Software Requirements Specification Template

**Smart Inventory**

Software Requirements Specification

1.0

5/14/2024

Harini Beeram

Santhosh Reddy Gurram

Skanda Shreekar Sira Prabhakar

Sai Teja Nakka

Rakshitha Boddireddy

Submitted in partial fulfillment

Of the requirements of

CSIS 44-691 Graduate Directed Project 1

# **Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
| 2/5/2024 | 1.0 | Santhosh Reddy Gurram | NA |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# **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 Page Number**

**1.Introduction**

1.1. Purpose…………………………………………………………………………….4

1.2. Scope……………………………………………………………………………....4

1.3. Definitions, Acronyms, and Abbreviations………………………………………..4

1.4. References…………………………………………………………………………4

1.5. Overview…………………………………………………………………………..4

**2.General Description**

2.1. Product Perspective………………………………………………………………..5

2.2. Product Functions………………………………………………………………….5

2.3. User Characteristics………………………………………………………………..5

2.4. General Constraints………………………………………………………………..5

2.5. Assumptions and Dependencies…………………………………………………...6

**3.Specific Requirements**

3.1. External Interface Requirements…………………………………………………...6

3.1.1.User Interfaces…………………………………………………………….6

3.1.2.Hardware Interfaces……………………………………………………….6

3.1.3.Software Interfaces………………………………………………………...7

3.1.4.Communications Interface…………………………………………………8

3.2. Functional Requirements……………………………………………………………8

3.3. Use Cases……………………………………………………………………………9

3.4. Class/Objects………………………………………………………………………..14

3.5. Non-Functional Requirements

3.5.1. Performance……………………………………………………………….15

3.5.2. Reliability………………………………………………………………….15

3.5.3.Availability………………………………………………………………....15

3.5.4. Security…………………………………………………………………….15

3.5.5. Portability…………………………………………………………………..15

3.6.Inverse Requirements…………………………………………………………………16

3.7.Design Constraints…………………………………………………………………….17

3.8.Logical Database Requirements……………………………………………………….17

3.9.Other Requirements……………………………………………………………………18

3.10. Prototypes (for complete project)……………………………………………………19

3.11. Use Case Diagrams…………………………………………………………………..21

**4. Design**

4.1 ER Diagram…………………………………………………………………………..22

4.2 GUI…………………………………………………………………………………...23

5.1 Data Flow Diagram………………………………………………………………………28

5.2 Sequence Diagram………………………………………………………………………..29

**1. Introduction:**

* The Smart Inventory project revolutionizes warehouse operations by seamlessly managing daily necessities. Through intuitive interfaces, users request and store goods while warehouse managers track and update inventory. Assigned IDs streamline tracking, ensuring transparency and accountability. Emphasizing data accuracy and security, the system safeguards sensitive information and maintains detailed transaction records. By harnessing technology, the project optimizes inventory management, enhancing efficiency, productivity, and user experience in the dynamic world of retail and logistics.

**1.1.Purpose:**

* The purpose of the Smart Inventory project is to streamline and optimize the management of goods related to daily needs within a warehouse setting. By implementing an efficient system for both users and warehouse managers, the project aims to facilitate seamless requests, storage, and tracking of inventory items. The overarching goal is to enhance convenience, accuracy, and productivity in managing inventory operations.

**1.2.Scope:**

* The scope of the Smart Inventory project encompasses the development of a user-friendly interface accessible to both users and warehouse managers, facilitating efficient interaction with the inventory system. Key functionalities include inventory management features enabling users to request goods and warehouse managers to process these requests, update inventory levels, and allocate specific IDs for tracking. Real-time tracking capabilities ensure seamless monitoring of goods movement within the warehouse, while a robust data management system maintains accurate records of transactions and inventory details. Additionally, a notification mechanism keeps users and warehouse managers informed about the status of requests, enhancing communication and responsiveness. Security measures safeguard sensitive data, and the system is designed to accommodate potential scalability, allowing for future expansion and feature integration. Through these components, the Smart Inventory project aims to streamline inventory operations, improve user experience, and enhance productivity within warehouse environments.

1.3.Definitions, Acronyms, and Abbreviations

1.4.References

**1.5.Overview:**

* The Smart Inventory project introduces an innovative solution for efficient management of goods in warehouse settings. It offers a user-friendly interface for seamless interaction between users and warehouse managers, enabling easy request submission, inventory updates, and allocation of unique IDs for tracking purposes. Real-time tracking capabilities ensure transparency and accountability throughout the inventory lifecycle. Robust data management protocols ensure the accuracy and security of sensitive information, while a notification system keeps stakeholders informed about the status of requests. With scalability in mind, the project is poised to adapt to evolving needs and technological advancements, promising to revolutionize inventory management processes and enhance overall operational efficiency.

**2. General Description:**

* For small and medium-sized enterprises, Smart Inventory is a complete inventory management solution that makes inventory tracking, ordering, and management easier. Businesses may optimize inventory levels, lower stockouts, and boost overall productivity thanks to the system's real-time stock level insight.

**2.1. Product Perspective**

* Although Smart Inventory is intended to be used independently, it may be interfaced with current ERP (Enterprise Resource Planning) and POS (Point of Sale) systems. It acts as a consolidated platform for ordering, reporting, and stock tracking, among other inventory management functions.

**2.2. Product Functions**

* Inventory Management: With Smart Inventory, users may monitor current stock levels in real time, including the amount of goods that are available for purchase, on hand, and in order.
* Stock warnings: To help users prevent stockouts, the system can send warnings when stock levels are low.
* Reporting and Analytics: To assist organizations in making wise decisions, Smart Inventory offers a range of statistics and analytics on things like sales patterns, stock aging, and inventory turnover.
* Barcode Scanning: Barcode scanning allows users to rapidly add or remove inventory items.
* User Management: To protect data, the system allows for several user accounts with different levels of access.

**2.3. User Characteristics**

* The retail, manufacturing, and distribution industries' businesses are the main customers of Smart Inventory. Those that utilize the system to manage their inventory operations are usually business owners, warehouse supervisors, and inventory managers who need a dependable and effective solution.

**2.4.General Characteristics**

* Internet Connection: To synchronize and update data in real-time, Smart Inventory needs an internet connection.
* Gear Compatibility: For some operations, the system may need suitable gear, such as mobile devices or barcode scanners.
* User Training: To fully utilize all of the system's functions, users may need to undergo training.

**2.5 Assumptions and Dependencies**

* Data Accuracy: Smart Inventory relies on the accuracy and timeliness of the data entered into the system.
* Integration: For some features, like barcode scanning or accounting software, the system could rely on third-party integrations.
* User Adoption: Using the functionality of the system and encouraging user adoption are key factors in the success of Smart Inventory.

**3.Specific Requirements**

**3.1. External Interface Requirements**

**3.1.1.User Interfaces**

Designing a user interface (UI) for a smart Inventory application involves considering the various functionalities and information that need to be presented in a clear and user-friendly manner. Here's a basic outline of what elements could be included:

1. Navigation: A sidebar or tab-based navigation to easily switch between different sections of the application such as inventory management, order processing, etc.
2. Inventory Management:
   * List view of current inventory with details like product name, quantity, location in the warehouse, etc.
   * Search and filter options to quickly find specific items.
   * Ability to add new items to the inventory.
3. Request:
   * List of products with details like order number.
4. Shipment Tracking:
   * Tracking of outgoing shipments with details like shipment number.
5. Reports:
   * Generate and view reports on inventory levels, order history, shipment status, etc.
6. Settings:
   * User profile settings, application settings, etc.
7. Notifications:
   * Alerts and notifications for low inventory, new orders, etc.
8. Responsive Design:
   * Ensure the UI is responsive and works well on different devices and screen sizes.
9. Accessibility:
   * Consider accessibility features for users with disabilities, such as screen reader compatibility and high contrast modes.
10. Security:
    * Implement secure login/authentication mechanisms to protect sensitive information.
11. Feedback:
    * Include a way for users to provide feedback or report issues with the application.

