Ministry of High Education
Culture and Science City at 6 of Oct,
The High Institute of Computer Science & Information Systems

المعهد العالى لعلوم الحاسب و نظم المعلومات

Graduation Project

Smart Inventory Management System

Assistant :

Eng. Momen Elngar

Supervised by:

Dr. Shaimaa Mahmoud

Project No : 603 **Academic Year:** 2024/2025

Ministry of High Education
Culture and Science City at 6 of Oct,
The High Institute of Computer Science & Information Systems

المعهد العالى لعلوم الحاسب و نظم المعلومات

Graduation Project

Smart Inventory Management System

Prepared by :

211009	زياد محمد احمد على	211969	عمر محمد احمد سيد
210927	عبدالرحمن محمد احمد محمد	211650	مارتينا صفوت ليتو رياض
211720	علاء عماد محمد على	210229	محمد محمد عبدالمؤمن محمد

Assistant :

Eng. Momen Elngar

Supervised by:

Dr. Shaimaa Mahmoud

Project No : 603 **Academic Year:** 2024/2025

Acknowledgment

First and foremost, thanks **Allah** who gave us strength, patience, and ability to accomplish this project .

We would like to extent our gratitude to **(Culture and Science City)** and the dean of Higher Institute of Computer Science and Information Systems for their cooperation and encouragement, which played a significant role in completing this project. Your support and partnership have been invaluable, and we are thankful for the opportunities you provided us.

From the bottom of our heart, we express deep gratitude to **Dr. Shaimaa Mahmoud** and **Eng. Momen Elngar** for their invaluable support and guidance throughout our graduation project. Their unwavering dedication, expertise, and encouragement have been instrumental in our success. We are privileged to have learned from individuals as accomplished and passionate as them. Their contributions as supervisors and mentors have played a crucial role in our achievements, and we are forever grateful for their unwavering support.

Project Abstract

In large-scale warehouses, manual inventory counting remains a prevalent practice, leading to significant inefficiencies such as human errors, time delays, and inaccuracies in tracking available or missing shipments. These challenges are particularly pronounced in high-volume, 24/7 operations like those of global logistics giants such as Amazon and Aramex. The reliance on manual processes not only hampers operational efficiency but also results in delays in order fulfillment, lost shipments, and increased costs. To address these issues, this project proposes the development of a '**Smart Inventory Management System**' that leverages advanced technologies to automate and streamline inventory tracking and management.

The proposed system aims to eliminate the inefficiencies of manual stocktaking by integrating modern tools such as barcode scanners, **RFID** (Radio-Frequency Identification), **IR** (Infrared) or ultrasonic sensors, and automated gates. These technologies will enable real-time tracking of shipments as they enter or exit the warehouse, as well as monitor the availability of storage spaces on shelves. The system will operate seamlessly at warehouse entry and exit points, as well as within the storage areas, ensuring continuous, 24/7 monitoring of inventory movements. By automating the tracking process, the system will provide warehouse managers and staff with accurate, up-to-date inventory data, reducing the need for human intervention and minimizing errors.

Key features of the system include automated gates equipped with barcode scanners and RFID readers to capture shipment data upon entry or exit, and sensors installed on shelves to detect whether they are full or empty. This data will be transmitted in real-time to a central system, which will update inventory levels and display the information on a dedicated web interface accessible to warehouse staff. The system will also use communication

protocols to connect embedded systems controlling the sensors and gates with the central database, ensuring seamless data flow and real-time updates.

The primary beneficiaries of this system are large warehouse owners and managers who face challenges in tracking the movement of shipments within and outside their facilities. By automating inventory management, the system will significantly improve operational efficiency, reduce costs, and enhance the accuracy of inventory data. This, in turn, will lead to faster order fulfillment, reduced shipment losses, and improved customer satisfaction. The system is designed to be scalable and adaptable, making it suitable for any large warehouse handling high volumes of shipments, regardless of the type of products or the size of the facility.

The Smart Inventory Management System represents a comprehensive solution to the challenges of manual inventory counting. By leveraging automation and real-time data tracking, the system will transform warehouse operations, ensuring accuracy, efficiency, and reliability in inventory management. This innovation will not only benefit warehouse operators but also contribute to the overall growth and competitiveness of businesses relying on large-scale logistics and supply chain operations.

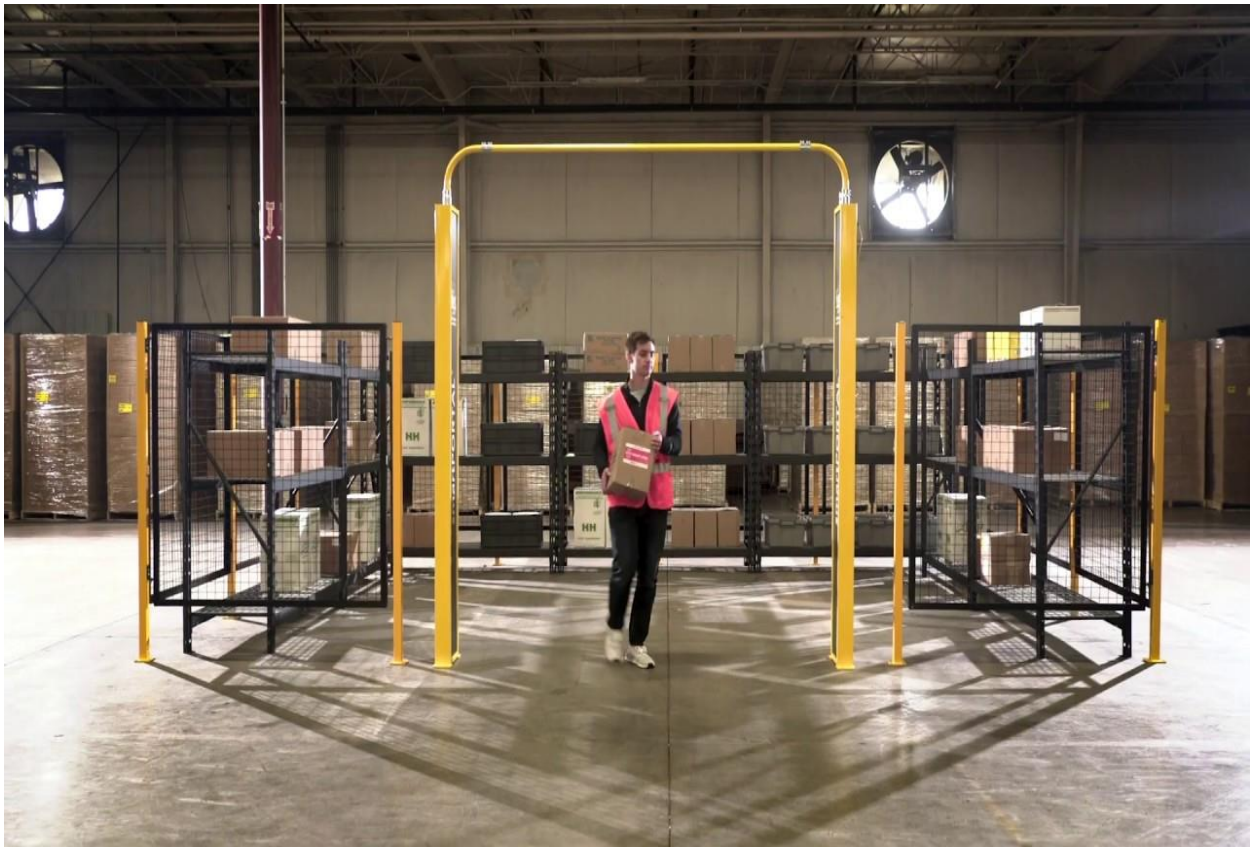
Table of Contents

Table of Figures

Chapter 1
Introduction

1.1 Overview

The Smart Inventory Management System project aims to automate stocktaking in large warehouses that suffer from issues in manual counting, such as human errors and delays in updating quantities. The system uses gates at the entry and exit gates, equipped with sensors and barcode scanners that automatically record all incoming and outgoing shipments. Additionally, shelves inside the warehouse are fitted with sensors that detect empty and filled spaces. The data is displayed in real-time on a dedicated website, helping warehouse staff monitor inventory efficiently and improve warehouse management operations.



