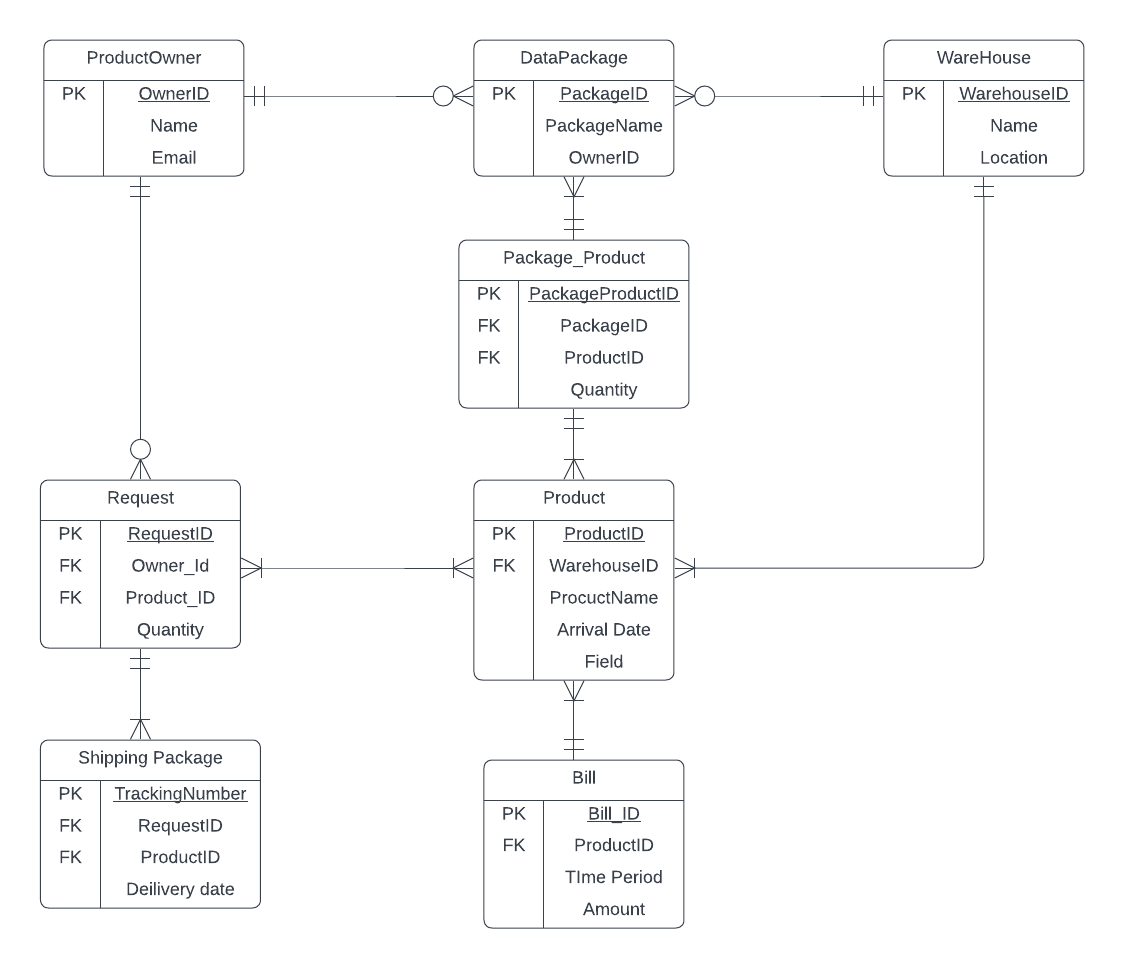


## Use Case Diagrams

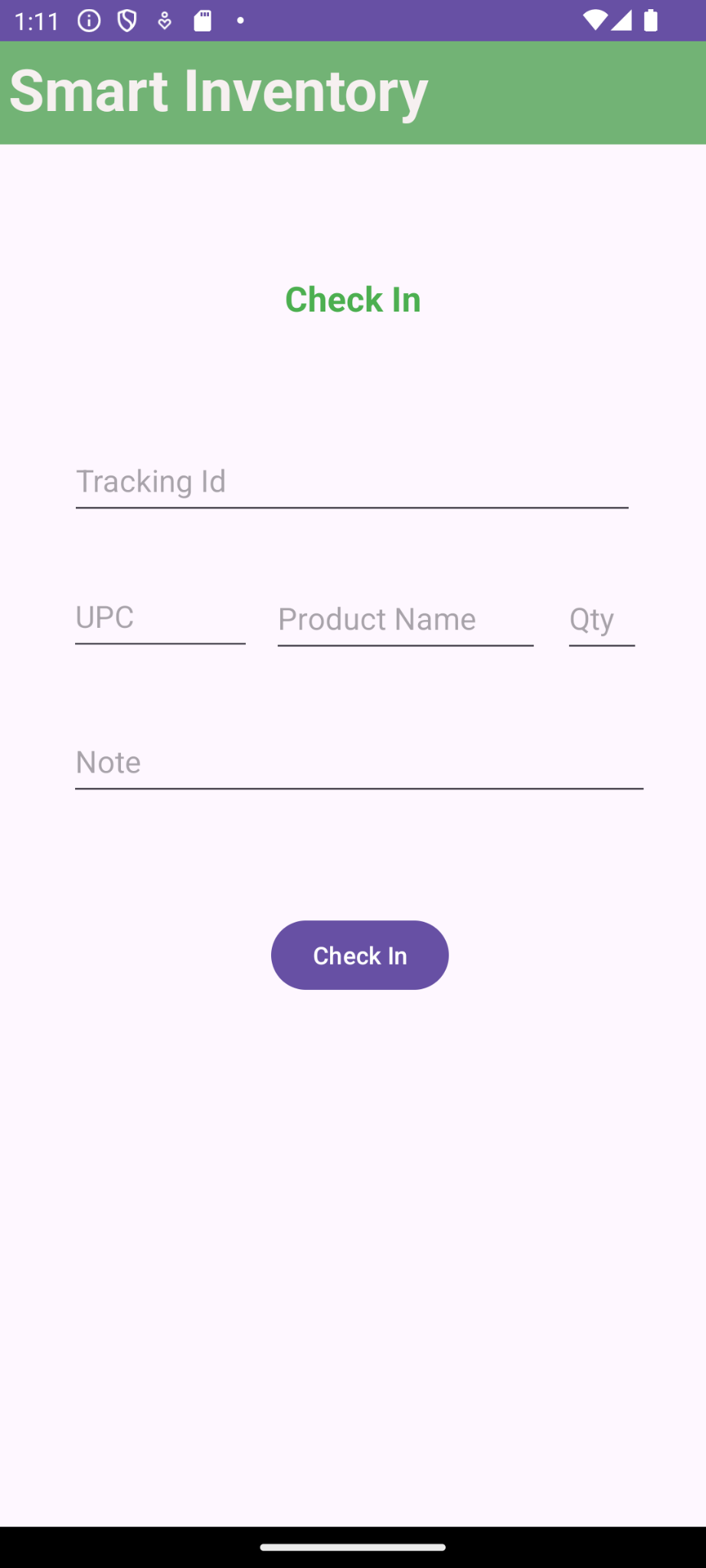
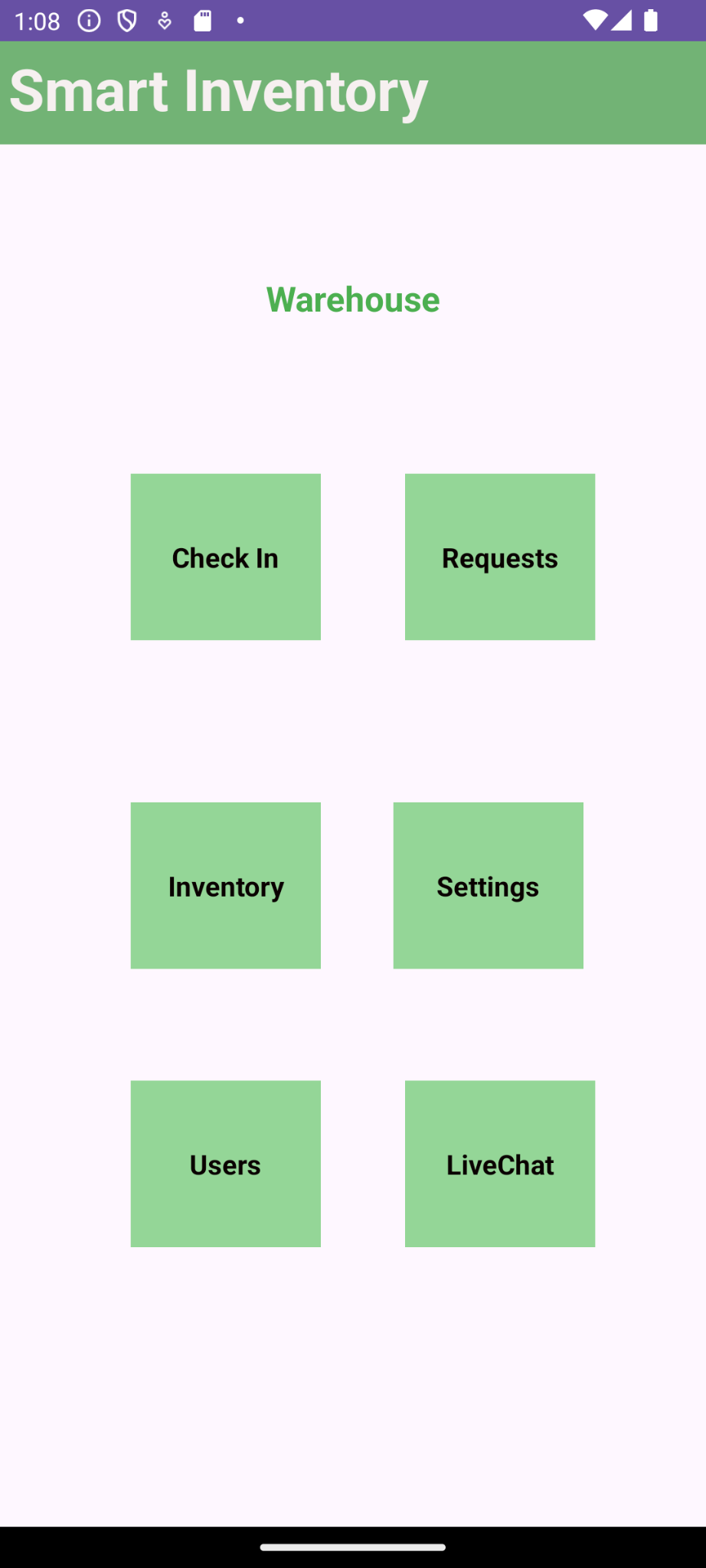