**3.1.2.Hardware Interfaces**

Our Hardware device is ioS device like a Mac or iPhone. We won't need additional hardware interfaces. Instead, you'll focus on creating a user interface (UI) that works seamlessly on iOS devices such as iPhones and iPads. Here are some considerations for designing the UI:

1. **Screen Size**: Design your UI to be responsive and adaptable to different screen sizes, especially considering the varying sizes of iPhones and iPads.
2. **Touch Interaction**: Utilize iOS's touch interface for intuitive interactions such as tapping, swiping, and dragging to navigate and interact with inventory items.
3. **Optimized Layout**: Design a layout that makes efficient use of the screen space, prioritizing important information and actions while keeping the interface uncluttered.
4. **Native Controls**: Use native iOS controls and elements for a familiar user experience, including navigation bars, tab bars, and buttons.
5. **Accessibility**: Ensure your UI is accessible to users with disabilities by supporting VoiceOver, Dynamic Type, and other accessibility features built into iOS.
6. **Performance**: Optimize the performance of your UI to ensure smooth animations and responsiveness, especially when dealing with large amounts of data.
7. **Data Visualization**: Use charts, graphs, and visual elements to present inventory data in a clear and understandable manner.
8. **Feedback and Confirmation**: Provide visual feedback and confirmation messages for user actions to enhance the user experience.

**3.1.3.Software Interfaces**

Software interfaces for your iOS smart inventory management system will involve the interaction between your app and external services, databases, or APIs. Here are some key software interfaces you might consider:

1. **Database Interface**: Use Core Data or SQLite to interact with a local database on the iOS device for storing and managing inventory data. Alternatively, you might use a cloud-based database like Firebase or AWS DynamoDB for remote storage.
2. **User Interface (UI) Interface**: Design your UI using UIKit framework, adhering to Apple's Human Interface Guidelines (HIG) for iOS app design to ensure a consistent and intuitive user experience.
3. **Location Services Interface**: Use Core Location framework to access the device's location information, which can be useful for tracking inventory movement within a warehouse or store.
4. **Security Interface**: Implement security measures such as data encryption, secure authentication, and secure data storage to protect sensitive inventory information.

**3.1.4.Communications Interface**

1. **RESTful AP**I: A Representational State Transfer (REST) API provides a lightweight communication interface that uses standard HTTP methods (GET, POST, PUT, DELETE) for data exchange. It allows clients to access and manipulate inventory data through well-defined endpoints, making it suitable for web-based and mobile applications.
2. **Message Queues**: Message queuing systems such as RabbitMQ or Apache Kafka facilitate asynchronous communication between components of the inventory analysis system. They enable decoupling of producers and consumers, ensuring reliable delivery of messages and scalable processing of inventory data.
3. **File Transfer**: File transfer protocols like FTP (File Transfer Protocol) or SFTP (SSH File Transfer Protocol) allow the transfer of inventory data files between systems. This approach is useful for batch processing and bulk data transfers, especially when dealing with large datasets or legacy systems that don't support real-time communication.
4. **Webhooks**: Webhooks provide a mechanism for triggering HTTP callbacks or notifications in response to events within the inventory management system. They enable real-time communication between systems by delivering payload data to designated endpoints, allowing for immediate processing and analysis of inventory events.
5. **Database Replication**: Database replication involves synchronizing data between databases in near real-time, ensuring consistency across distributed systems. By replicating inventory data between databases, analysis systems can access up-to-date information for reporting and decision-making purposes.

**3.2. Functional Requirements**

1. **User Authentication and Authorization:**
   * The application should provide secure user authentication mechanisms, such as username/password login, biometric authentication (e.g., Touch ID, Face ID), or OAuth integration with third-party authentication providers.
   * Authorized users should have access to specific inventory management functionalities based on their roles and permissions, such as viewing inventory, adding new items, or generating reports.
2. **Inventory Management:**
   * Users should be able to view a comprehensive list of available inventory items, including details such as product name, description, quantity in stock, and location.
   * The application should support adding new inventory items manually or through barcode scanning, with options to input additional details like category, supplier, and purchase price.
   * Users should be able to update inventory quantities, mark items as received or sold, and adjust stock levels based on physical counts or discrepancies.
3. **Search and Filtering:**
   * The application should provide robust search and filtering capabilities to help users quickly find specific inventory items based on various criteria such as name, category, supplier, or location.
   * Users should be able to apply multiple filters simultaneously to narrow down search results and locate items efficiently.
4. **Inventory Analysis and Reporting:**
   * The application should offer built-in analytics tools to analyze inventory data, such as sales trends, stock turnover rates, and inventory valuation.
   * Users should be able to generate customizable reports and visualizations to gain insights into inventory performance, identify potential stock shortages or surpluses, and make data-driven decisions.
   * Reports should be exportable in common formats (e.g., PDF, CSV) for sharing with stakeholders or integration with external systems.
5. **Alerts and Notifications:**
   * The application should support configurable alerts and notifications to alert users of critical inventory events, such as low stock levels, expiring products, or incoming shipments.
   * Notifications should be delivered in real-time via push notifications or within the application's interface, with options to acknowledge or dismiss alerts.
6. **Integration with External Systems:**
   * The application should integrate seamlessly with external systems and services, such as ERP (Enterprise Resource Planning) systems, e-commerce platforms, or accounting software, to synchronize inventory data and streamline business operations.
   * APIs or integration connectors should be available to facilitate data exchange and interoperability with third-party systems.

**3.3 Use Cases:**

**User Requesting Goods:**

Description: A user logs into the system and navigates to the inventory interface.

Steps:

The user selects the desired items from the inventory catalog.

Enter the quantity required and any additional specifications.

Submit the request.

Outcome: The system records the request, updates the inventory levels, and generates a unique request ID. A notification is sent to the warehouse manager for processing.

**Warehouse Manager Processing Request:**

Description: The warehouse manager receives a notification about a new user request.

Steps:

Accesses the pending requests dashboard.

Reviews the details of the request, including item, quantity, and user information.

Confirms availability of the requested items in the inventory.

Allocates the items for the user and generates a unique ID for tracking.

Updates the inventory levels.

Outcome: The user receives a notification confirming the approval of their request along with the assigned ID. The inventory is adjusted to reflect the allocated items.

**Inventory Tracking and Reporting:**

Description: The warehouse manager needs to track inventory movement and generate reports.

Steps:

Accesses the inventory tracking dashboard.

Views real-time inventory levels, including items in stock, allocated items, and stored items.

Generates reports on inventory turnover, stock levels, and user activity.

Outcome: The warehouse manager gains insights into inventory trends, identifies areas for improvement, and ensures efficient resource allocation.

**3.4 Use Cases:**

**User Class:**

Description: Represents a user of the inventory system.

Attributes:

UserID: Unique identifier for each user.

Name: User's name.

Role: Role of the user (e.g., regular user, administrator).

Methods:

requestGoods(): Method to request goods from the warehouse.

storeGoods(): Method to store goods in the warehouse.

**InventoryItem Class:**

Description: Represents an item in the warehouse inventory.

Attributes:

ItemID: Unique identifier for each item.

Description: Description of the item.

Quantity: Quantity of the item in stock.

ReorderPoint: Minimum quantity at which the item should be reordered.

Methods:

updateQuantity(): Method to update the quantity of the item in stock.

isLowStock(): Method to check if the item is below the reorder point.

**Request Class:**

Description: Represents a request made by a user for goods from the warehouse.

Attributes:

RequestID: Unique identifier for each request.

User: User making the request.

ItemsRequested: List of items requested by the user.

Status: Status of the request (e.g., pending, approved, declined).

Methods:

processRequest(): Method to process the request by the warehouse manager.

generateID(): Method to generate a unique ID for the request.

**WarehouseManager Class:**

Description: Represents a warehouse manager responsible for managing inventory operations.

Attributes:

ManagerID: Unique identifier for the warehouse manager.

Name: Manager's name.

Permissions: Level of permissions granted to the manager (e.g., full access, read-only).

Methods:

processRequest(): Method to process user requests and update inventory accordingly.

handleStorage(): Method to handle storage requests from users.

generateReport(): Method to generate inventory reports.

**Notification Class:**

Description: Represents notifications sent to users and warehouse managers.

Attributes:

NotificationID: Unique identifier for each notification.

Recipient: User or manager receiving the notification.

Content: Content of the notification.

Timestamp: Timestamp indicating when the notification was sent.

Methods:

sendNotification(): Method to send notifications to recipients.

**Transaction Class:**

Description: Represents transactions involving the movement of goods within the warehouse.

Attributes:

TransactionID: Unique identifier for each transaction.