1.2 Problem Statement

In large warehouses, inventory counting is performed manually by humans. This method causes many problems, such as human errors, time loss, and inaccuracies in knowing the actual quantities available or missing from shipments. These problems become more pronounced, especially in warehouses operating around the clock and dealing with large volumes of inventory, like those used by companies such as Amazon and Aramex. The primary beneficiaries of this system are large warehouse owners and managers who face challenges in tracking the movement of shipments inside and outside the warehouse. This system targets large companies that rely on massive warehouses and need accuracy and speed in inventory counting and management. The smart inventory management system that I will develop is a comprehensive solution aimed at eliminating the problems caused by manual stocktaking. This system relies on automating the process using modern technologies such as gates, sensors, barcodes, and IR or ultrasonic sensors or RFID . These tools will automatically track shipments as they enter or exit the warehouse and update the data in the system without the need for human intervention. This system will be implemented in any large

warehouse that handles massive quantities of products, regardless of the type of shipment or the size of the warehouse. The system will operate at the warehouse's entry and exit gates, as well as inside the warehouse on the shelves themselves to track available shipments and identify empty spaces. The system will be ready to operate 24/7, allowing the tracking of any product entering or leaving the warehouse at any time during the day. The entire process will be automated and will occur in real time, meaning warehouse workers will have continuous updates on inventory at any given moment. Manual inventory counting is inefficient and causes many problems such as human errors, delays, and inaccuracies in determining actual inventory quantities. This leads to

operational inefficiencies and delays in fulfilling orders or even losing shipments. Automating the counting process will save time, ensure accuracy, and reduce the need for human intervention, significantly improving overall operational efficiency.

HOW:

The system will rely on a gate at the warehouse's entry gate and another at the exit gate. Shipments will move on these gates, which will be equipped with sensors , barcode scanners , RFID . The sensors and barcode scanners will read each shipment's barcode upon entry or exit and transmit this information to the central system.

Additionally, inside the warehouse, shelves will be equipped with sensors to measure whether they are full or empty. These sensors will send real-time data to the system to identify available storage spaces.

When a shipment leaves the warehouse, the quantities will be updated in the system, and the sensor on the shelf where the shipments was stored will inform the system that the shelf is now empty. All this data will be displayed on a dedicated website for warehouse staff.

The system will use communication protocols to connect the embedded systems that control the sensors and gates with the central system for display on the web.

1.3 Project Contribution

The Smart Inventory Management System project offers the following contributions:

1. **Automation of Inventory Management:**

   By replacing manual inventory counting with automated systems using sensors, barcode scanners, and RFID, the project eliminates human errors, reduces time delays, and enhances accuracy in tracking shipment movements.

2. **Real-time Data Tracking:**

   The system provides continuous, real-time updates on shipment entry, exit, and shelf occupancy. This ensures warehouse staff always have access to accurate and up-to-date inventory information.

3. **Improved Operational Efficiency:**

   Automating stocktaking minimizes delays and inefficiencies in order fulfillment, making operations smoother and faster. This benefits warehouses handling large volumes of inventory, like those of Amazon and Aramex.

4. **Scalable Solution for Various Warehouses:**

   The system is designed to function in warehouses of any size or shipment type, making it adaptable to different industries.

5. **24/7 Operation Capability:**

   The system can operate continuously without human intervention, supporting warehouses that function round the clock and ensuring uninterrupted inventory management.

6. **Web-based Monitoring Platform:**

   The integration of real-time data with a dedicated web application enhances user experience by enabling remote monitoring and management of inventory.

7. **Enhanced Space Utilization:**

   Shelf sensors help optimize storage space by identifying available and filled slots, improving warehouse organization and capacity usage.

By introducing these innovative features, the project aims to revolutionize warehouse management, reducing reliance on manual methods and paving the way for a smarter, more efficient approach to inventory tracking.

1.4 Project Tools

1.4.1 Hardware Tools

- ESP8266 WiFi Module
- Atmega32
- RC522 RFID Module
- IR Sensor
- Buzzer
- LEDs
- LCD
- Batteries / Voltage source
- Capacitors and Other Electronic Components
- Breadboard
- USB ASP
- Jumper Wires
- Datasheet

1.4.2 Software Tools

- Figma
- MSSQL
- Visual Studio
- PostMan
- Eclipse
- Simulide
- Proteus 8

1.5 Book Structure

This book is organized as follow:

- **Chapter 1: Introduction**
  - Briefly describes the project, its components and give overviews.
- **Chapter 2: Theoretical Background**
  - Describes theoretical background, hardware component and software interface we use in project.
- **Chapter 3: Analysis and Design**
  - This chapter discussed the architecture of our system.
- **Chapter 4: Implementation**
  - Describes the project configuration stages

Chapter 2
Theoretical Background

2.1 The Front-End

The front end refers to the part of a software application, website, or system that users interact with directly. It includes everything the user sees and interacts with, such as the layout, buttons, forms, images, and animations.[1]

The front end consists of three main technologies:

1. HTML (HyperText Markup Language) – Structures the content on the web page.

2. CSS (Cascading Style Sheets) – Styles the content, including colors, fonts, and layout.

3. JavaScript – Adds interactivity, such as dynamic content updates, animations, and user input handling.

Front-End Frameworks & Libraries:

1. React.js
2. Angular
3. Vue.js
4. Bootstrap

2.1.1 HTML

HTML (HyperText Markup Language) is the standard language used to create and structure web pages. It defines the elements of a webpage, such as text, images, links, buttons, and forms, allowing web browsers to display content correctly.[1]

Key Features of HTML:

1. Structure & Layout – Organizes content using headings, paragraphs, lists, and sections.
2. Hyperlinks & Navigation – Enables linking between different web pages.
3. Multimedia Support – Embeds images, audio, and video into web pages.
4. Forms & User Input – Allows users to enter data through input fields, buttons, and forms.
5. Semantic Elements – Uses meaningful tags to improve accessibility and SEO (e.g., headers, footers, articles).

Why is HTML Important?

- It is the foundation of all websites.
- Works with CSS for styling and JavaScript for interactivity.
- Ensures web pages are accessible and well-structured.

2.1.2 TypeScript

TypeScript is a programming language developed by Microsoft that extends JavaScript by adding static typing and other powerful features. It helps developers write more structured, maintainable, and error-free code, especially in large applications.[1]

Key Features of TypeScript:

1. Static Typing – Unlike JavaScript, TypeScript allows developers to specify data types, which helps catch errors during development.
2. Better Code Readability – Type definitions and object structures make the code easier to understand.
3. Object-Oriented Programming (OOP) – Supports features like classes, interfaces, and inheritance, making it suitable for complex applications.
4. Improved Developer Experience – Provides better auto-completion, refactoring tools, and debugging support in modern code editors.
5. Compatibility with JavaScript – TypeScript is a superset of JavaScript, meaning existing JavaScript code can be used without modification.
6. Compiles to JavaScript – Since browsers don't understand TypeScript directly, it needs to be compiled into standard JavaScript before running.

Why Use TypeScript?

- Fewer bugs: Detects and prevents errors before the code runs.
- Easier maintenance: The structured nature makes it easier to work on large projects.
- Great for teams: Helps developers collaborate more effectively with clear type definitions.
- Widely adopted: Used in popular frameworks like Angular, React, and Node.js.

2.1.3 Angular

Angular is a front-end web framework developed by Google for building dynamic, single-page web applications (SPAs). It is based on TypeScript and follows a component-based architecture, making web development more modular and scalable.[1]

Key Features of Angular:

1. Component-Based Architecture – Applications are built using reusable components, making development and maintenance easier.
2. Two-Way Data Binding – Automatically synchronizes data between the user interface and the application logic.
3. Dependency Injection – Efficiently manages dependencies to improve code flexibility and testing.
4. Directives & Templates – Extends HTML with additional functionality, enabling interactive and dynamic content.
5. Built-in Routing – Supports navigation between different views or pages without needing a full-page reload.
6. High Performance – Optimized rendering and built-in support for lazy loading improve application speed.