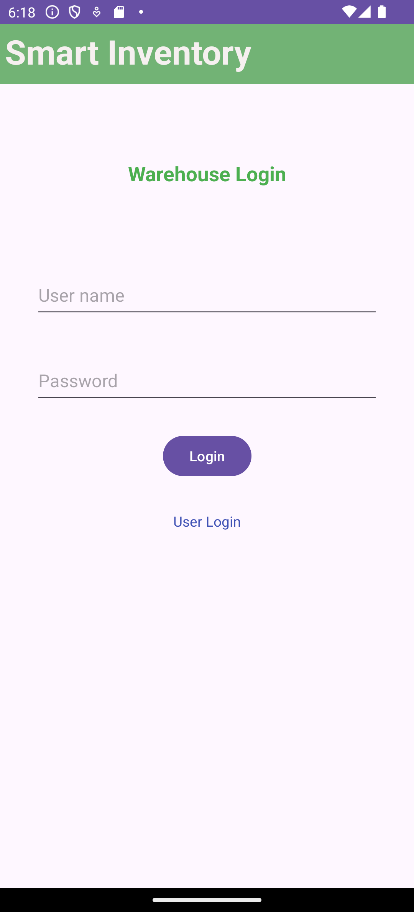
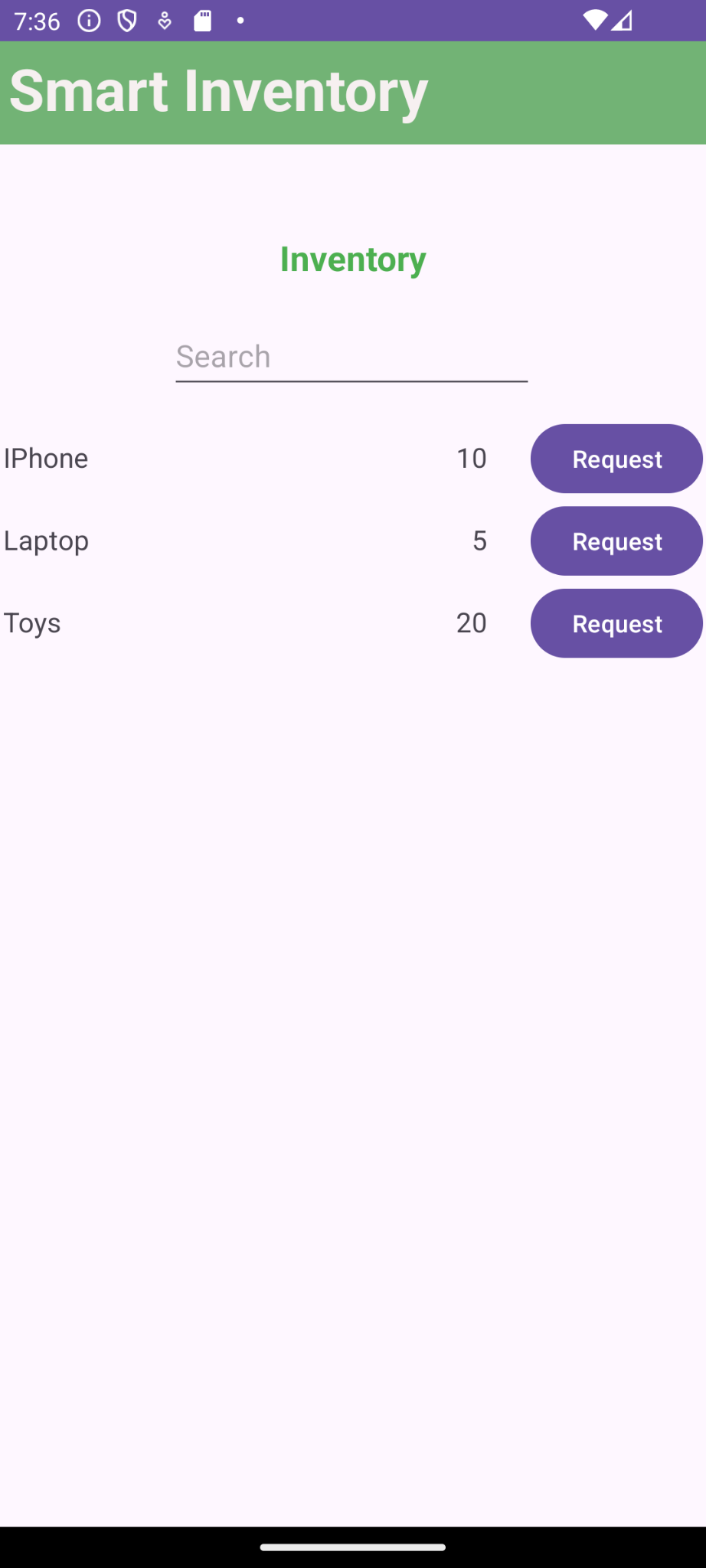
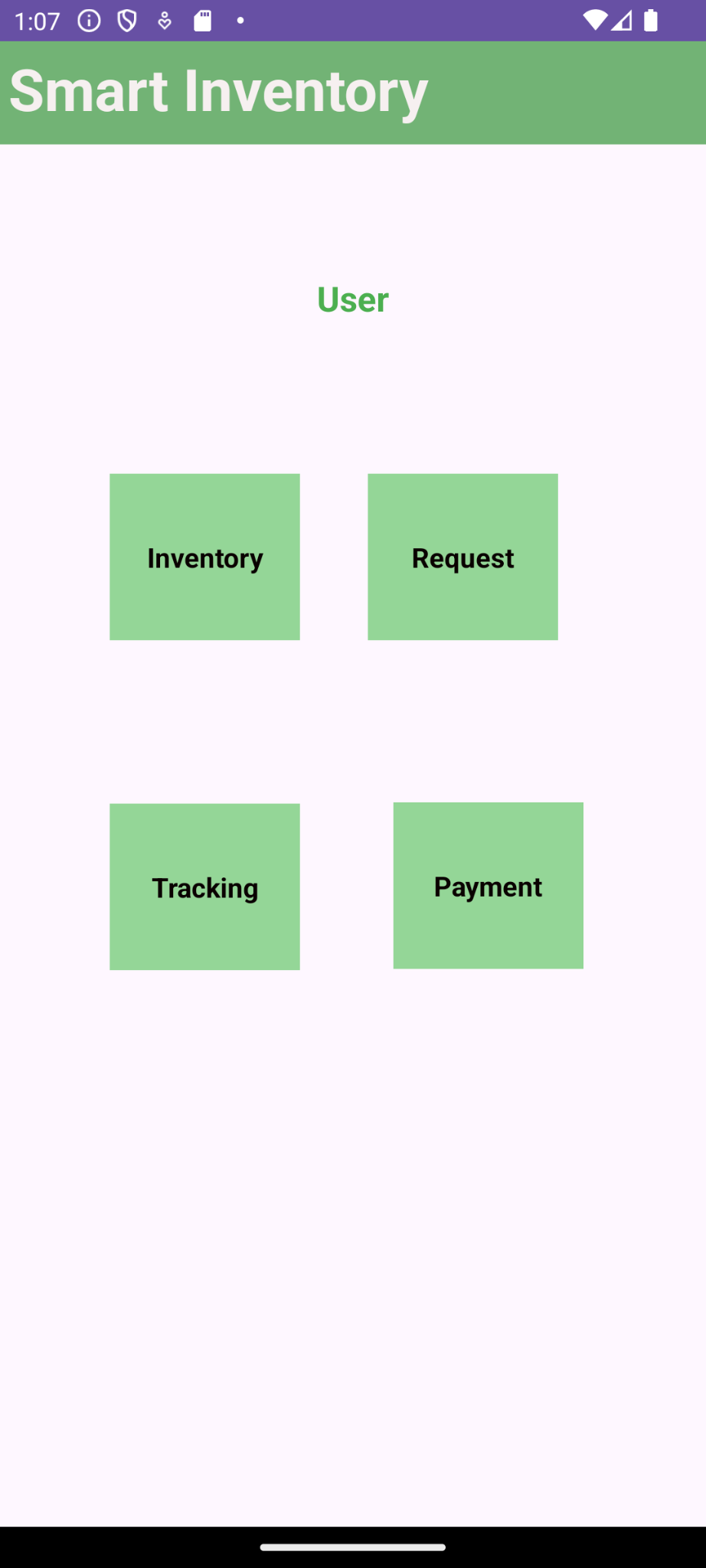
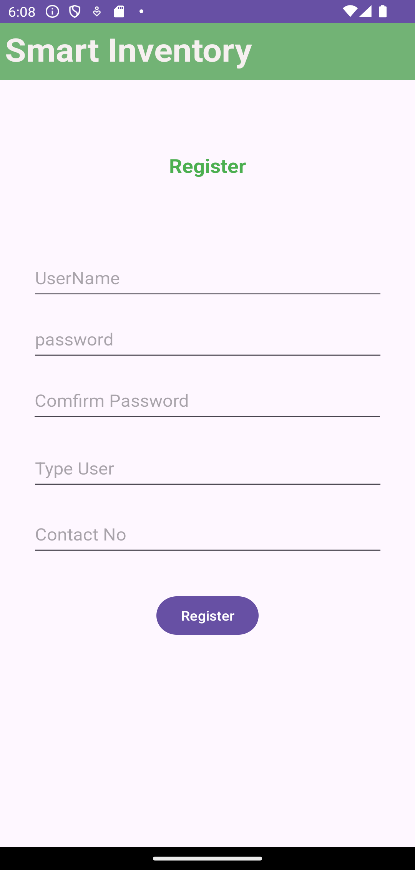