Type: Type of transaction (e.g., receiving, issuing, returning).

ItemsInvolved: List of items involved in the transaction.

Quantity: Quantity of each item involved in the transaction.

Timestamp: Timestamp indicating when the transaction occurred.

Methods:

logTransaction(): Method to log details of the transaction.

**Supplier Class:**

Description: Represents suppliers providing goods to the warehouse.

Attributes:

SupplierID: Unique identifier for each supplier.

Name: Name of the supplier.

ContactInfo: Contact information of the supplier (e.g., address, phone number).

Methods:

updateCatalog(): Method to update the inventory catalog with items supplied by the supplier.

placeOrder(): Method to place orders for goods with the supplier.

**StorageSpace Class:**

Description: Represents storage spaces within the warehouse for storing goods.

Attributes:

SpaceID: Unique identifier for each storage space.

Capacity: Maximum capacity of the storage space.

ItemsStored: List of items currently stored in the space.

Methods:

checkAvailability(): Method to check the availability of space for storing goods.

allocateSpace(): Method to allocate space for storing goods.

**Location Class:**

Description: Represents physical locations within the warehouse.

Attributes:

LocationID: Unique identifier for each location.

Aisle: Aisle number where the location is situated.

Shelf: Shelf number within the aisle.

Bin: Bin number on the shelf.

Methods:

getLocationDetails(): Method to retrieve details of the location.

**Report Class:**

Description: Represents reports generated for analyzing inventory data.

Attributes:

ReportID: Unique identifier for each report.

Type: Type of report (e.g., inventory turnover, stock level).

Data: Data included in the report (e.g., tables, charts).

GeneratedBy: User or manager who generated the report.

Methods:

generateReport(): Method to generate the report based on specified parameters.

**3.5** **Non-Functional Requirements:**

**3.5.1. Performance:**

* The application should load inventory data quickly, with minimal latency, even when dealing with large datasets.
* Response times for user interactions such as searching, filtering, and updating inventory information should be within acceptable limits (e.g., under 1 second).
* The application should efficiently utilize device resources (CPU, memory, battery) to ensure smooth performance without draining the battery excessively.

**3.5.2. Reliability:**

* The application should be stable and robust, minimizing crashes or unexpected behavior.
* It should handle network disruptions gracefully and provide appropriate error handling mechanisms to guide users in case of failures.
* Data integrity should be maintained at all times, with mechanisms in place to prevent data loss or corruption.

**3.5.3. Availability:**

* It should support offline mode, allowing users to access cached inventory data and perform essential tasks even without an internet connection.
* In cases of server-side dependencies, the application should gracefully handle temporary unavailability and provide feedback to users.
* The application should be available for use whenever required, with minimal downtime for maintenance or updates.

**3.5.4. Security:**

* User authentication and authorization mechanisms should be implemented to ensure that only authorized personnel can access sensitive inventory data.
* Data transmission between the application and any backend servers should be encrypted to prevent eavesdropping or tampering.
* The application should adhere to iOS security best practices and guidelines to mitigate risks such as data breaches or unauthorized access.

**3.5.5. Portability:**

* It should adapt to various screen sizes and resolutions, providing a consistent user experience regardless of the device used.
* If applicable, the application should support features specific to the iOS ecosystem, such as integration with iCloud for data synchronization across devices.
* The application should be designed and developed using iOS platform standards to ensure compatibility across different iOS devices (iPhone, iPad)

**3.6 Inverse Requirements**

**Inverse Requirement 1:** The system should not allow users to request goods without providing necessary authentication or authorization.

**Inverse Requirement 2:** The system should not automatically approve user requests without validation of inventory availability or warehouse manager confirmation.

**Inverse Requirement 3:** The system should not allow users to store goods in the warehouse without adhering to specified storage guidelines or restrictions.

**Inverse Requirement 4:** The system should not allow warehouse managers to process storage requests without proper inspection or verification of stored items.

**Inverse Requirement 5:** The system should not disclose sensitive inventory information to unauthorized users or entities.

**Inverse Requirement 6:** The system should not allow modification of inventory data without proper logging and tracking of changes.

**Inverse Requirement 7:** The system should not automatically generate reports or notifications without user request or configuration.

**Inverse Requirement 8:** The system should not permit unauthorized users to access or modify system settings, configurations, or permissions.

**Inverse Requirement 9:** The system should not store or display excessive personal information about users beyond what is necessary for inventory management purposes.

**Inverse Requirement 10:** The system should not perform actions that could potentially compromise the security or integrity of inventory data or transactions.

**3.7 Design Constraints**

**Scalability:** The system should be able to manage potentially huge and growing amounts of data, such as product information, transaction records, and user requests. It should be able to scale seamlessly to meet increased usage while maintaining performance and data integrity.

**Reliability and Availability**: The system should be extremely reliable, allowing it to be used whenever necessary. This includes developing methods to prevent and recover from failures, such as backup and redundancy plans, in order to reduce downtime and ensure data integrity.

**Security**: The system should prioritize data security to protect sensitive information including inventory levels, transaction details, and user data from unauthorized access, tampering, and data breaches. This includes putting in place authentication, access control, and encryption techniques to protect the system and its data.

**Interoperability**: The system should be built to work easily with existing warehouse management systems, as well as other essential software and hardware components used in the retail and logistics industries. This guarantees that multiple systems communicate and exchange data seamlessly, allowing for more efficient operations and minimizing friction.

**Usability**: The system should be user-friendly and easy to understand, with interfaces that streamline user interactions and minimize the learning curve for users. This includes clearly stating instructions, assistance, as well as ensuring accessibility for users with diverse requirements.

**3.8. Logical Database Requirements**

**1. Data Model**: Define a data model that represents the entities and relationships relevant to inventory management, such as products, warehouses, transactions, and users. Use a relational database model to organize the data into tables.

**2. Database Schema**: Design the database schema based on the data model, specifying the structure of each table, including the attributes (columns) and their data types, constraints, and relationships with other tables.

**3. Normalization**: Apply normalization techniques to ensure the database is free from redundancy and update anomalies. Normalize the database to at least third normal form (3NF) to optimize storage and maintain data integrity.

**4.** **Indexing**: Identify the fields that will be frequently used for querying and create indexes on those fields to improve query performance.

**5.** **Data Integrity**: Enforce data integrity constraints, such as primary keys, foreign keys, unique constraints, and check constraints, to maintain data accuracy and consistency.

**6.** **Security**: Implement security measures to protect the database from unauthorized access and ensure data privacy. This includes setting up user authentication and access control mechanisms.

**7.** **Backup and Recovery**: Define a backup and recovery strategy to protect against data loss. Regularly backup the database and create a plan for restoring data in case of a failure.

**8.** **Performance Tuning**: Monitor and optimize database performance by analyzing query execution plans, identifying bottlenecks, and implementing performance tuning techniques, such as index optimization and query optimization.

**9.** **Scalability**: Design the database to be scalable, allowing for future growth in data volume and user traffic. Consider using techniques such as sharding or replication to distribute the workload and improve scalability.

**3.9. Other Requirements**

1. **Real-time Inventory Updates:** Ensure that the system supports real-time updates to inventory levels, allowing for accurate and up-to-date tracking of goods**.**
2. **User Permissions:** Implement role-based access control to restrict user access to certain features or data based on their role within the organization, ensuring data security and privacy.
3. **Multi-location Support:** Support inventory management across multiple warehouse locations or stores, allowing for centralized control and visibility of inventory levels.
4. **Customizable Reporting**: Provide customizable reporting capabilities, allowing warehouse managers to generate reports tailored to their specific needs and requirements

**3.10. Prototypes**

For the Smart Inventory project, we can create prototypes for both the user and the inventory manager interfaces. Here are the basic screens for each:

### **User Interface Prototypes:**

1. User Registration/Login:

* Fields for registration (name, email, password).
* Login form.

1. User Dashboard:

* View of inventory stores.
* Option to add new inventory stores.
* Option to view/edit existing inventory stores.