Why Use Angular?

- Ideal for building large-scale, enterprise-level applications.
- Strong community support and regular updates from Google.
- Provides a structured framework with built-in tools for testing and development.
- Works well with backend services and APIs for dynamic data handling.

2.2 The Back-end / Server

The backend is the part of a software application or website that operates behind the scenes, handling data processing, logic, and database interactions. It is responsible for managing requests from the front end, processing them, and returning the appropriate responses.[2]

Key Functions of the Backend:

1. Database Management – Stores, retrieves, and manipulates data using databases like MySQL, PostgreSQL, or MongoDB.
2. Server Logic – Executes business rules and processes data according to user requests.
3. Authentication & Security – Manages user logins, permissions, and data protection.
4. API Development – Provides a way for the front end to communicate with the backend using REST or GraphQL APIs.
5. Performance Optimization – Ensures fast and efficient processing of data and requests.

Backend Technologies:

- Programming Languages: Python, JavaScript (Node.js), Java, PHP, C#, Ruby
- Frameworks: Express.js, Django, Flask, Spring Boot, Laravel, .NET
- Databases: MySQL, PostgreSQL, MongoDB, Firebase
- Servers & Hosting: Apache, Nginx, AWS, Google Cloud

A server is a computer or system that provides services, resources, or data to other computers (clients) over a network. It processes requests from clients, such as web browsers or applications, and responds with the necessary data.[2]

Types of Servers:

1. Web Server – Hosts websites and delivers web pages (e.g., Apache, Nginx).
2. Database Server – Stores and manages data for applications (e.g., MySQL, PostgreSQL).
3. Application Server – Runs backend logic and processes (e.g., Node.js, Tomcat).
4. File Server – Stores and shares files over a network.
5. Mail Server – Handles email sending and receiving (e.g., Microsoft Exchange).

How Servers Work:

1. A client (like a web browser) sends a request to the server.
2. The server processes the request, retrieves data if needed, and sends a response.
3. The client receives and displays the response, such as a webpage or file.

Why Are Servers Important?

- Enable websites and applications to function.
- Store and manage large amounts of data.
- Provide security, reliability, and scalability for businesses.

2.2.1 .NET

.NET is a software development framework created by Microsoft for building various types of applications, including web, desktop, mobile, cloud, gaming, and IoT solutions. It provides a platform-independent environment for developers to write, compile, and run applications using multiple programming languages.[2]

Key Features of .NET:

1. Cross-Platform Development – Supports Windows, Linux, and macOS through .NET Core (now part of .NET 5+).
2. Multiple Language Support – Works with C#, F#, and VB.NET.
3. Robust Frameworks & Libraries – Provides built-in libraries for networking, security, data access, and UI development.
4. Scalability & Performance – Optimized for high-performance applications, including cloud and enterprise systems.
5. Security & Reliability – Includes features like authentication, encryption, and secure data handling.
6. Integration with Microsoft Ecosystem – Works seamlessly with Azure, SQL Server, and Microsoft 365.

.NET Variants:

- .NET Framework – The original version, mainly for Windows applications.
- .NET Core – A cross-platform, open-source framework for modern applications (merged into .NET 5+).
- ASP.NET – A framework for building web applications and APIs.

Why Use .NET?

- Strong community support and long-term Microsoft backing.
- High performance, especially for web and API development.

2.3 Database

A database is an organized collection of data that is stored, managed, and accessed electronically. It allows users and applications to efficiently store, retrieve, update, and delete data. Databases are essential for applications, websites, and enterprise systems that handle large amounts of structured or unstructured data.[3]

Key Features of Databases:

1. Data Storage – Keeps information in an organized manner.
2. Data Retrieval – Allows users to access and search for specific data.
3. Data Management – Supports updates, deletions, and modifications of records.
4. Security & Access Control – Ensures only authorized users can access or modify data.
5. Scalability – Handles increasing amounts of data efficiently.

Types of Databases:

1. Relational Databases (RDBMS) – Uses structured tables and SQL (e.g., MySQL, PostgreSQL, SQL Server).
2. NoSQL Databases – Handles unstructured or semi-structured data (e.g., MongoDB, Firebase, Cassandra).
3. Cloud Databases – Hosted on cloud platforms for scalability and remote access (e.g., Amazon RDS, Google Cloud Firestore).
4. Graph Databases – Focuses on relationships between data points (e.g., Neo4j).

Why Are Databases Important?

- Enable efficient data storage and retrieval.
- Support business applications, websites, and mobile apps.
- Ensure data consistency, security, and integrity.

2.3.1 MSSQL

MSSQL (Microsoft SQL Server) is a relational database management system (RDBMS) developed by Microsoft. It is designed to store, retrieve, and manage structured data efficiently, making it widely used for business applications, web development, and enterprise solutions.[3]

Key Features of MSSQL:

1. Relational Database Structure – Stores data in tables with predefined relationships.
2. T-SQL (Transact-SQL) – A powerful query language used for database operations.
3. Security & Encryption – Includes authentication, role-based access control, and encryption for data protection.
4. High Performance & Scalability – Supports large datasets and enterprise-level applications.
5. Backup & Recovery – Provides automated backup, recovery, and disaster recovery features.
6. Integration with Microsoft Tools – Works well with Power BI, Azure, .NET, and other Microsoft services.

Editions of MSSQL:

- Express – Free, lightweight version for small applications.
- Standard – Suitable for medium-sized businesses.
- Enterprise – High-performance edition for large-scale applications.
- Azure SQL Database – Cloud-based version hosted on Microsoft Azure.

Why Use MSSQL?

- Reliable for handling large amounts of structured data.
- Strong security and compliance features.
- Works seamlessly with Microsoft technologies.

Chapter 3
Analysis and Design

3.1 The requirements

Purpose:

- The purpose of this project is to develop a '**Smart Inventory Management System'** that automates and streamlines the process of tracking and managing inventory in large warehouses. The system aims to eliminate the inefficiencies and inaccuracies associated with manual inventory counting, such as human errors, time delays, and incorrect stock data. By leveraging modern technologies such as barcode scanners, RFID, sensors, and real-time data updates, the system will provide warehouse owners and managers with accurate, up-to-date information about their inventory. This will improve operational efficiency, reduce costs, and enhance customer satisfaction by ensuring timely and accurate order fulfillment.

Project Scope:

- The scope of the project includes the development of a comprehensive inventory management system designed for large warehouses handling high volumes of shipments. The system will operate at warehouse entry and exit points, as well as within the storage areas, to track shipments in real-time. Key components of the system include:

    - Automated Gates: Equipped with barcode scanners and RFID readers to capture shipment data upon entry or exit.
    - Shelf Sensors: Installed on shelves to detect whether they are full or empty and provide real-time updates.
    - Central System: A web-based interface for warehouse staff to access and manage inventory data.
    - Real-Time Updates: Continuous monitoring and updating of inventory levels as shipments move in and out of the warehouse.

Functional Requirements:

These requirements describe the tasks the system must perform and the features it must provide.

1. **Automated Inventory Tracking:**
    - Use barcode scanners, RFID, and sensors to automatically track shipments as they enter or exit the warehouse.
    - Provide real-time updates to the central system to reflect changes in inventory levels.
2. **Product Management:**
    - Add, update, and delete products with details such as name, serial number, category, shelf, and warehouse.
    - Display a list of all products with their current status (available, out of stock, faulty).
3. **Category Management:**
    - Create and manage product categories (e.g., Max, Min, Medium).
    - Update or delete categories as needed.
4. **Shelf Management:**
    - Add and manage shelves with details such as shelf name, availability status, and associated warehouse.
    - Track which shelves are full or empty in real-time.
5. **Warehouse Management:**
    - Add and manage warehouses with details such as name, location, and capacity.
    - Display a list of all warehouses and their current inventory levels.

6. **User Management:**
    - Register and manage users with different roles (Admin, Data Entry, User).
    - Assign permissions based on user roles to control access to system features.
7. **Real-Time Reporting:**
    - Generate real-time reports on inventory levels, product availability, and faulty devices.
    - Provide a dashboard for warehouse managers to monitor key metrics.