## 4. Design

**4.1. ER diagram**



**4.2. GUI**

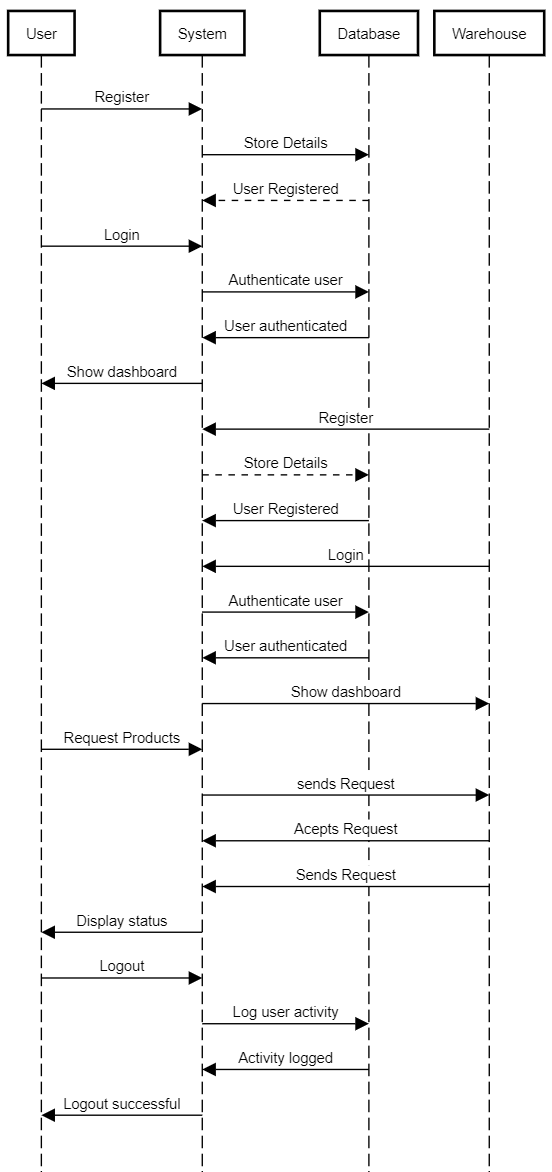


**5. Analysis Models**

**5.1. Data Flow Diagram**

|  |
| --- |
|  |

**5.2. Sequence Diagram**



# 6. User Manual

**6.1 Introduction**

This user manual is designed to guide users through the features and functionalities of the Warehouse Inventory Management System. It covers installation, main features, user and manager interfaces, and troubleshooting.

**Installation**

1. Download the APK file from the release section.

2. Install the APK on your Android device.

3. Ensure that you have internet access for Firebase functionalities, as connectivity is required for database and notification features

**Features**

**Login**

1. Launch the app and navigate to the login page.

2. Enter your credentials and click "Login."

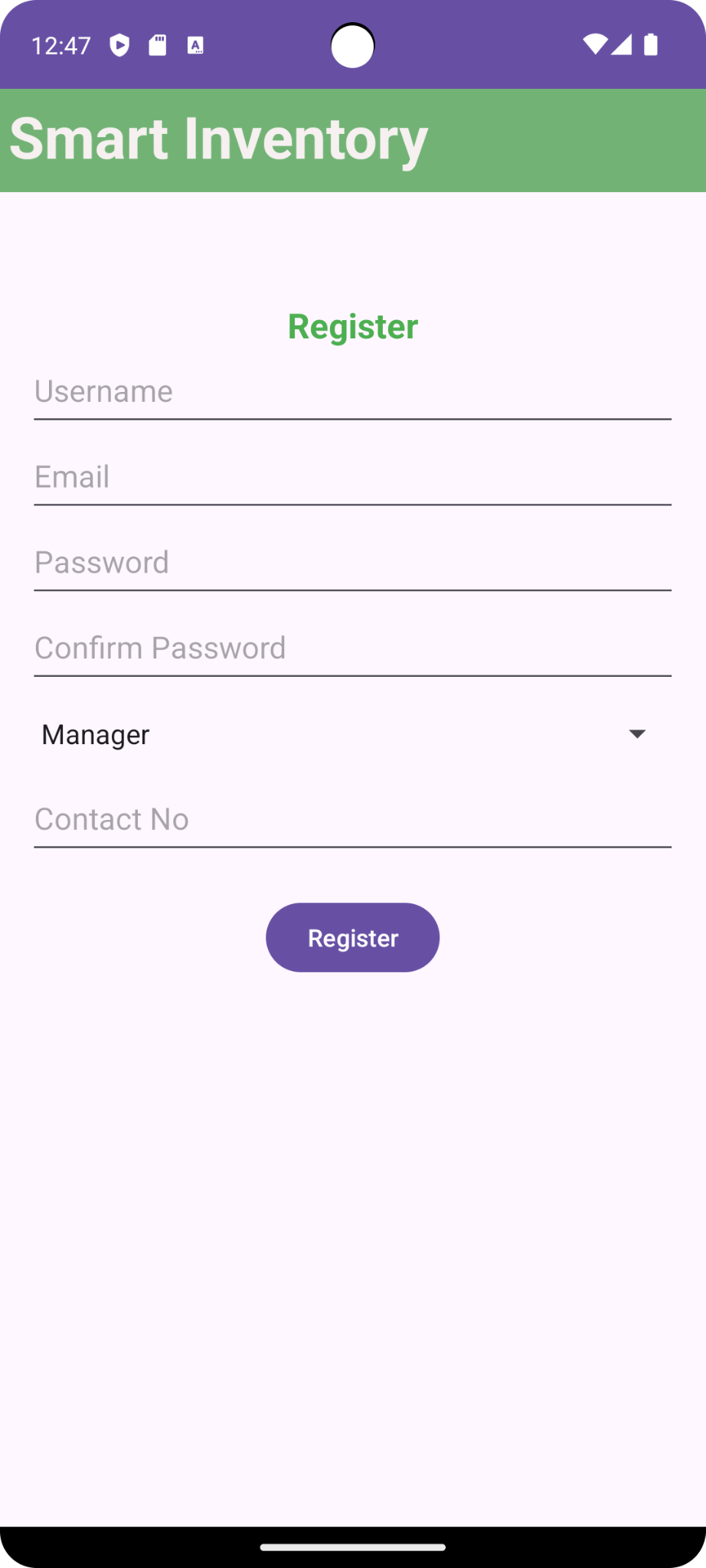
3. If you do not have an account, use the "Register" option.