1. Add Product to Inventory:

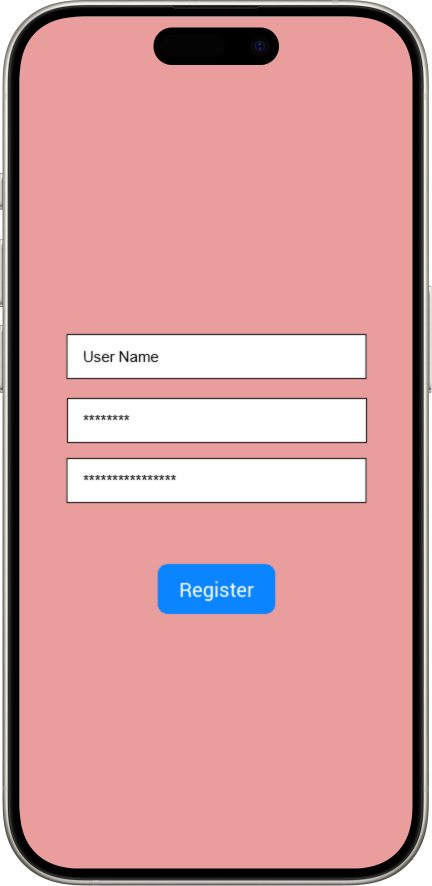
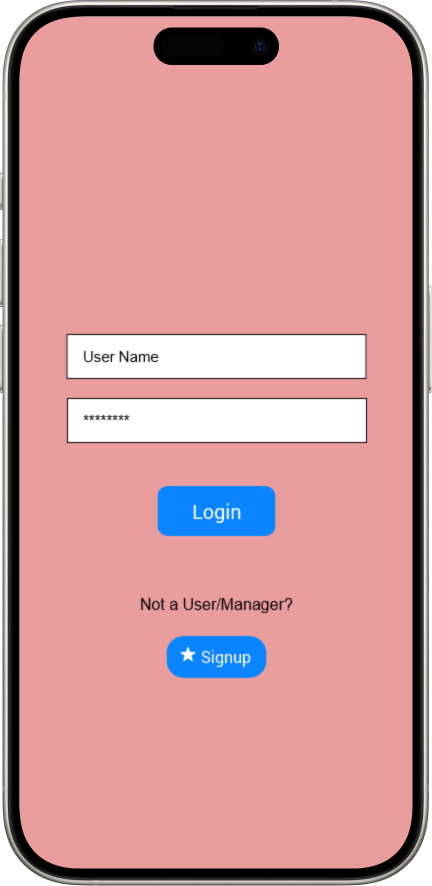
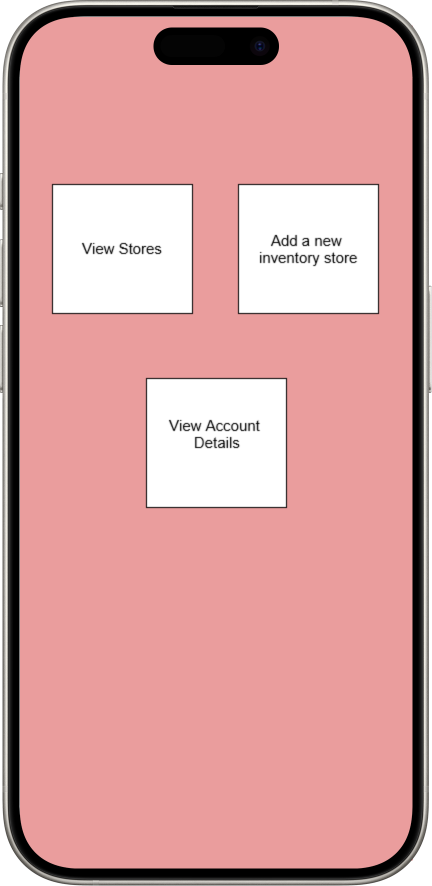
* Form to add a new product to an inventory store (name, quantity, description,etc.).
* Selection of the inventory store to add the product to.

1. View Inventory Store:

* List of products in an inventory store.
* Option to edit/delete products.

1. Shipping Products:

* Form to select products from the inventory store for shipping.
* Option to select destination and shipping details.

Register Page Login Page User Page 1



User Page2

### **Inventory Manager Interface Prototypes:**

1. Inventory Manager Login:

Login form for inventory managers.

1. Inventory Manager Dashboard:

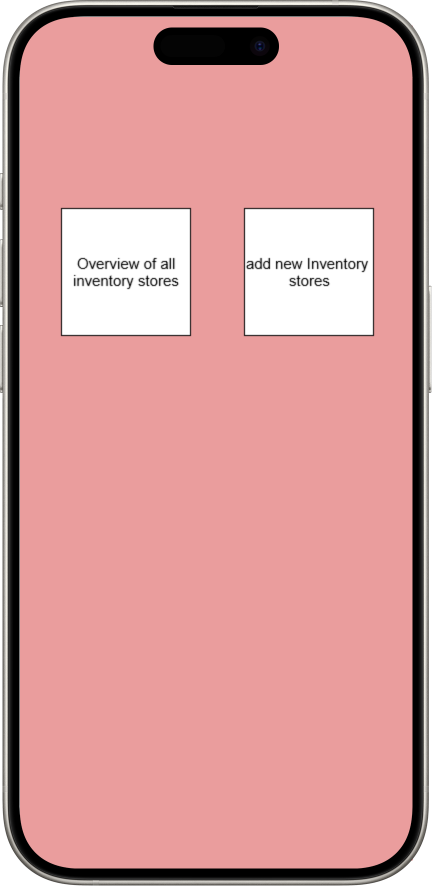
* Overview of all inventory stores.
* Option to add new inventory stores.
* Option to view/edit existing inventory stores.

1. Manage Inventory Store:

* List of products in an inventory store.
* Option to edit/delete products.
* Option to add new products.

1. Shipping Management:

* List of pending shipments.
* Option to update shipment status (shipped, delivered, etc.).
* Option to view shipment details.



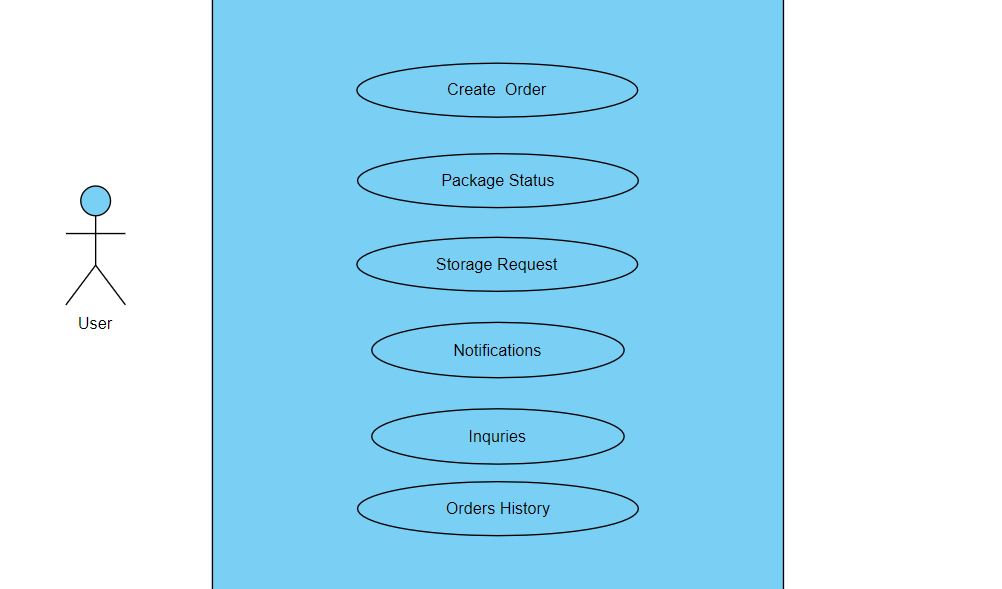
Inventory Manager page1 Inventory Manager page2