Non-Functional Requirements:

These requirements describe how the system should perform in terms of performance, security, and usability.

1. **Performance:**
    - The system should operate smoothly with a large number of shipment and warehouses without delays.
    - Data should be updated in real-time (Real-Time Updates) when shipments are added or modified.
2. **Security:**
    - The system should be protected with strong passwords and data encryption.
    - There should be role-based access control (Role-Based Access Control) to prevent regular users from accessing system settings.

3. Usability:

- The user interface should be simple and easy to use, even for non-technical users.
- There should be clear instructions for users when adding new shipments or warehouses.

4. Compatibility:

- The system should be compatible with different browsers (Chrome, Firefox, Safari) and devices (computer, mobile).

5. Availability:

- The system should be available 24/7 with minimal downtime.

3.2 Object-oriented analysis (OOA)

Object-oriented analysis (OOA) is the initial phase of object-oriented analysis and design (OOAD), focusing on understanding and defining the problem domain and system requirements by identifying objects, their attributes, methods, and relationships. It's a way to model real-world entities and their interactions as software objects. The goal is to create a conceptual model of the system before moving into the design and implementation phases.

3.2.1 Sequence Diagram

- The diagram illustrates the sequence of tracking a shipment within the smart inventory management system, starting from user login and shipment creation, followed by identifying available shelf space using sensors. It then shows the shipment's entry into the warehouse using barcode or RFID verification, automatic storage status updates, and finally, the shipment's exit with real-time data synchronization to the central system. This ensures accurate tracking, efficient storage, and seamless warehouse operations. (show figure 3.1(1)(2))

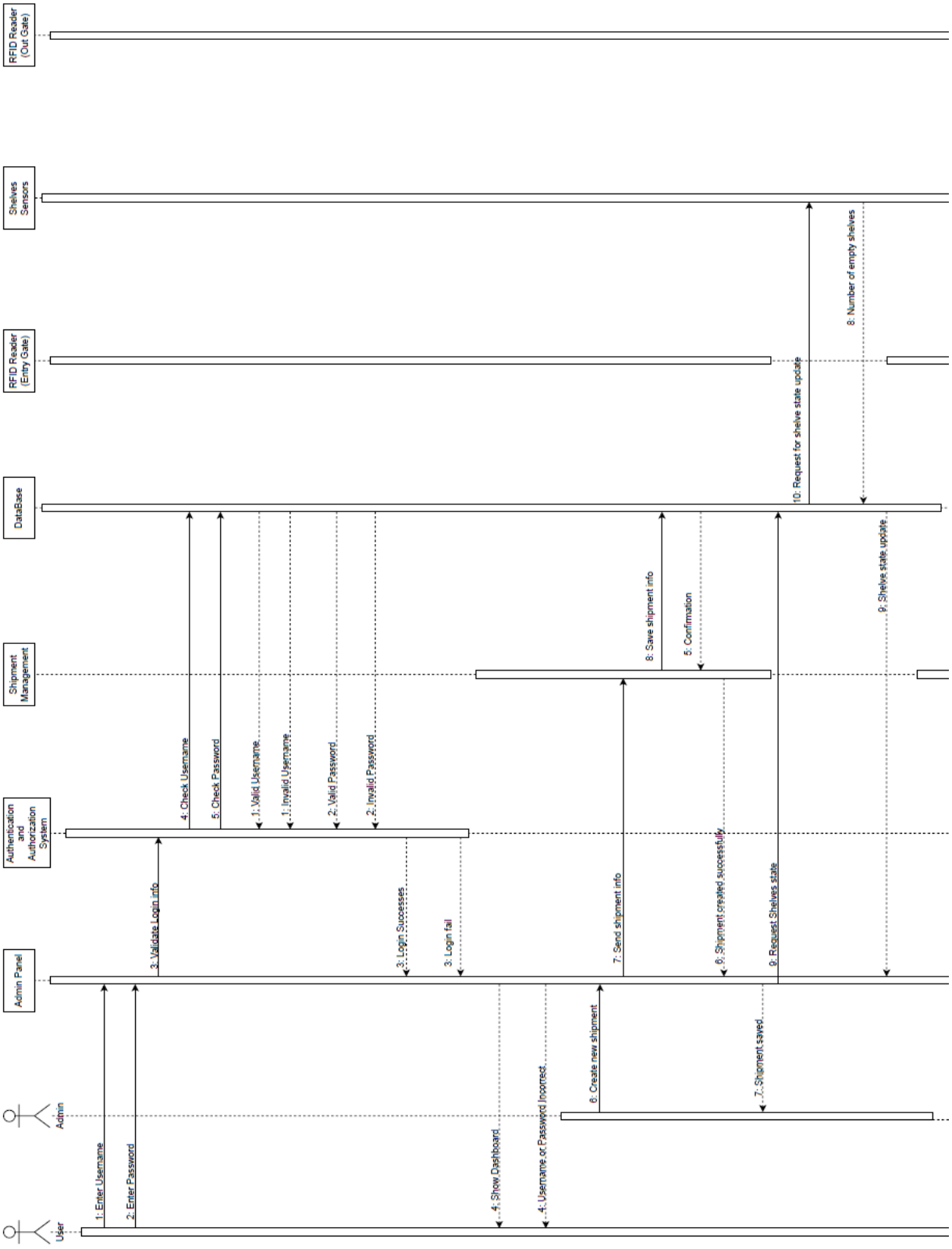


Figure 3.1: Sequence Diagram (1)