**Registration**

1. Navigate to the registration page.

2. Fill in the required information and submit the form.

3. You will receive an email verification link. After verification, you can log in to the application.



**Reset Password**

1. If you forget your password, navigate to the "Reset Password" section.

2. Enter your email address and follow the instructions to reset your password.

**6.2 Warehouse Manager Interface**

**Warehouse Management**

1. View a list of usernames representing users in the warehouse.

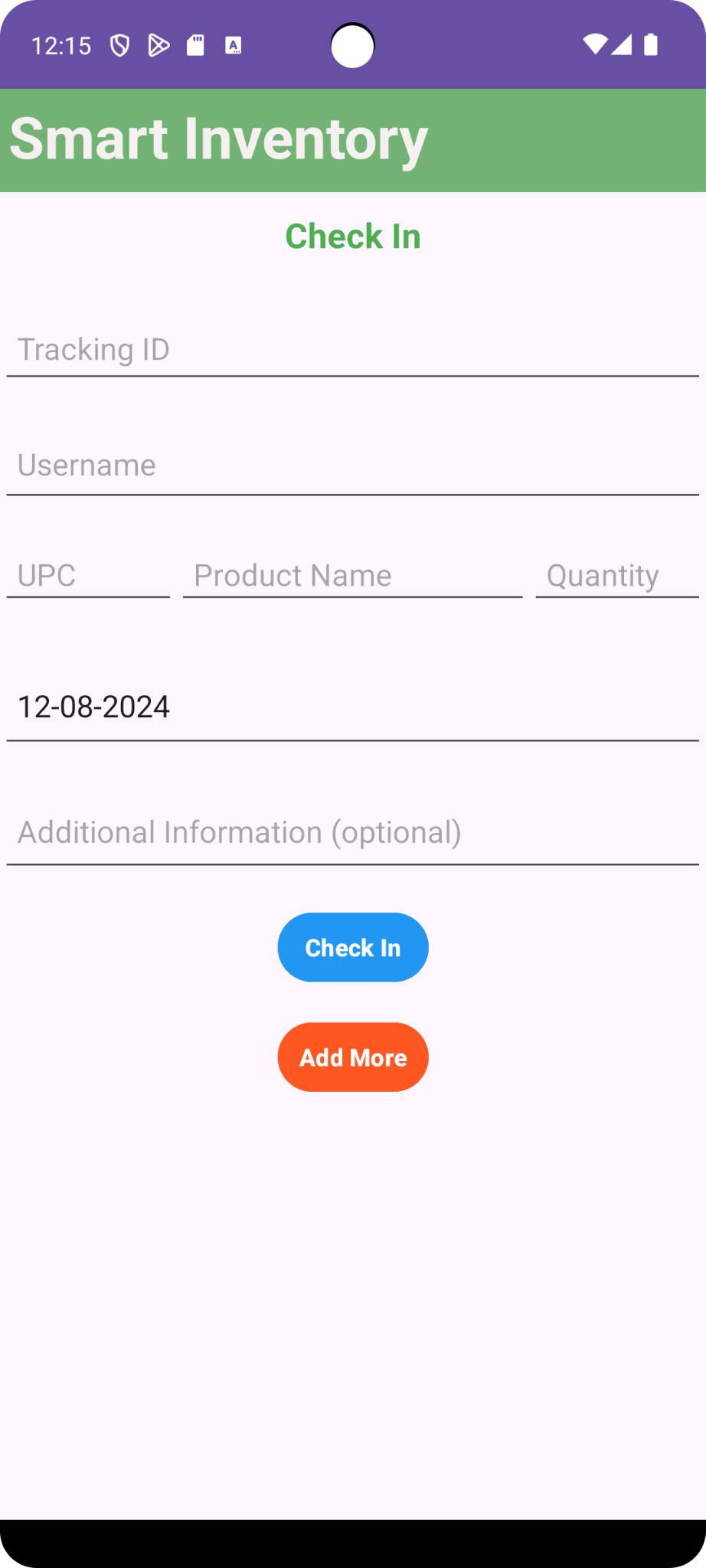
2. Select a username to view that user’s corresponding inventory.

**Check-In**

1. Enter the details for new packages arriving at the warehouse, including Tracking ID, Username, UPC, Product Name, and Quantity. The Date will be fetched automatically.

2. To add multiple products to the same check-in, use the "Add More" button.

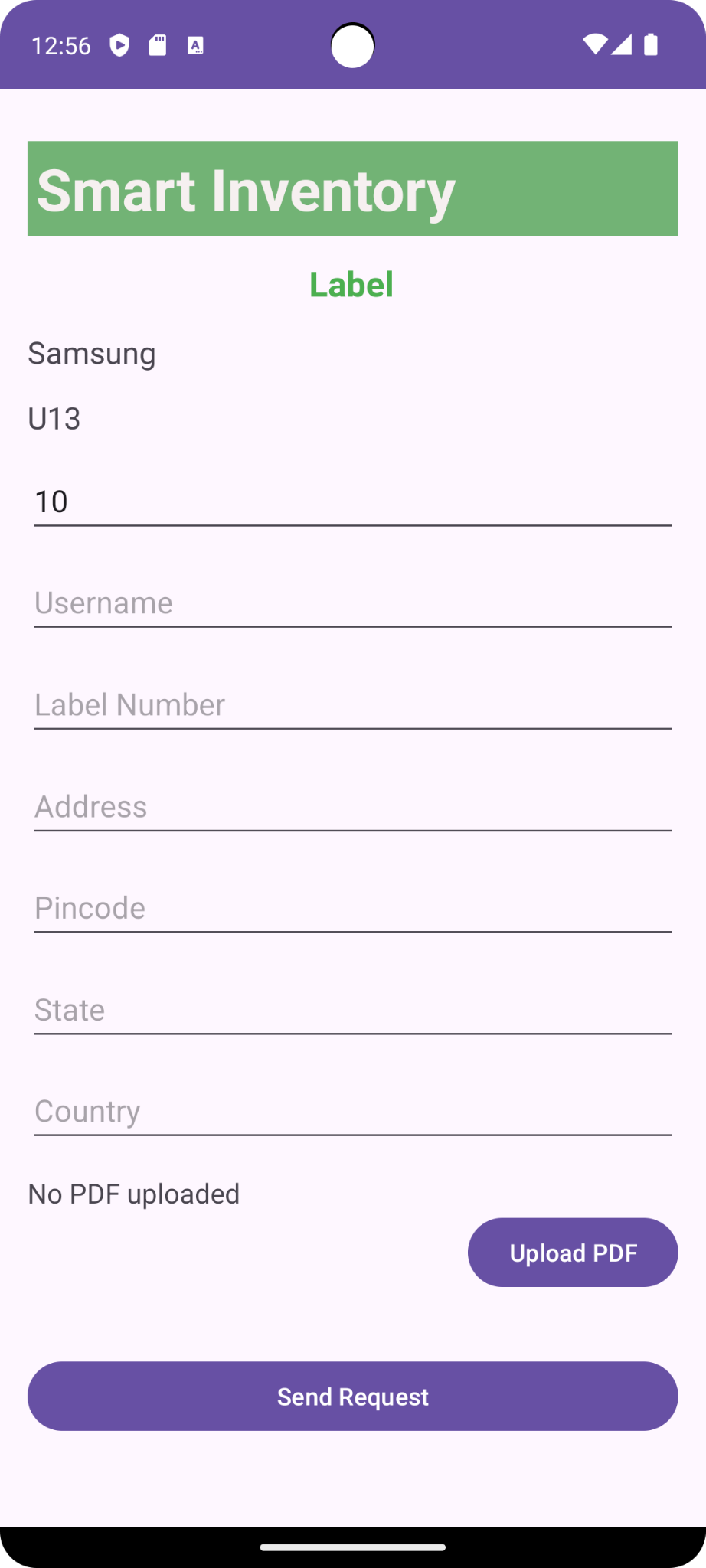
3. Once the check-in is submitted, the data will be stored and reflected in both the user’s and warehouse’s inventory.



**Request Management**

1. View and manage requests related to inventory and packages.

2. User requests appear in the warehouse manager's request interface, where requests can be moved to "Pending" or "Completed."