### **Additional Features:**

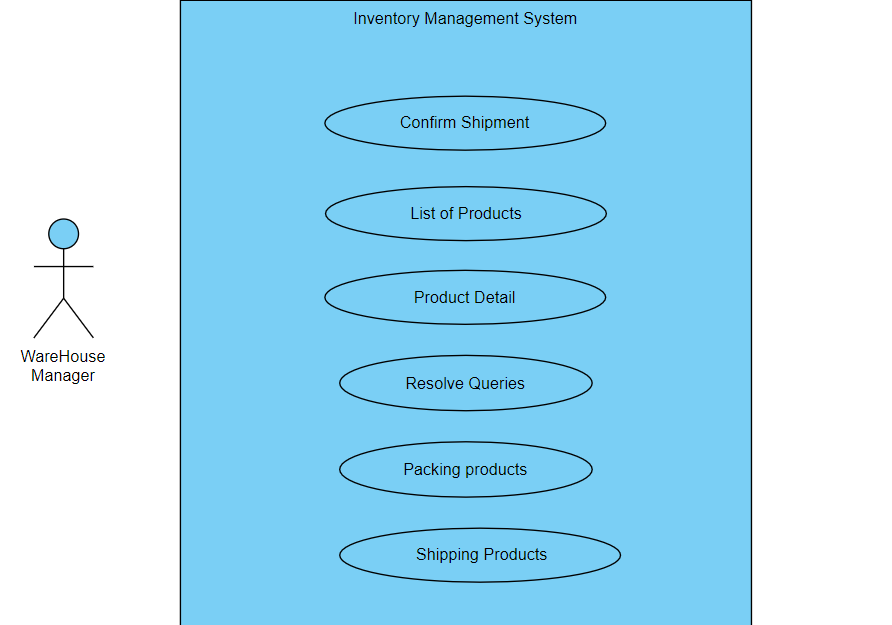
* Notifications: Notify users and the inventory manager about low stock, new product additions, etc.
* Reports: Generate reports for the inventory manager regarding inventory usage, popular products, etc.
* Barcode Scanner: Allow users to scan product barcodes for easy addition to the inventory.
* These are just basic prototypes and can be expanded upon based on the specific requirements and functionalities of the Smart Inventory project.

**3.11.Use Case Diagrams**

User Use Case Diagram

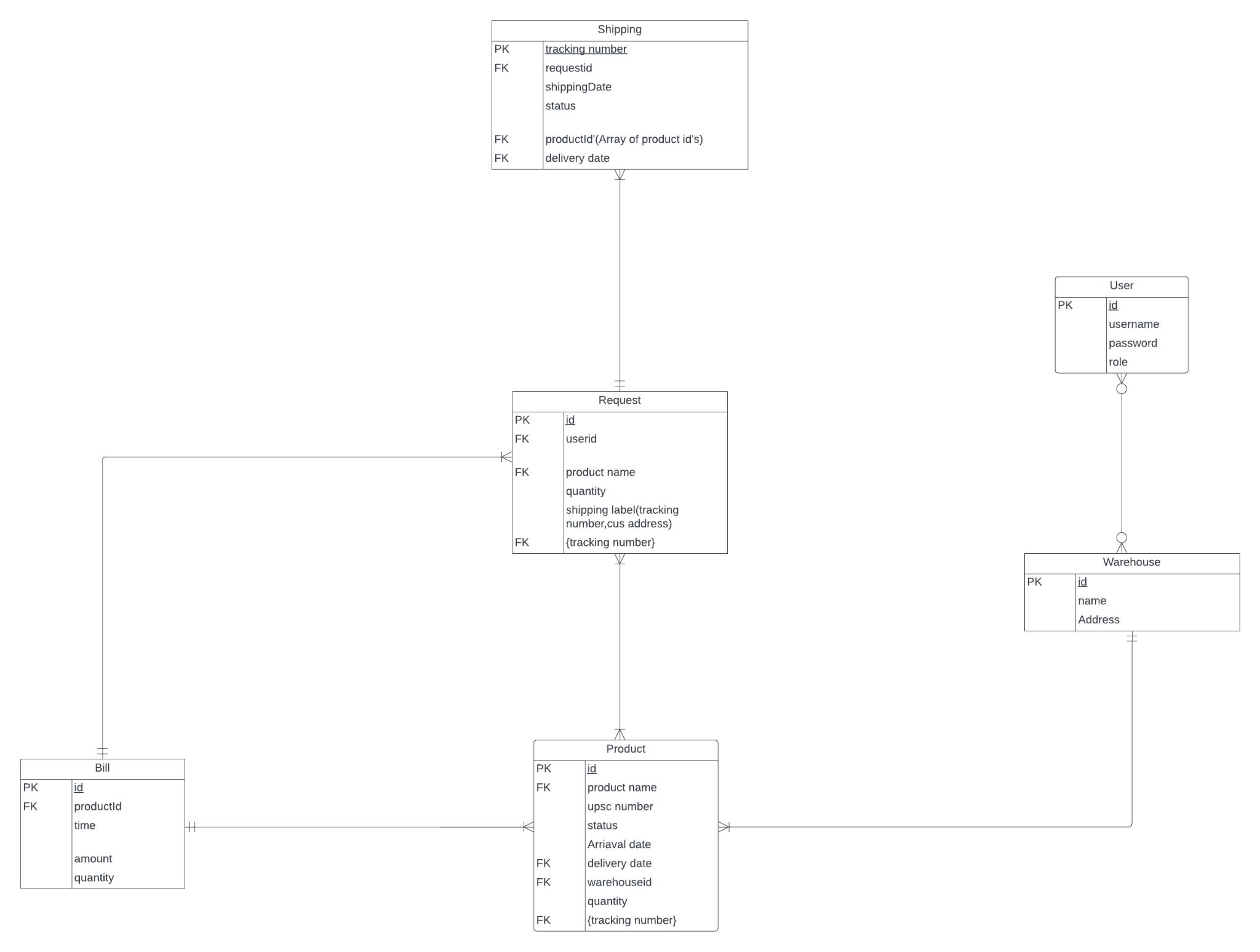


Inventory Manager Use Case Diagram

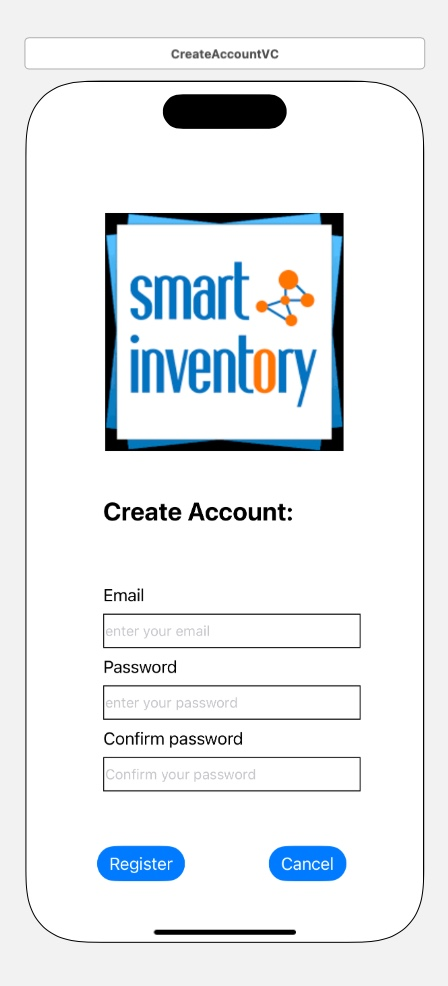
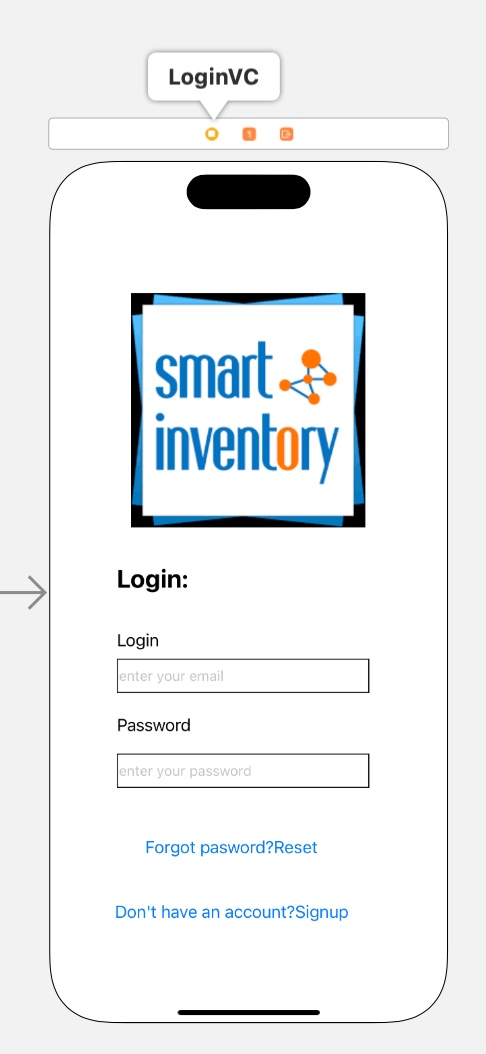