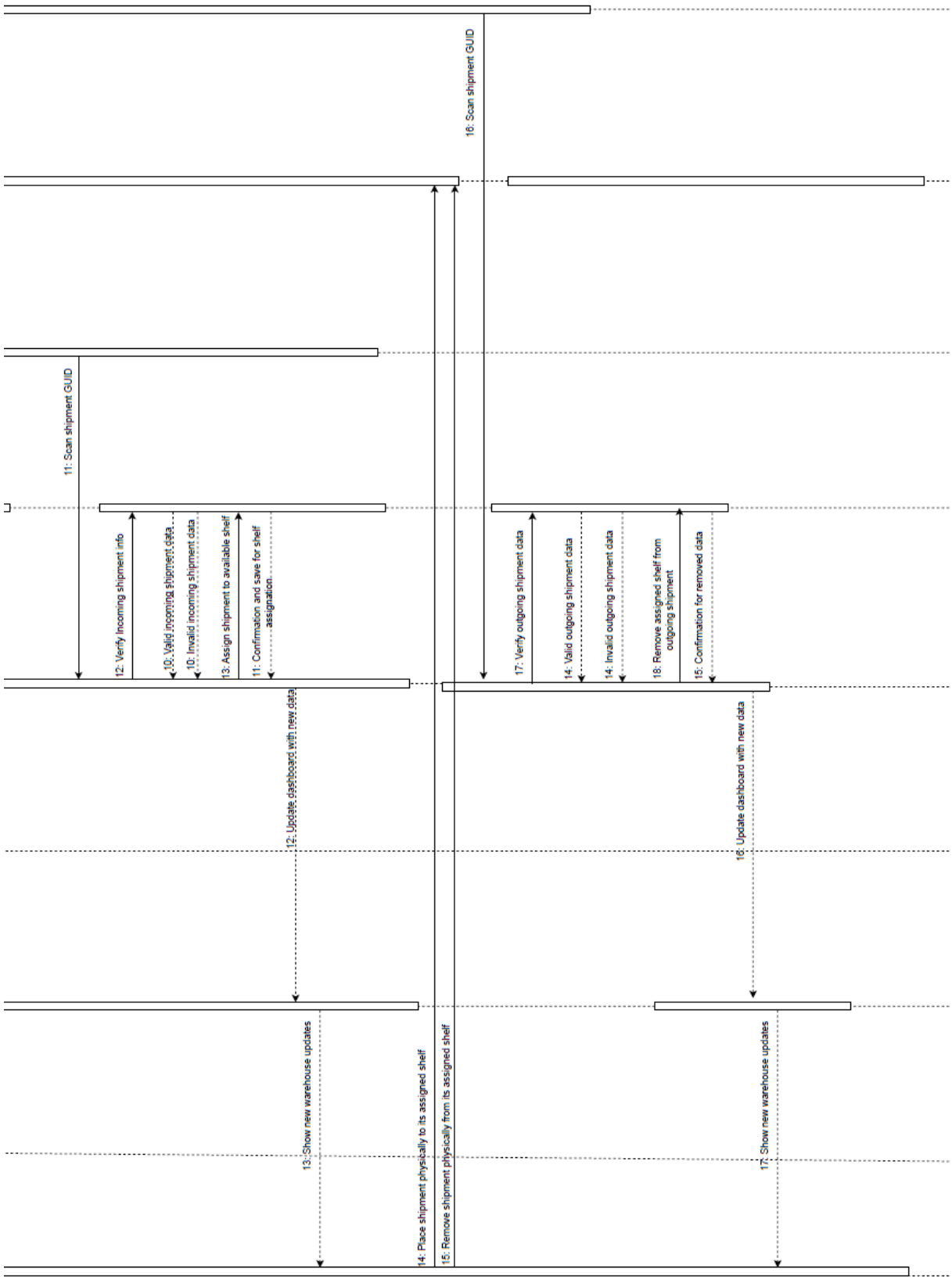


Figure 3.1: Sequence Diagram (2)

All diagrams are uploaded to GitHub in good resolution

3.2.2 Activity Diagram

- The Activity Diagram shows the process of handling incoming and outgoing shipments in the smart inventory system. It starts with creating a warehouse and entering its data, followed by creating a new shipment. The system checks for an empty shelf—if none is found, it alerts the staff. If space is available, the shipment's RFID is scanned, verified, and assigned to the shelf. The system then updates the inventory and displays details such as total categories, total units, and availability. Upon shipment exit, the system scans and verifies the shipment ID, updates the database, and shows the updated inventory status. (show figure 3.2(1)(2))

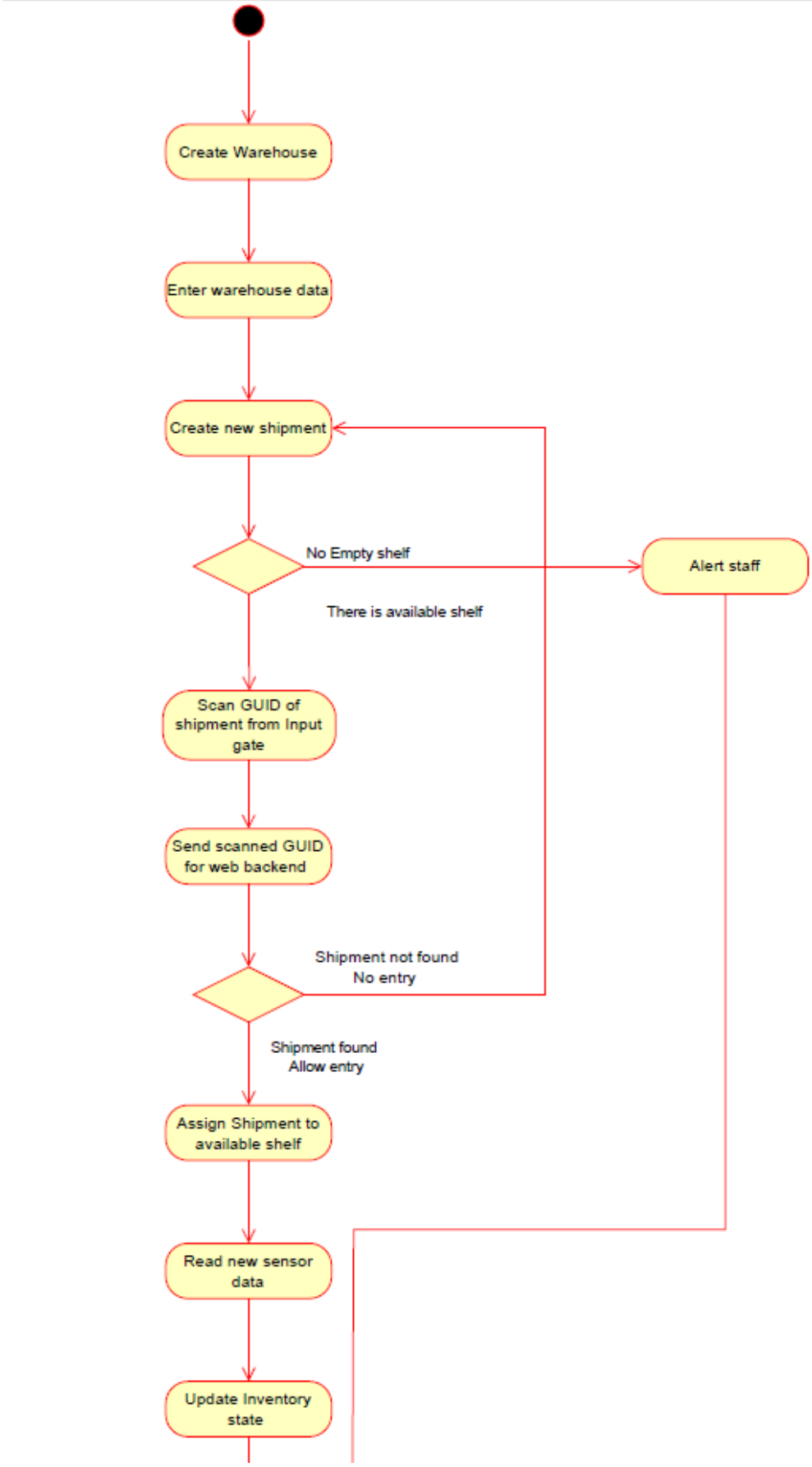


Figure 3.2: Activity Diagram (1)