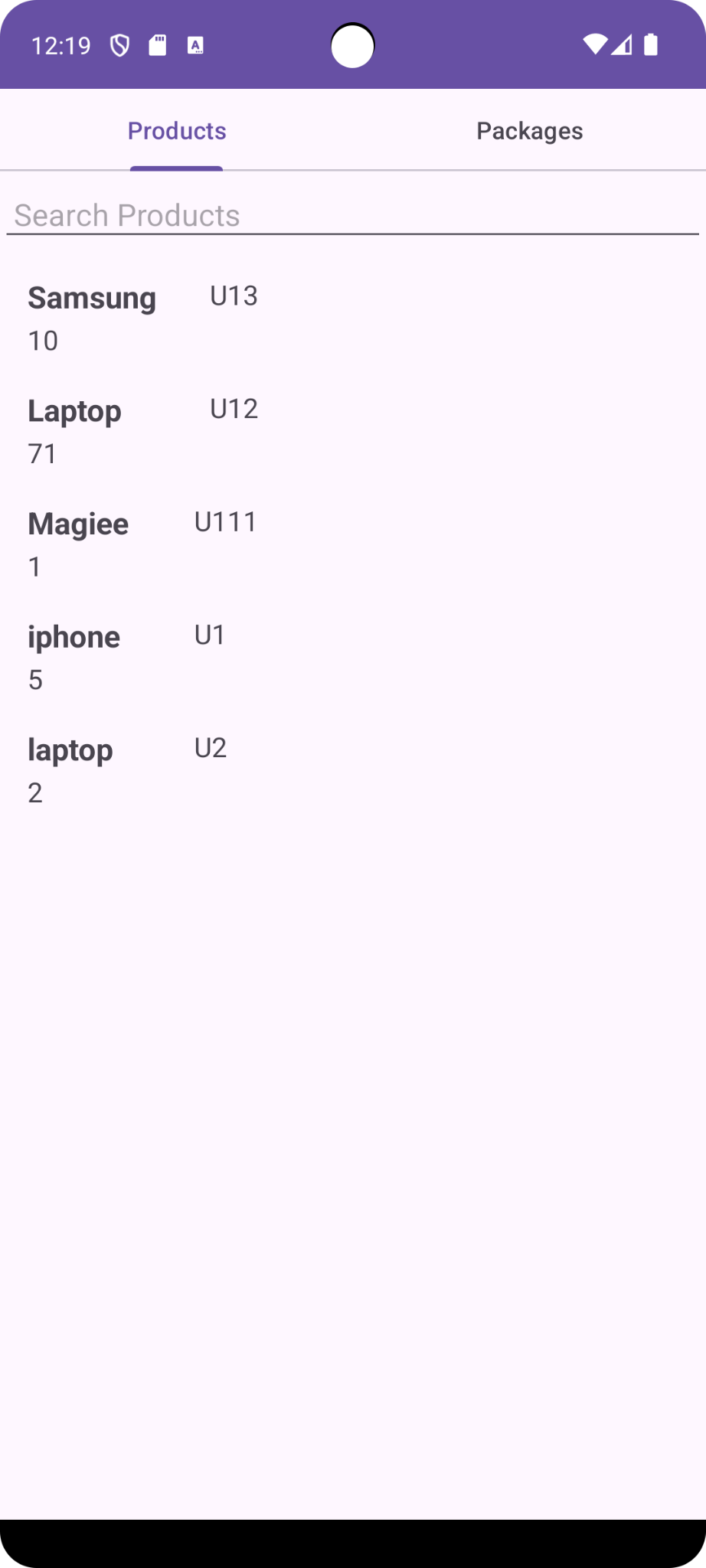
**Inventory Management**

1. Access and manage inventory items for various users.

2. View an organized list of users. Select a user to access two tabs:

- Products Tab: View all checked-in products for that user. Select a product to view its deduction history.

- Packages Tab: View all package IDs for that user. Selecting a package displays the products within it.

 → 

**Live Chat**

1. Communicate with users through the live chat feature.

2. The chat interface shows a list of users. When a user sends a message, a notification and a "new message" indicator will appear next to the user's name.



**6.3 User Interface**

**User Inventory Management**

1. Access and manage your inventory with two tabs: \*\*Products\*\* and \*\*Packages\*\*.

- Products Tab: View all checked-in products for your account. Each product has a "Request" button to request additional action on the product.

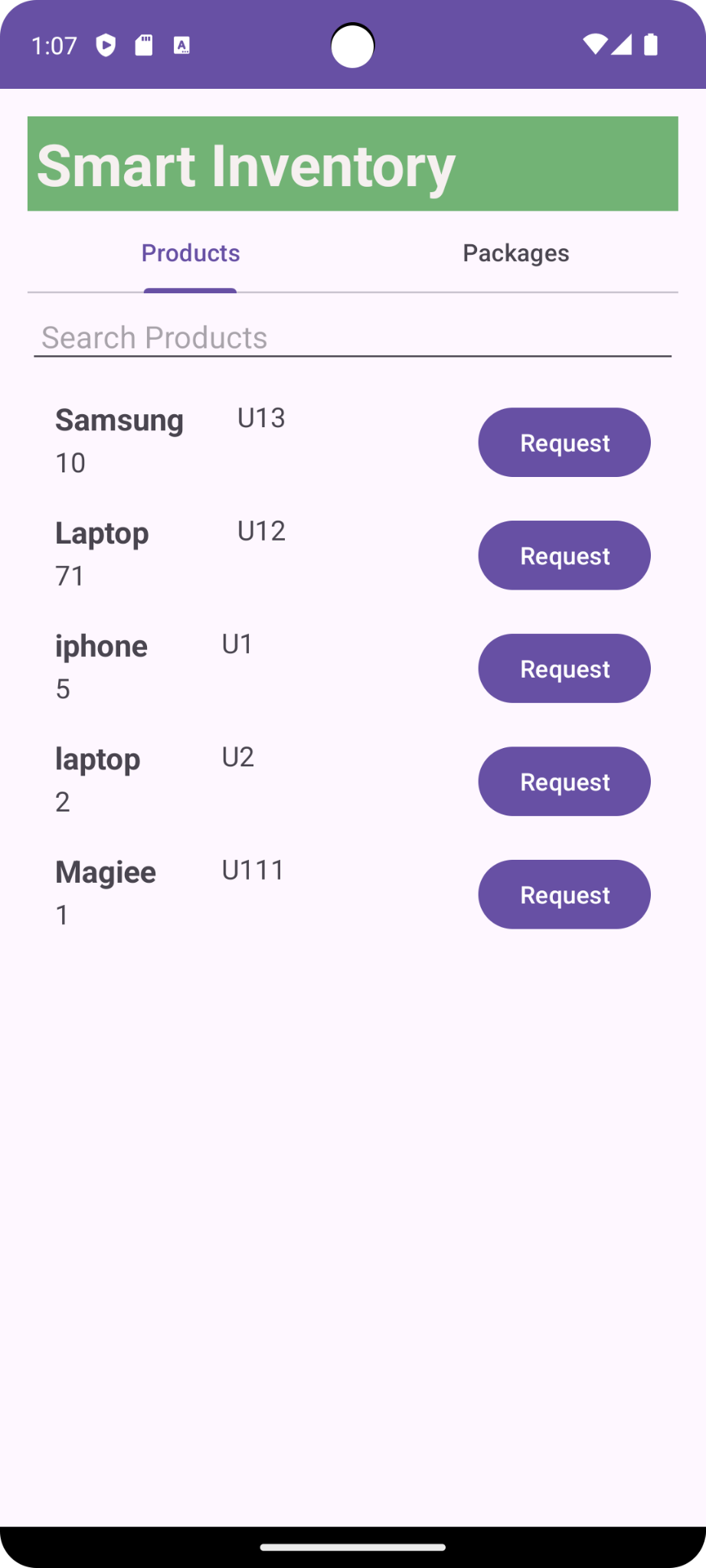
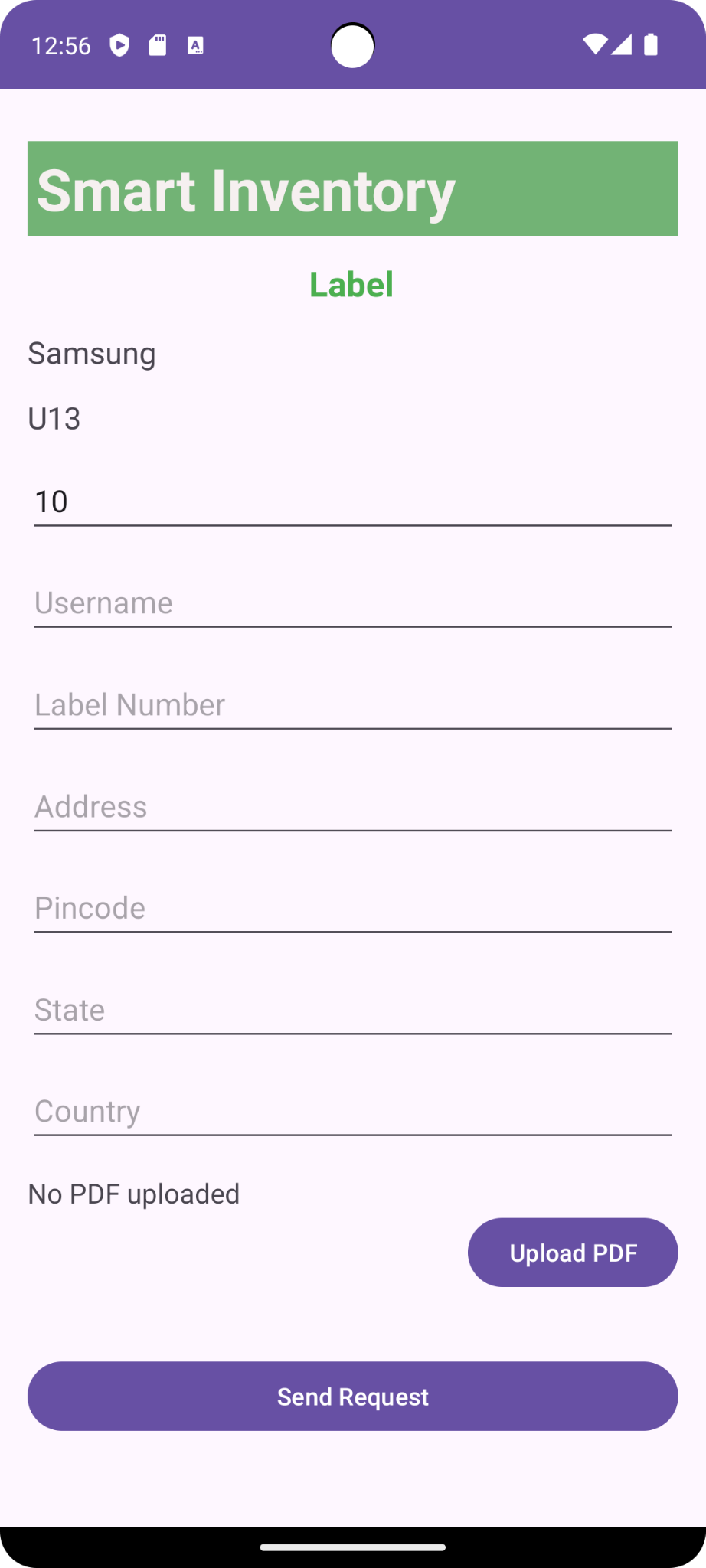
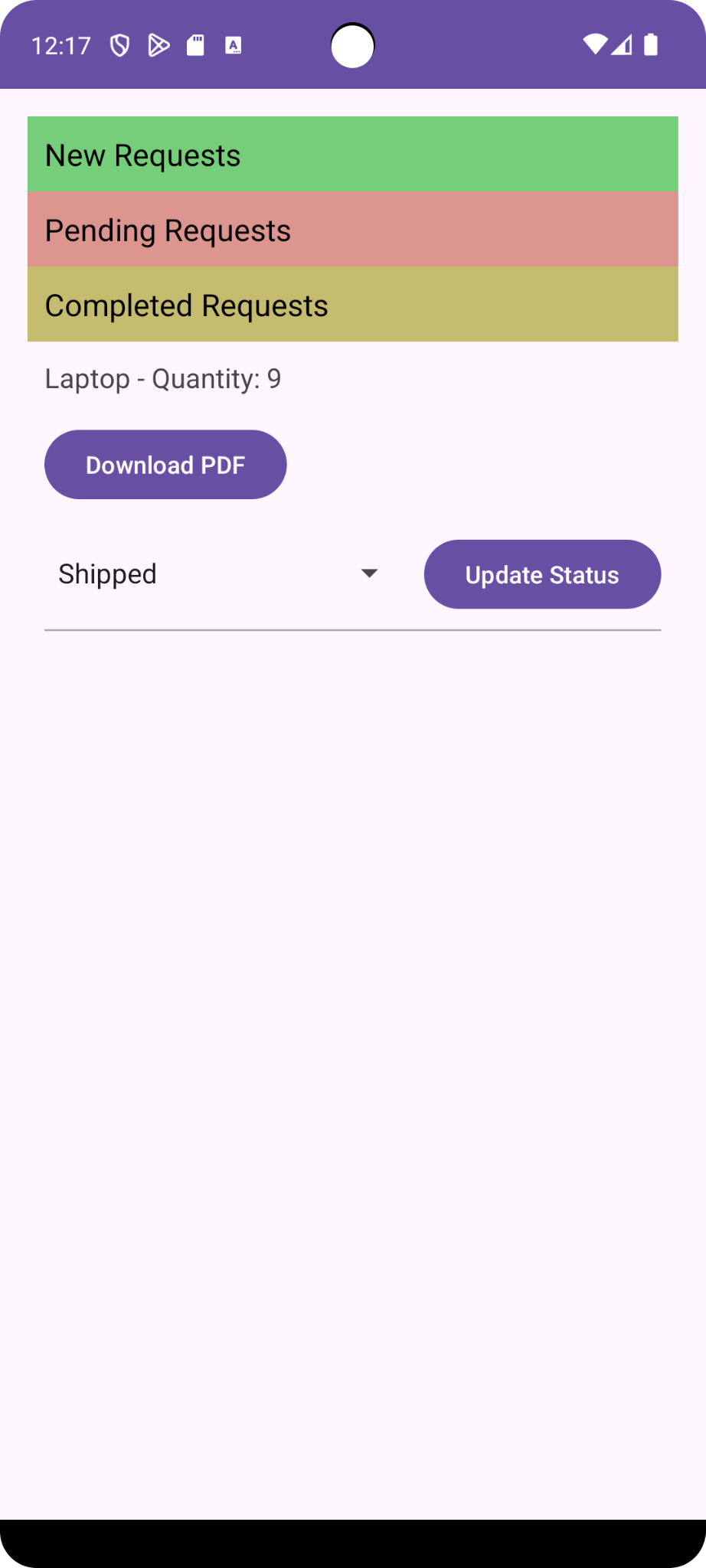
- Packages Tab: View all package IDs. Select a package to see its products, along with a "Request" button to raise a request for that package.