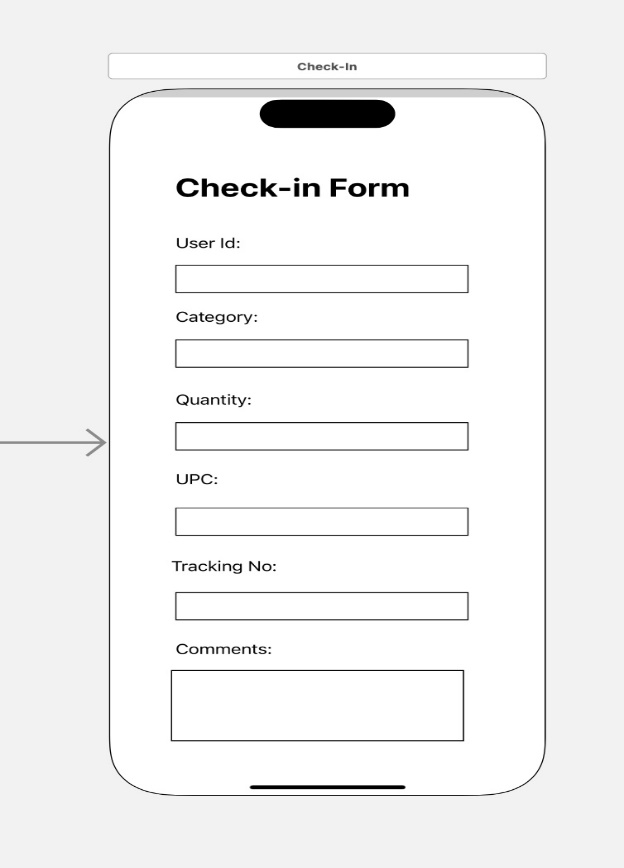
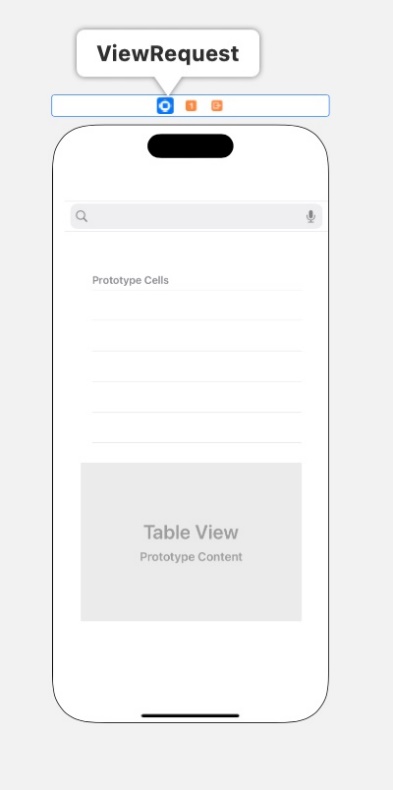
**4. Design**

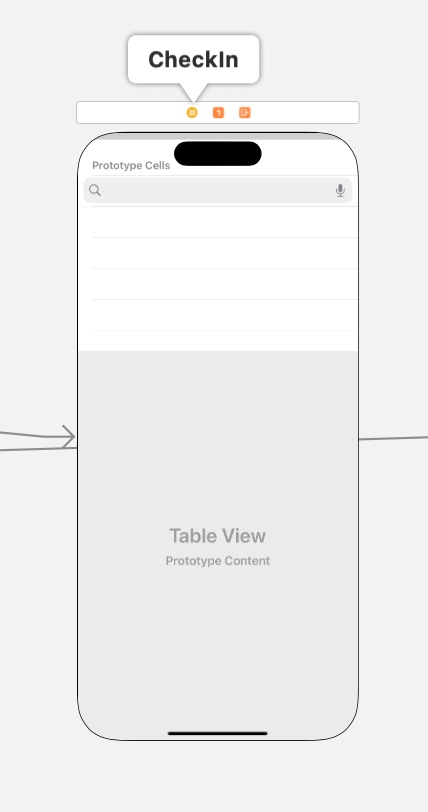
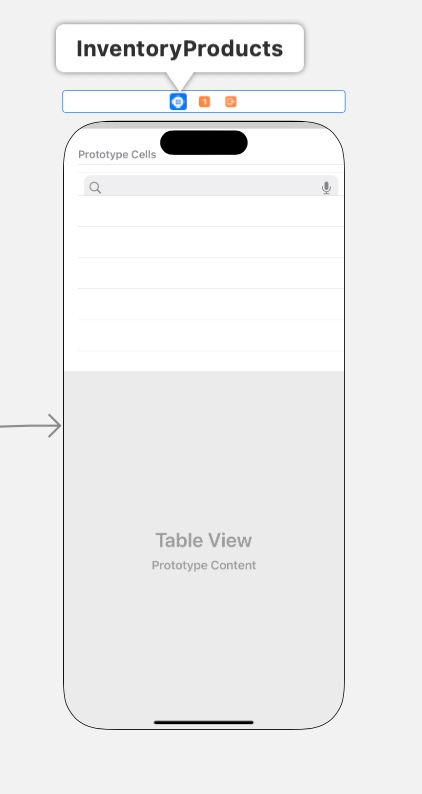
**4.1 ER Diagram**

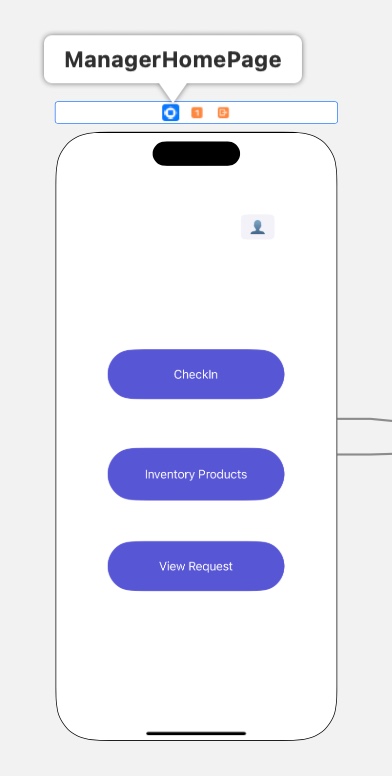
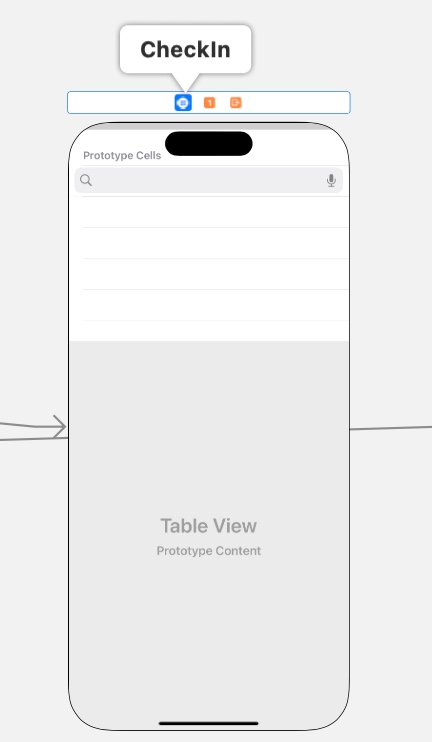