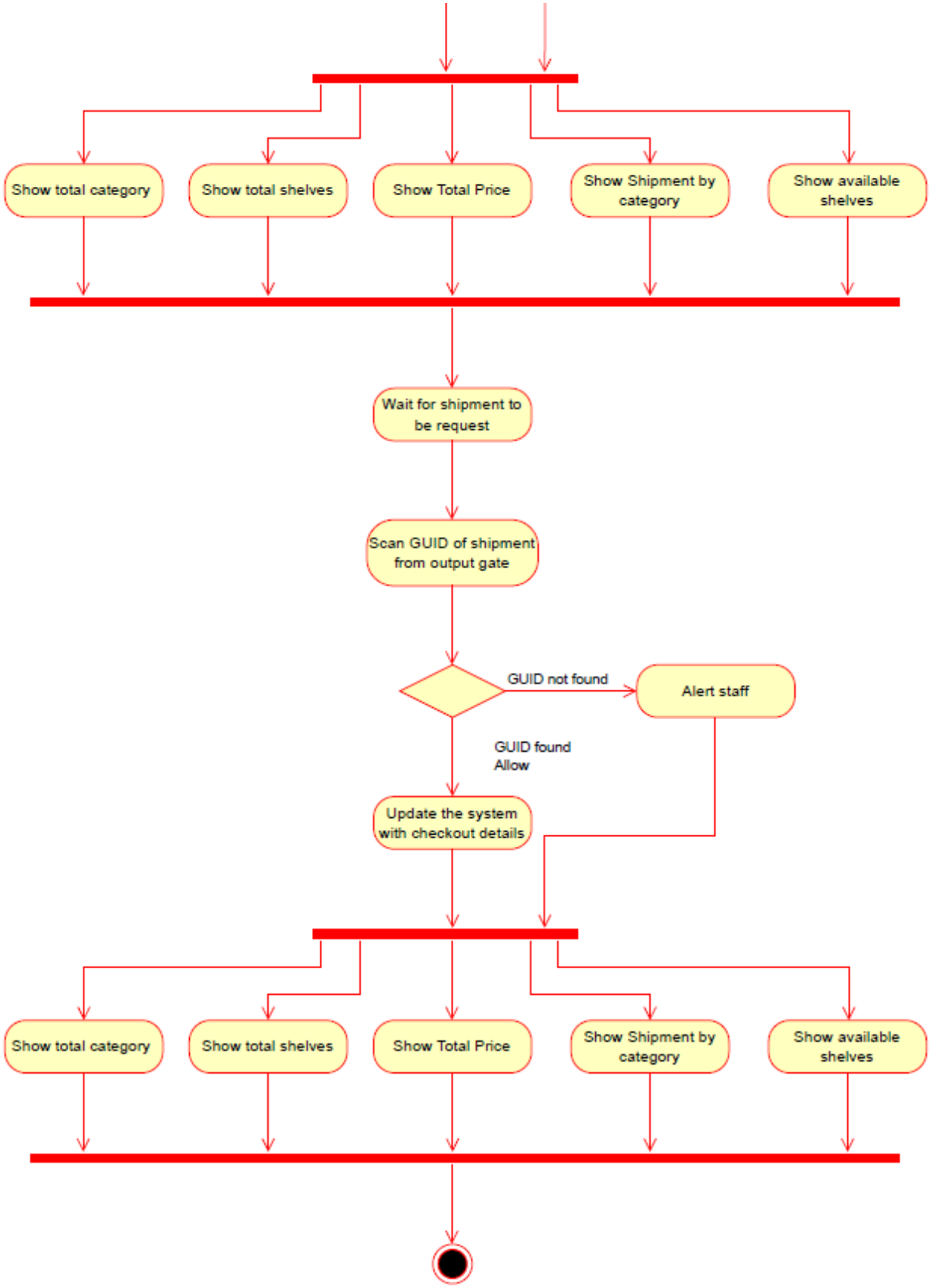


Figure 3.2: Activity Diagram (2)

All diagrams are uploaded to GitHub in good resolution

3.2.3 Use case Diagrams

- These use case diagrams illustrate the core functionalities of the Smart Inventory Management System. The main use case covers actions such as logging in, managing shipments, tracking their movement in and out, real-time inventory updates, error alerts, and displaying data on a web dashboard. Admins and users interact with the system through an admin panel and warehouse interface. The first sub-use case focuses on how admins manage shipment data—creating, updating, deleting shipments, and generating unique identifiers. The second sub-use case details the process of handling incoming shipments using RFID and IR sensors to scan, assign, and update inventory automatically in real time. (show figure 3.3,3.4)

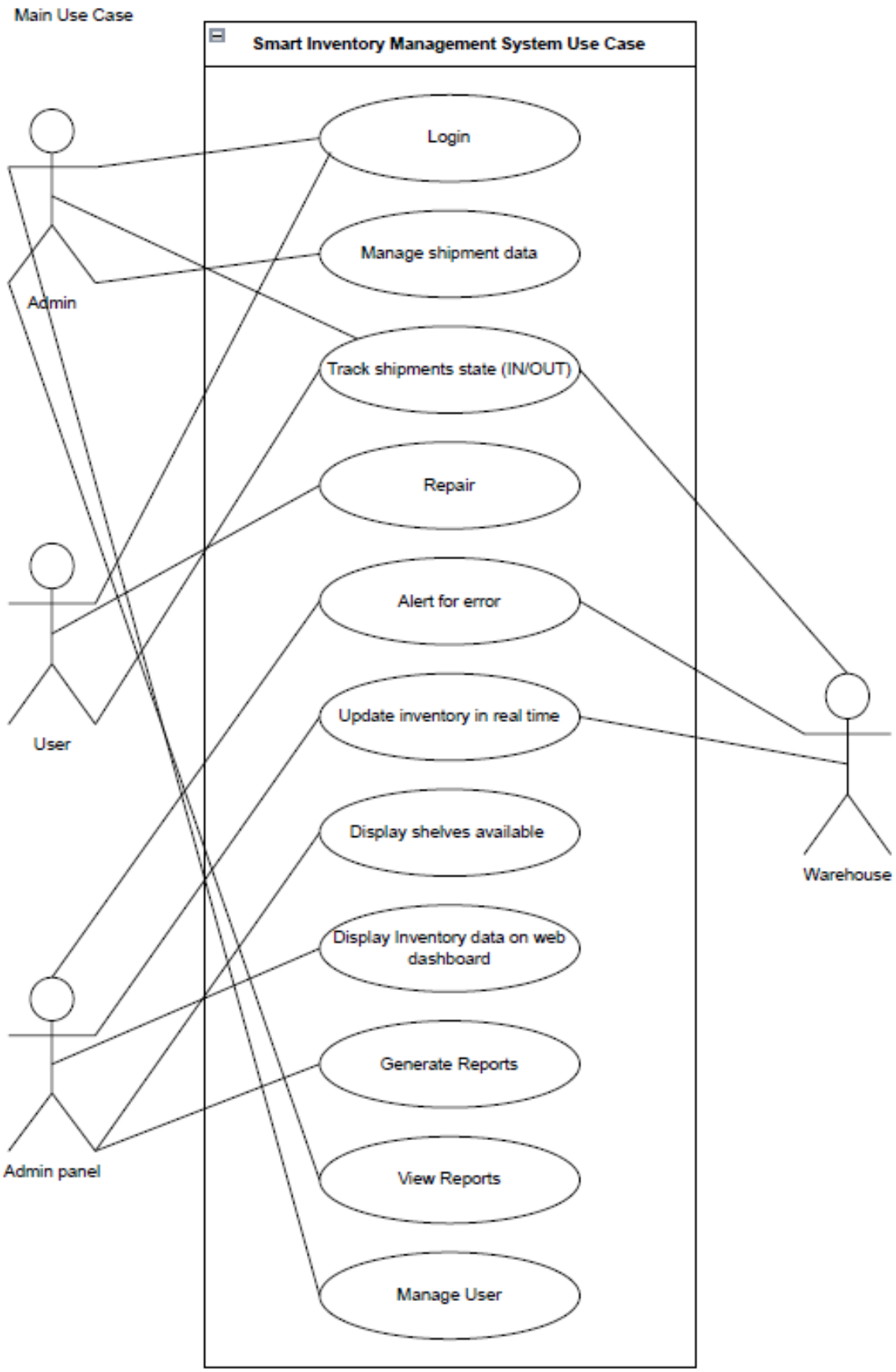
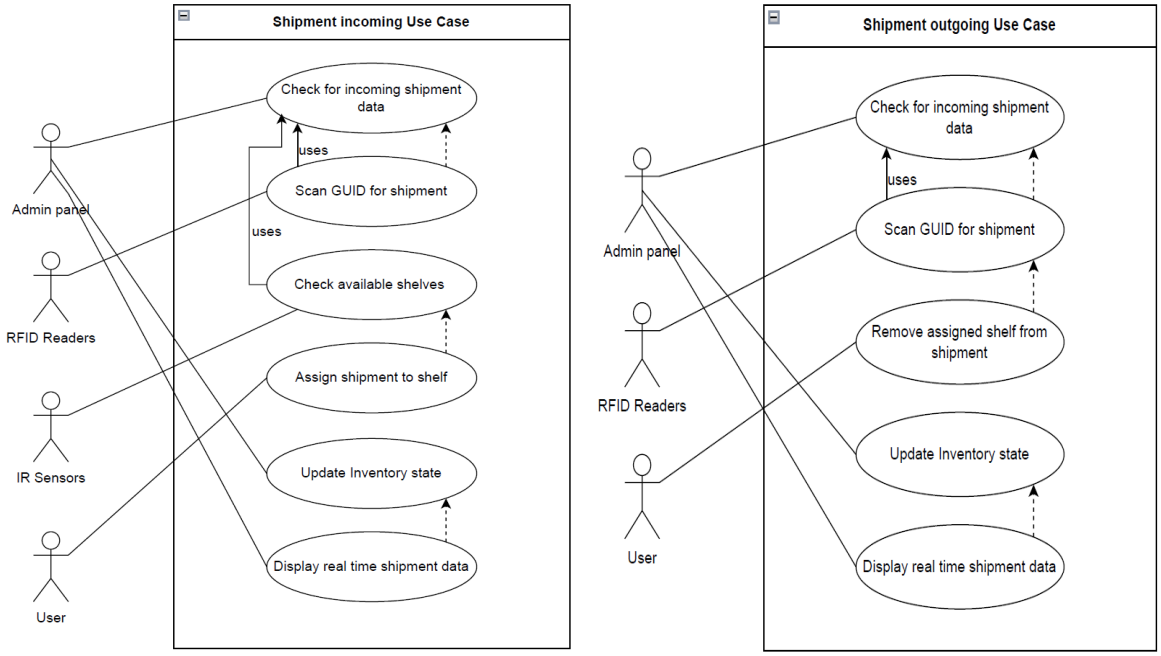