**Request**

1. Select the "Request" button from the inventory tab to open the request page.

2. Upload a label; details will be filled automatically.

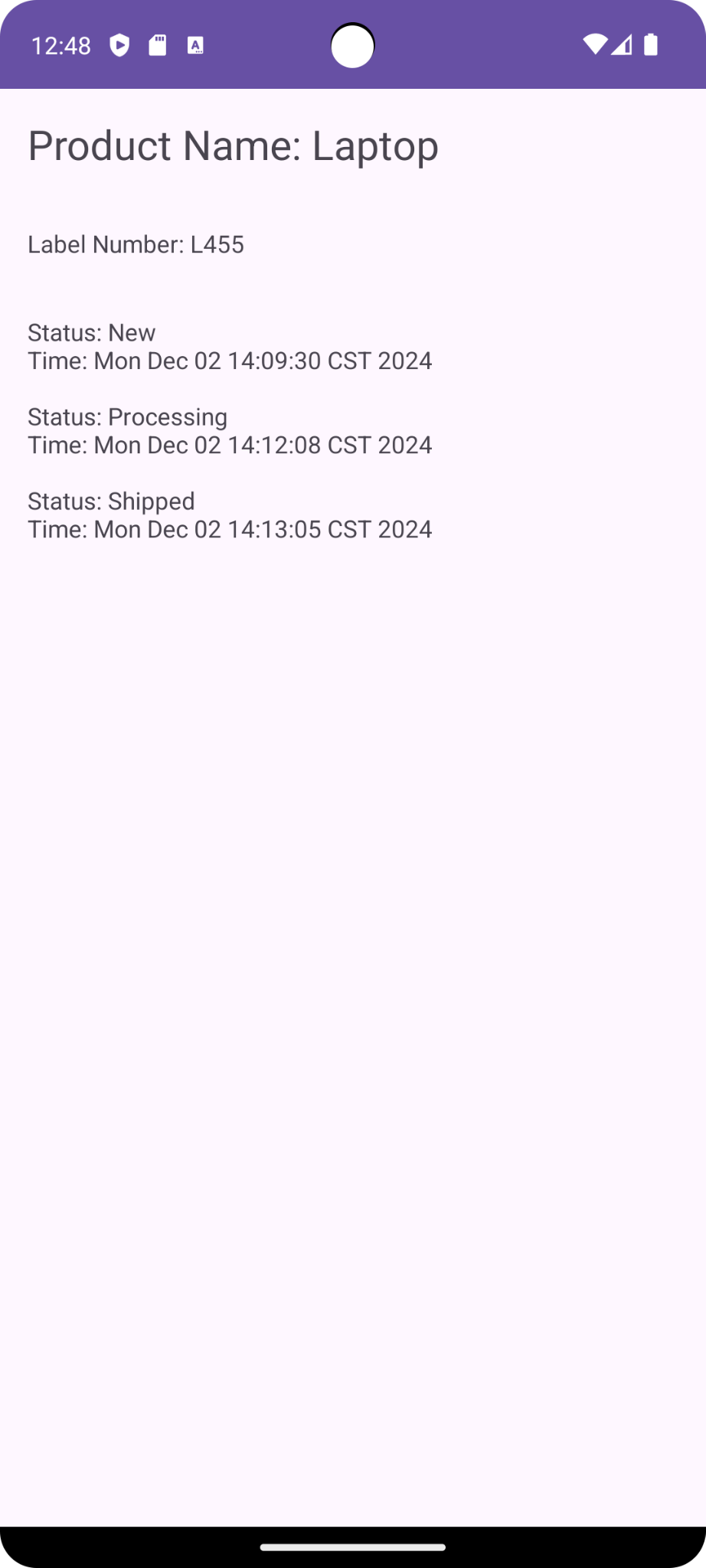
3. Click "Send Request" to submit it. The request will be visible in the warehouse manager’s request page.