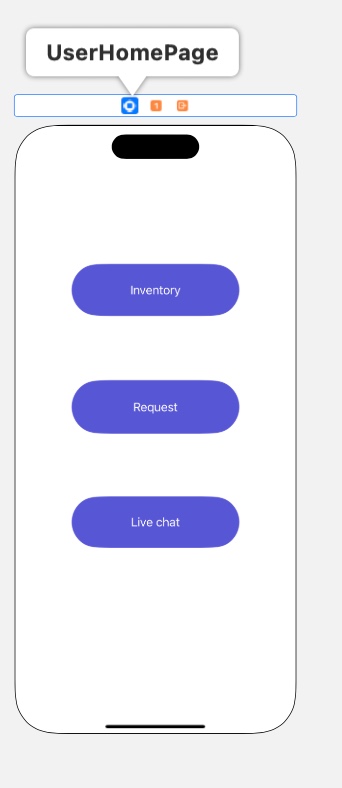
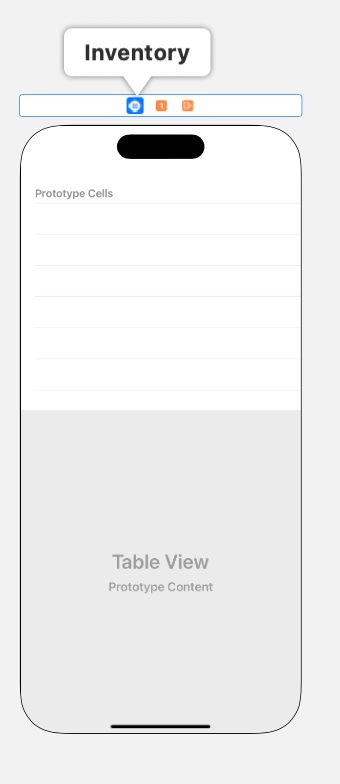
**4.2 GUI**