Figure 3.3: Main Use case Diagram



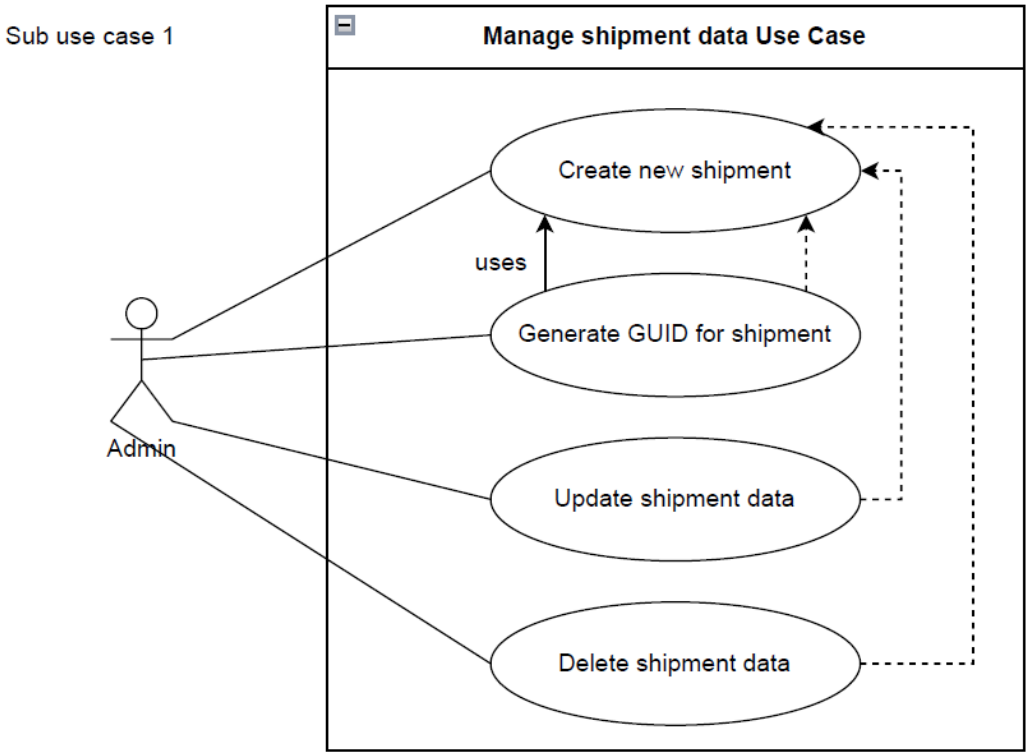


Figure 3.4: Sub Use case Diagram

All diagrams are uploaded to GitHub in good resolution

3.2.4 Context Diagram

- The context diagram represents the Smart Inventory Management System as a single high-level process that interacts with various external entities. These entities include the Warehouse Staff, who interact with the system through tasks like logging in, creating shipments, and viewing inventory status. The RFID/Barcode Scanner System communicates automatically with the system to track shipments entering and leaving the warehouse. The Shelf Sensors provide real-time data about available or occupied storage spaces. The Web Interface acts as a platform for staff to monitor and manage operations. The diagram shows how all these external components exchange data with the system, emphasizing the system's role in automating and coordinating warehouse inventory processes. (show figure 3.5)

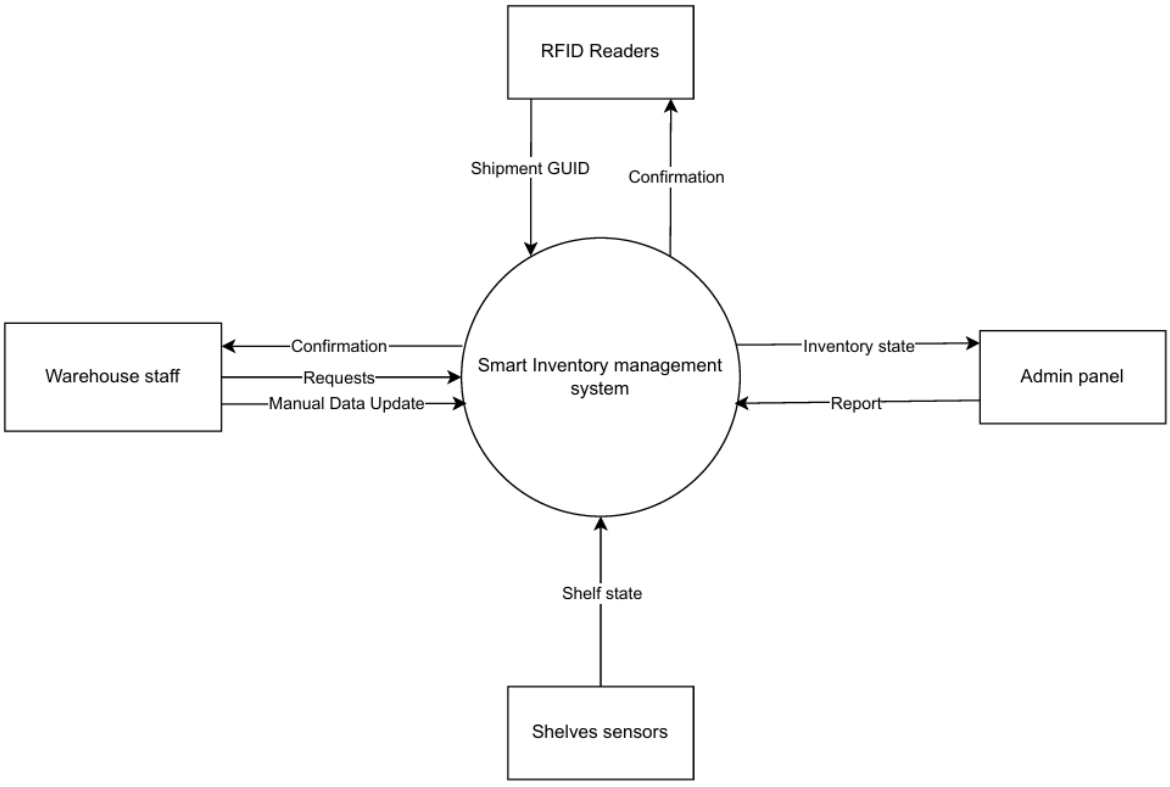


Figure 3.5: Context Diagram

3.2.5 Mapping

- The mapping in the smart inventory management system defines how different system components interact and correspond to one another. It connects the front-end interfaces (such as login, dashboard, and shipment creation) with backend functionalities like user authentication, shipment data storage, and shelf availability checks. Each action performed by the user on the website (e.g., creating a shipment, checking inventory status) maps to specific operations in the database and is linked to signals from physical sensors (e.g., RFID, barcode readers, IR sensors) that track shipment entry, exit, and shelf status. This mapping ensures seamless communication between the user interface, system logic, and hardware components to maintain real-time inventory accuracy. (show figure 3.6)