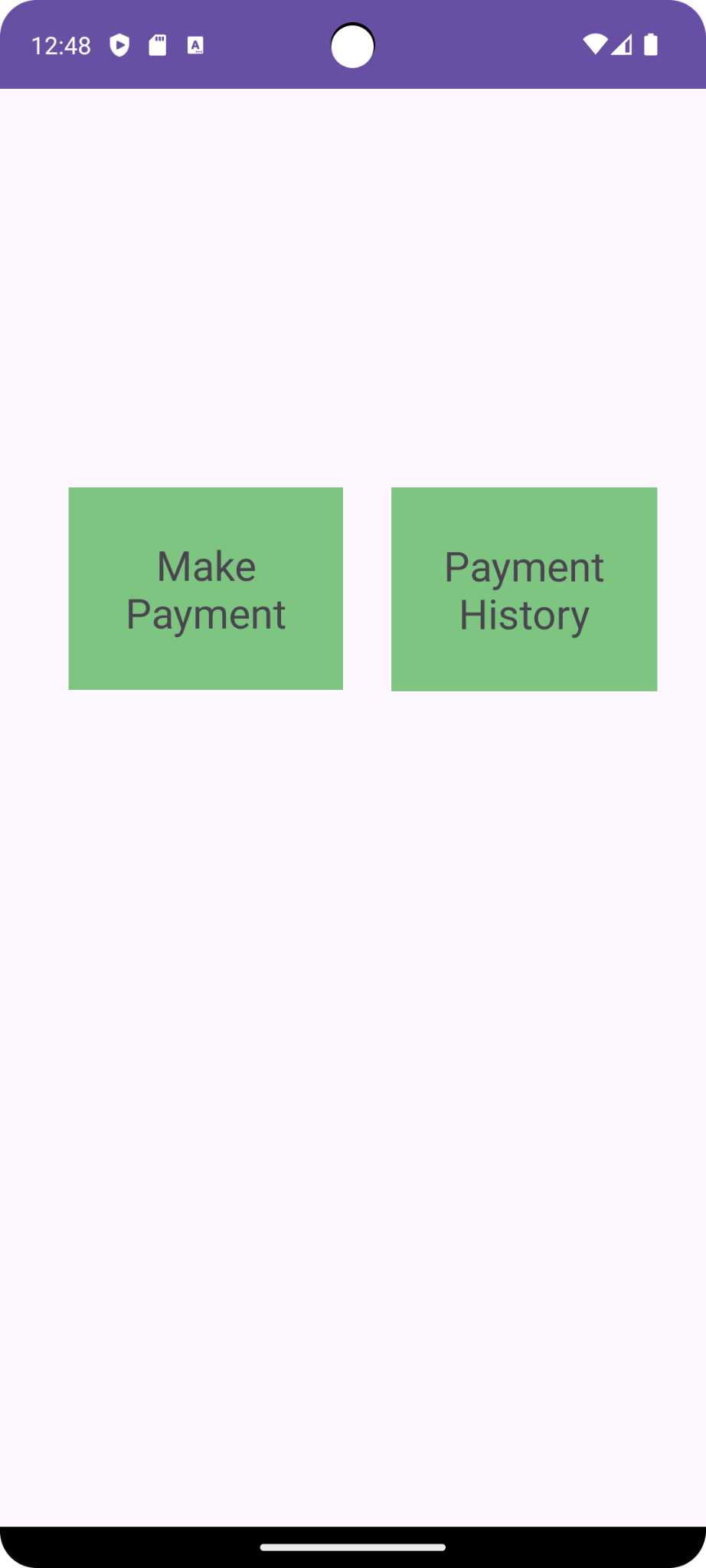
**Package Tracking**

1. View and manage packages using the package tracking interface.

2. All label numbers with product names are listed. Selecting one allows you to track the product's location within the warehouse.

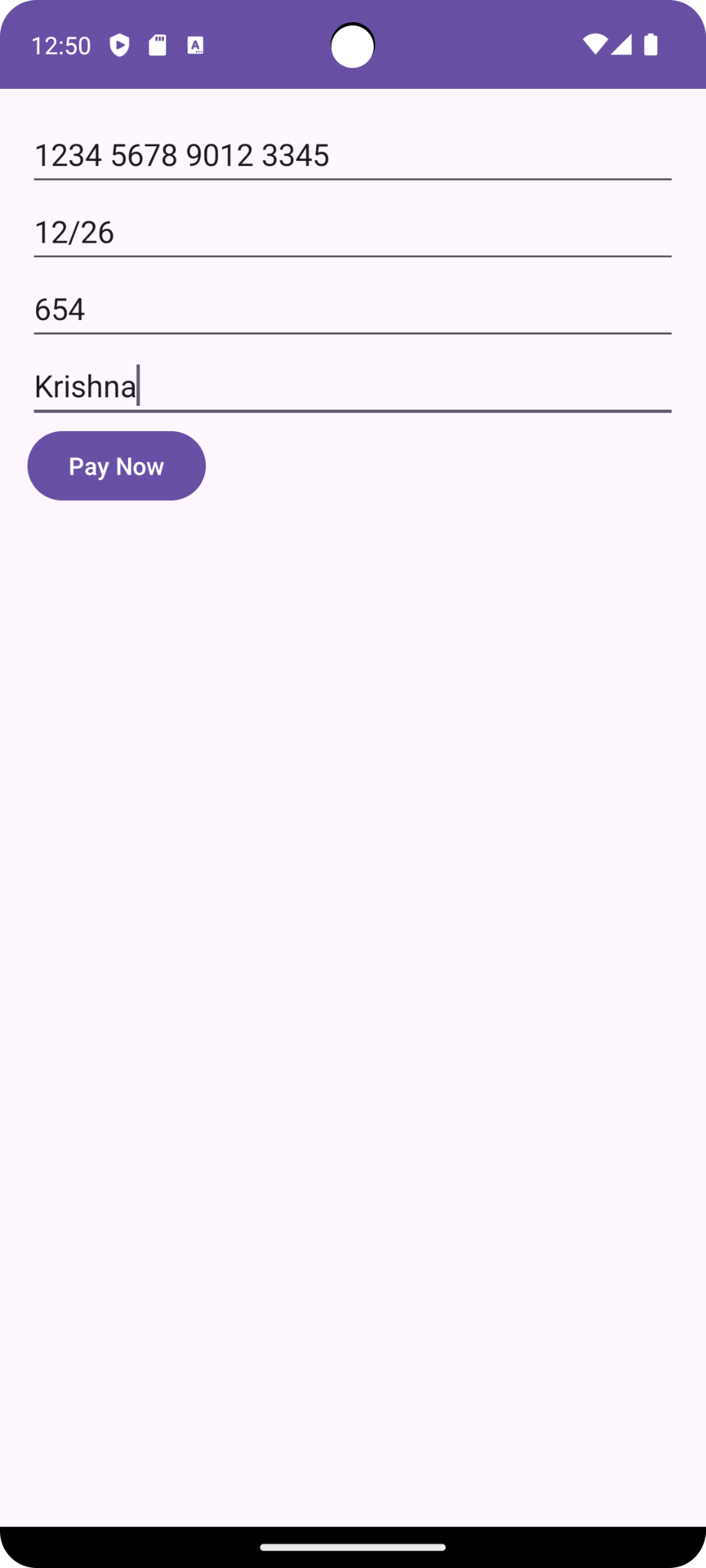


**Payment**



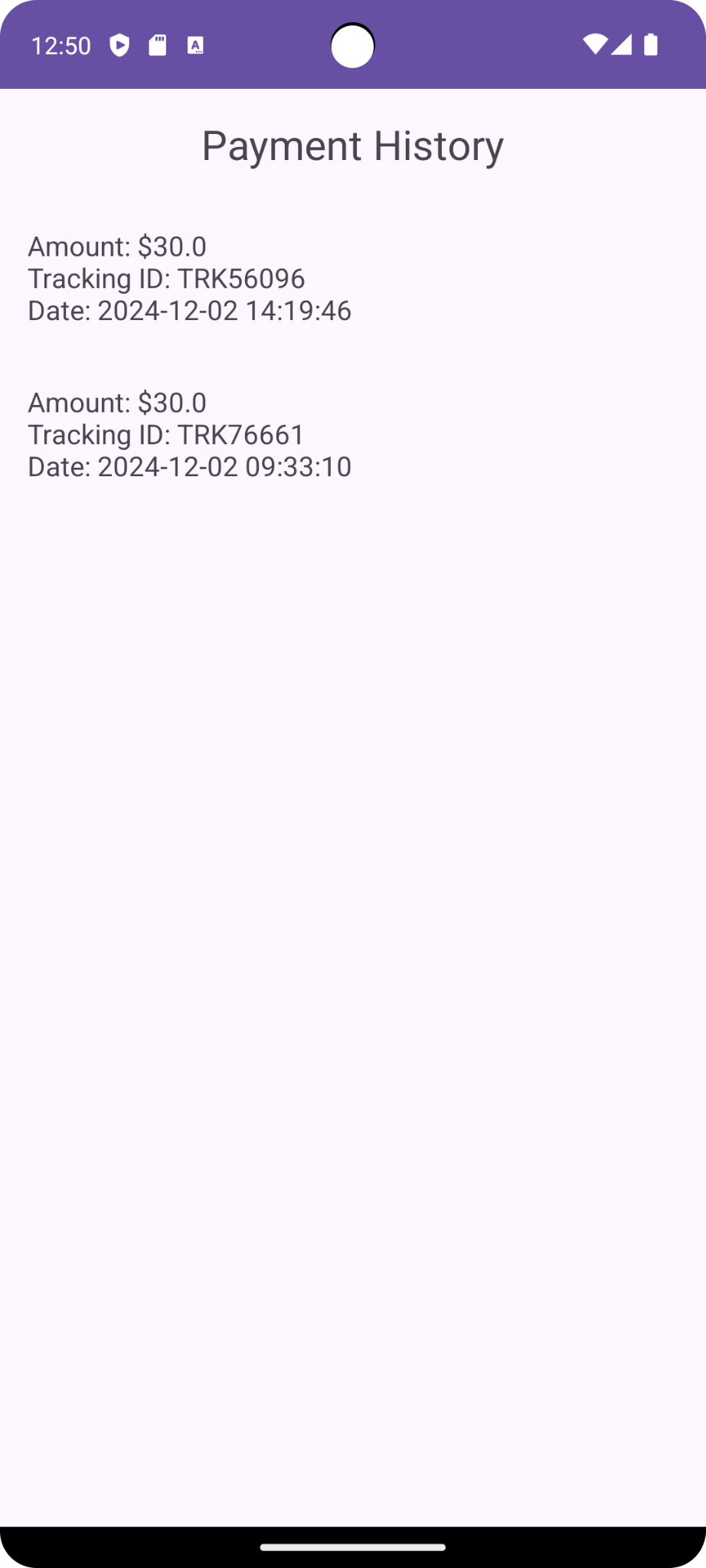
**1. Make Payment:**

- View the total amount, including check-in, storage, and checkout fees.