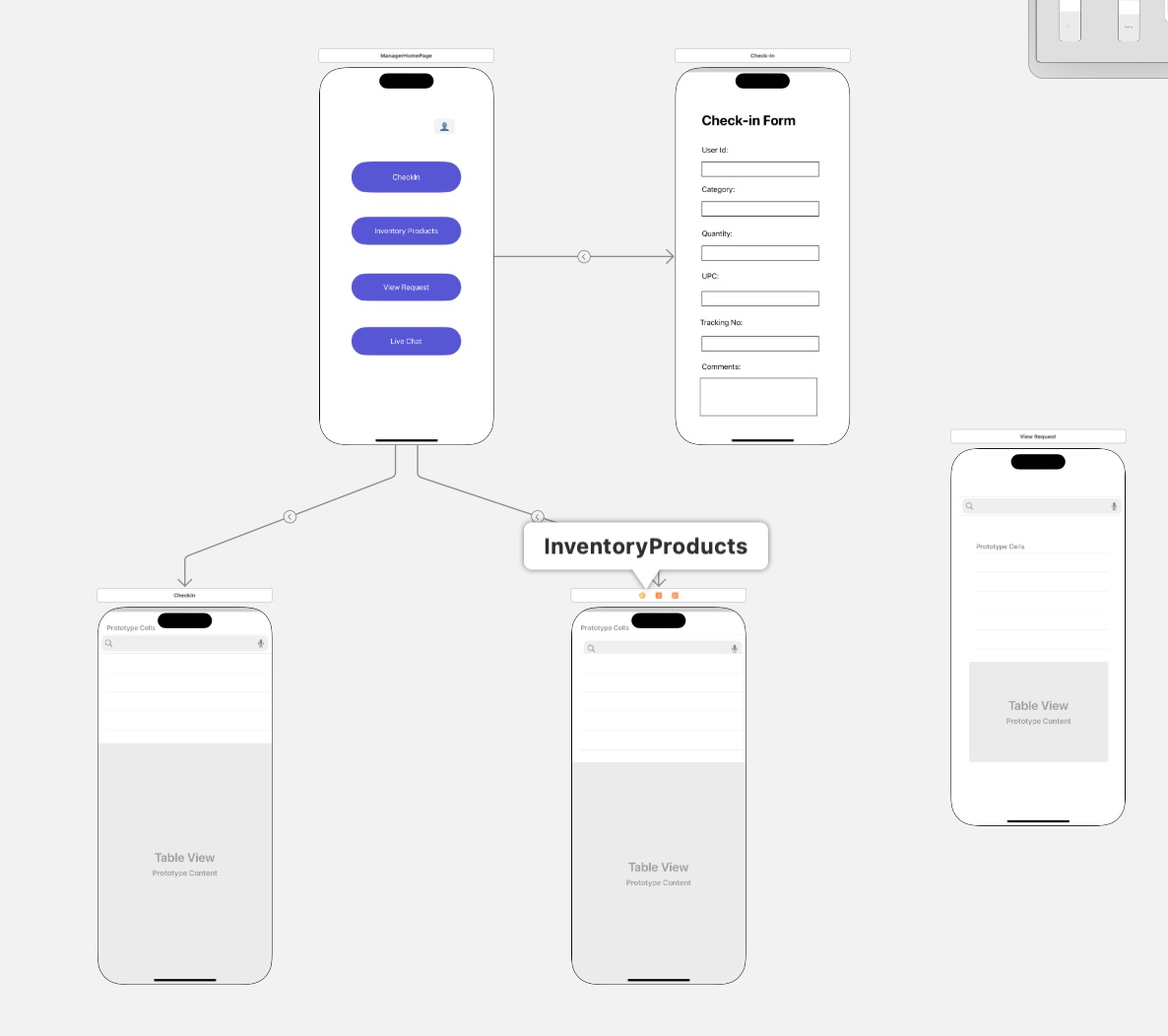


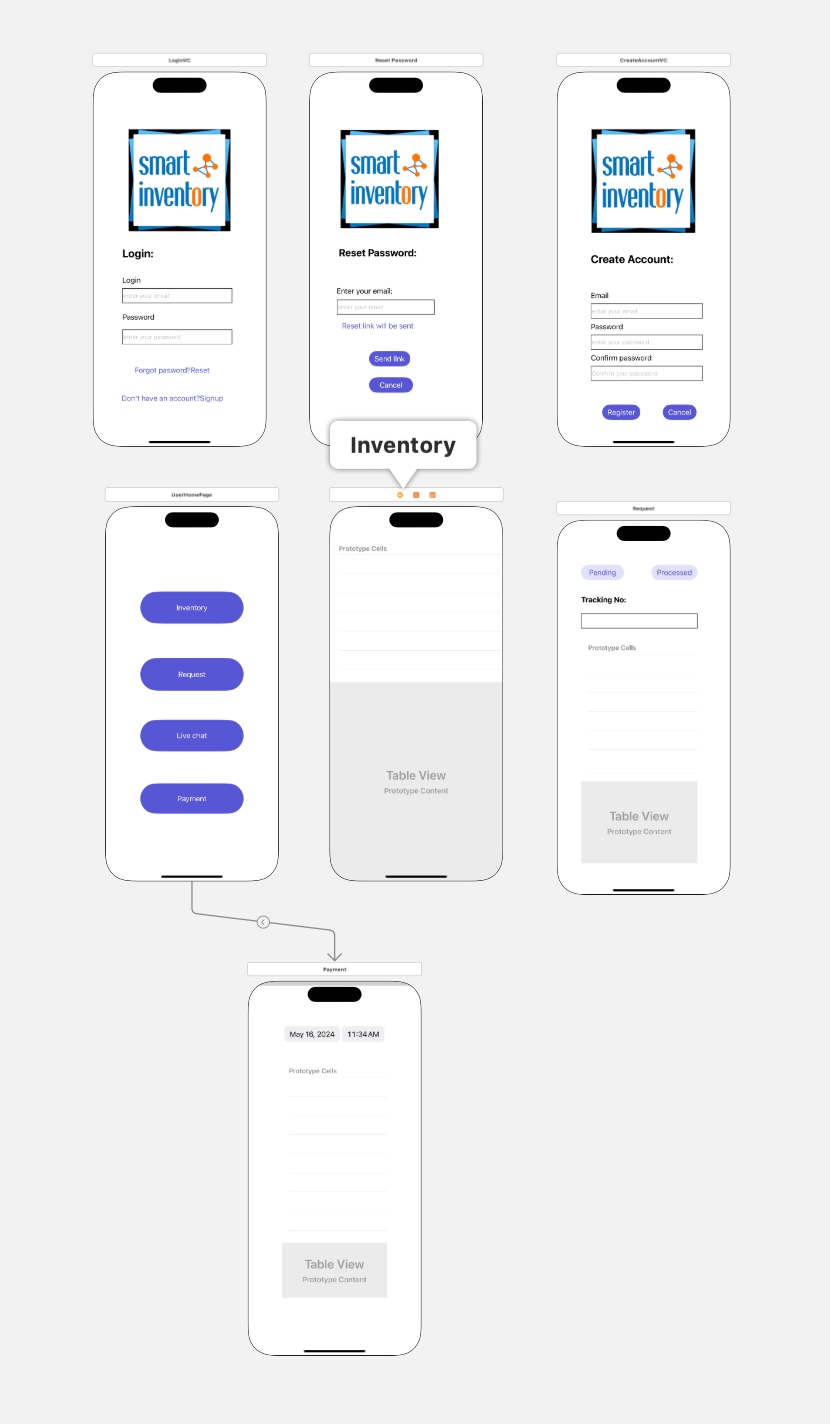


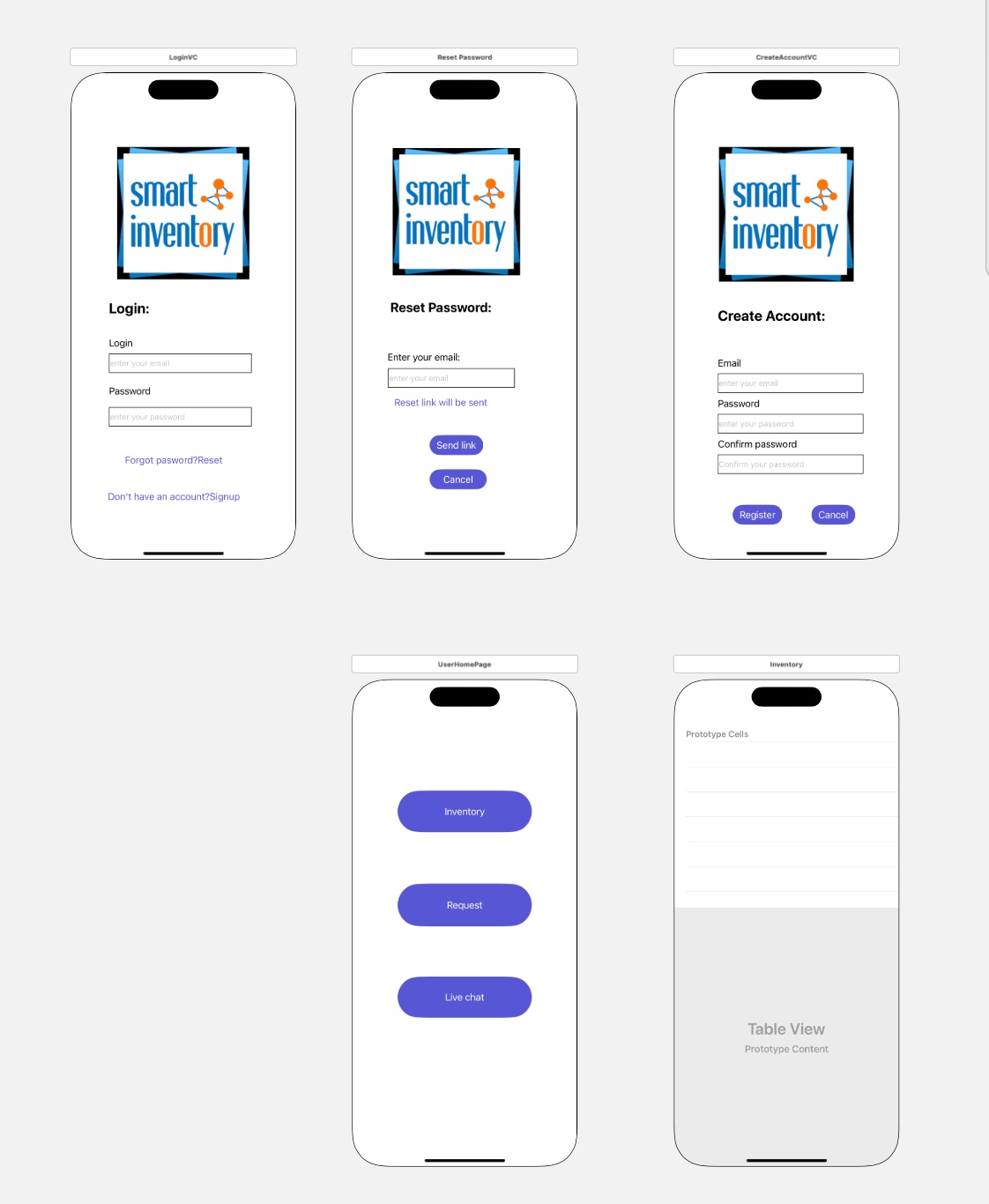




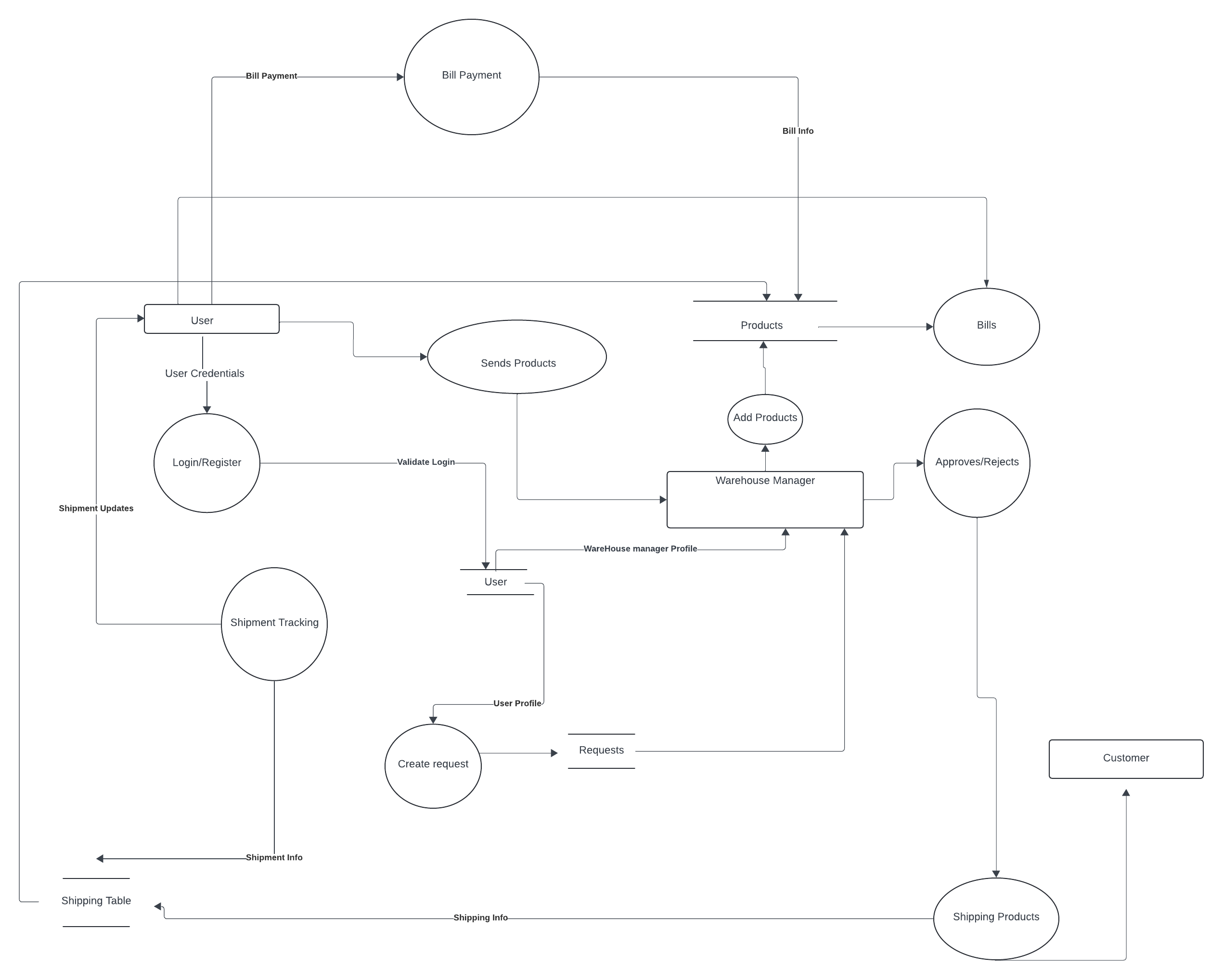








**5.1 Data flow diagram**



**5.2 Sequence Diagram**

A diagram of a diagram

Description automatically generated with medium confidence