- Click the "Pay" button to proceed to a payment page where you can enter card details and complete the payment.  

**2. Payment History:**

- View a list of past payments, showing transaction IDs along with dates and times.



**Live Chat**

1. Communicate with warehouse personnel through the live chat feature.

2. The interface lists managers and offers a chat option. If a manager sends a message, a notification and a "new message" indicator will appear next to the manager’s name.

**Troubleshooting**

* **Login Problems:** Verify credentials and check network connectivity.
* **Data Syncing Errors:** Restart the app and ensure internet access for real-time updates.
* **Internet Connectivity:** Ensure your device is connected to the internet.
* **Performance Issues:** Restart the application if you experience delays or performance issues.
* **Firebase Issues:** If Firebase functionalities (like chat or inventory updates) aren't working, check your connection and log back in.

**7. Technical Manual**

**Overview**

The Warehouse Inventory Management System is designed for warehouse managers and users to manage inventory, check-ins, requests, payments, package tracking, and communication via live chat. The application leverages Android SDK, and Firebase for backend services (authentication, database, notifications), and provides a seamless user interface for efficient management.

**System Components and Architecture**

The system is divided into various modules: Login, Registration, Warehouse Management, Inventory Management, Requests, Payment Processing, Live Chat, and Tracking. Each module is composed of Java classes (backend) and XML layout files (frontend). Below is a detailed breakdown of the components:

**1. Authentication**

**Java Files:**

• MainActivity.java: Handles user login using Firebase Authentication.

• Register.java: Manages the registration process, creating a new user account.

• ResetPasswordActivity.java: Facilitates password resetting through email-based instructions.

**XML Files:**

• activity\_main.xml: Layout for the login page where users enter credentials.

• activity\_register.xml: Layout for the registration page.

• Activity\_reset\_password.xml: Layout for resetting the password

**2. Warehouse Inventory Management**

**Java Files:**

• WarehouseInventoryActivity.java: Displays a list of all registered usernames and allows the warehouse manager to select users to view their inventory.

• WarehouseItem.java: Represents an individual item in the warehouse, including product details such as name, quantity, etc.

• WarehouseAdapter.java: Adapter to populate the warehouse inventory list with data.

• UsernameAdapter.java: Adapter for displaying usernames in the list.

• UserInventoryActivity.java: Displays the inventory for a specific user when selected from the warehouse list.

**XML Files:**

• activity\_warehouse\_inventory.xml: Layout to show the list of usernames in the warehouse.

• username\_item\_layout.xml: Defines the layout for displaying individual usernames in the RecyclerView.

**3. Warehouse Manager Interface**

**Java Files:**

• warehouseInterface.java: Manages the overall interface for the warehouse manager to handle inventory operations, requests, and check-ins.

**XML Files:**

• Activity\_warehouse\_interface.xml: The layout for the warehouse manager's main interface.

**4. Check-In Management**

**Java Files:**

• checkIn.java: Handles the process of checking in new items (products) into the warehouse.

**XML Files:**

• Activity\_check\_in.xml: Layout for adding products to the warehouse when a new shipment arrives. Includes fields for tracking, product details, and quantity.

**5. Requests Management**

**Java Files:**

• Request.java: Handles the creation and management of requests from users or warehouse managers.

• RequestDetails.java: Displays detailed information about specific requests.

**XML Files:**

• Activity\_request\_item.xml: Layout for individual request items.

• Request\_item.xml: Defines the layout for displaying request information, including a spinner for managing request statuses.

• Activity\_request.xml: Main layout for managing all requests.

**6. Warehouse Products and Packages**

**Java Files:**

• WarehousePackagesAdapter.java: Adapter for displaying packages in the warehouse.

• WarehouseProductsAdapter.java: Adapter for displaying products in the warehouse.

• WarehousePackagesFragment.java: Fragment for managing packages in the warehouse.

• WarehouseProductsFragment.java: Fragment for managing products in the warehouse.

• WarehouseProductsListActivity.java: Activity for listing products in the warehouse.

**XML Files:**

• Activity\_product\_list.xml: Displays a list of products in the warehouse.

• activity\_warehouse\_packages\_fragment.xml: Layout for managing warehouse packages.

• activity\_warehouse\_products\_fragment.xml: Layout for managing warehouse products.

• fragment\_warehouse\_packages.xml: Fragment layout for displaying package-related information.

• fragment\_warehouse\_products.xml: Fragment layout for displaying product-related information.

• Warehouse\_inventory\_item.xml: Defines the layout for inventory items in the warehouse.

• Activity\_username\_list.xml: Displays the list of usernames in the warehouse.

• Activity\_warehouse\_inventory.xml: Main activity layout for the warehouse inventory interface.

• Username\_item\_layout.xml: Layout for each username in the list.

**7. Live Chat**

**Java Files:**

• ChatActivity.java: Manages live chat functionality between warehouse personnel and users.

• ChatAdapter.java: Adapter for displaying chat messages.

• ChatMessage.java: Represents a single chat message.

• MyFirebaseMessagingService.java: Handles push notifications for new chat messages via Firebase Cloud Messaging.

**XML Files:**

• Activity\_chat.xml: Layout for the main chat interface.

• Item\_chat\_message.xml: Layout for each chat message.

• Item\_chat\_message\_left.xml: Layout for messages sent by the other party (left side).

• Item\_chat\_message\_right.xml: Layout for messages sent by the current user (right side).

• Item\_user\_list.xml: Layout for displaying the user list with the option to start a chat.

**8. User Inventory Interface**

**Java Files:**

• userInterface.java: Manages the user interface for inventory management and requests.

**XML Files:**

• Activity\_user\_Interface.xml: Layout for the user interface, including inventory details and actions.

**9. Inventory Management**

**Java Files:**

• Inventory.java: Core class for managing inventory items for the users.

• InventoryAdapter.java: Adapter for displaying inventory items.

• InventoryItem.java: Represents each inventory item.

• ProductAdapter.java: Adapter for products.

• PackagesFragment.java: Fragment for managing packages.

• ProductsFragment.java: Fragment for managing products.

**XML Files:**

• Activity\_inventory\_item.xml: Defines the layout for individual inventory items.

• fragment\_products.xml: Layout for the products tab.

• fragment\_packages.xml: Layout for the packages tab.

• Item\_layout.xml: Defines the layout for inventory items with request buttons.

• Item\_tracking\_id.xml: Layout for displaying tracking IDs of items.

**10. Request Management in User Interface**

**Java Files:**

• RequestDetailsActivity.java: Manages the request details and request submission from the user.

**XML Files:**

• Activity\_request\_details.xml: Layout for displaying request details.

• Product\_quantity\_row.xml: Layout for displaying quantity information in a request.

**11. Package Tracking**

**Java Files:**

• ProductTrackingDetailActivity.java: Provides detailed tracking information for a product.

• TrackingActivity.java: Displays package tracking details and allows the user to track products within the warehouse.

**XML Files:**

• Activity\_product\_tracking\_detail.xml: Layout for detailed tracking of a specific product.

• Activity\_tracking.xml: Layout for displaying tracking information for packages.

**12. Payment Processing**

**Java Files:**

• Payment.java: Core class for handling payment calculations and processing.

• PaymentOptionsActivity.java: Displays available payment options to the user.

• PaymentPageActivity.java: Manages the payment page where users enter payment details.

• StoragePaymentCalculator.java: Handles the calculation of storage and checkout fees.

• CardPaymentActivity.java: Manages card payment processing.

• PaymentHistoryActivity.java: Displays payment history to the user.

**XML Files:**

• Activity\_payment\_history.xml: Layout to display payment history.

• Item\_payment\_history.xml: Defines the layout for individual payment history entries.

• Activity\_payment\_options.xml: Layout for displaying payment options.

• Activity\_payment\_page.xml: Layout for the payment page.

• Activity\_card\_payment.xml: Layout for entering card payment details.

**Firebase Integration**

• Authentication: Firebase handles the login, registration, and password reset functionalities.

• Firestore Database: Stores inventory data, requests, tracking information, and chat messages.

• FCM Notifications: Enables real-time notifications for new messages in the chat feature.