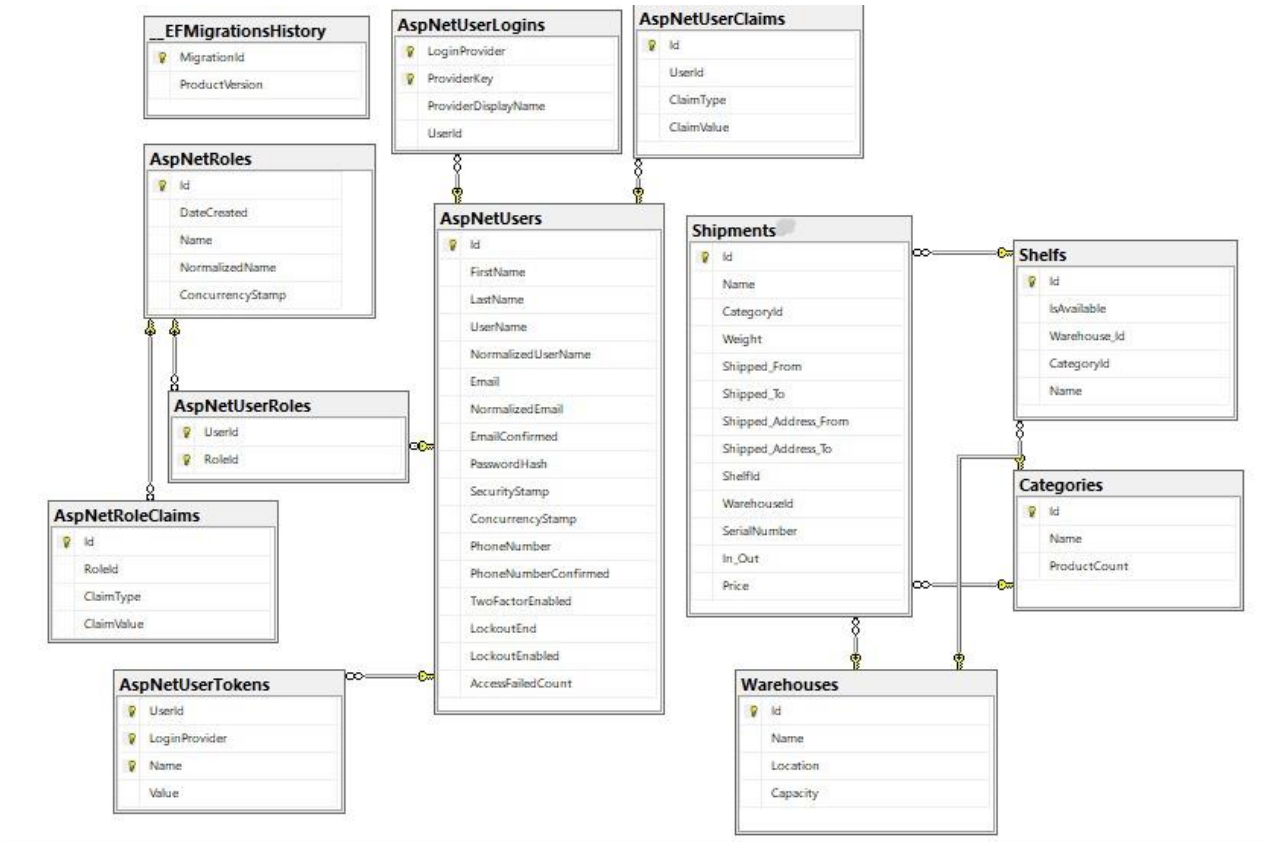


Figure 3.6: Mapping

All diagrams are uploaded to GitHub in good resolution

3.2.6 ERD diagram

(Entity-Relationship Diagram - ERD)

- The ERD illustrates the database structure of the smart inventory management system, highlighting the key entities and how they are connected. The main entities include Users, Shipments, Warehouses, Shelves, Inventory Records, and Categories. A User can create multiple Shipments, each Shipment is stored on a specific Shelf, and each Shelf belongs to a Warehouse. Shipments are also categorized by type through the Category entity. An Inventory Record tracks the status and movement of each Shipment in real-time. The relationships between these entities ensure smooth tracking of inventory flow, warehouse space availability, and shipment classification. (show figure 3.7)

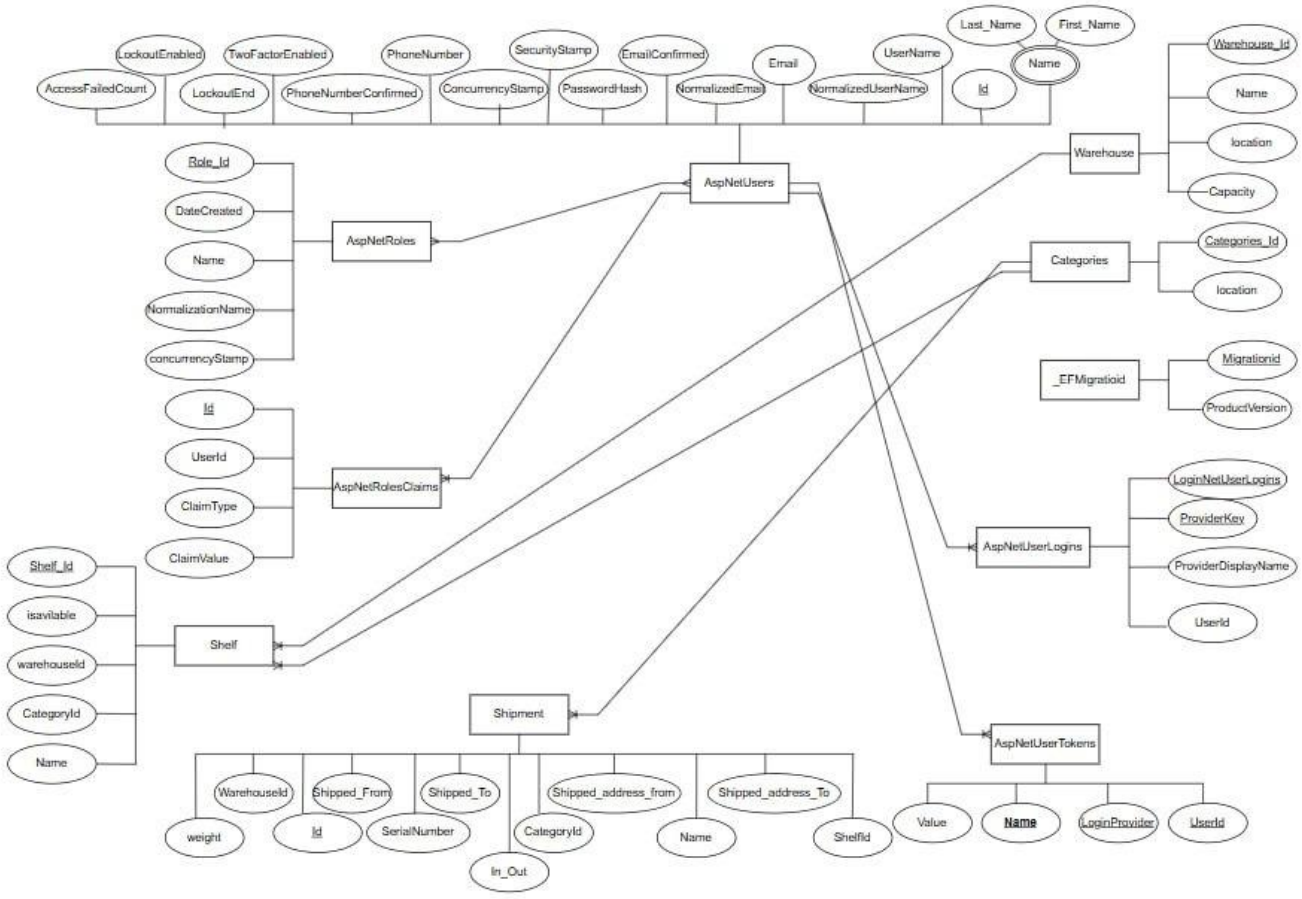
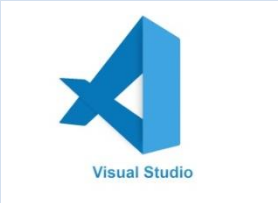


Figure 3.7: ERD diagram

All diagrams are uploaded to GitHub in good resolution

3.3 Software Tools

Software Tools	Description
Figma 	Figma is a cloud-based design tool used for creating user interfaces, prototypes, and collaborative design projects. It allows multiple users to work simultaneously on the same project, making it ideal for team collaboration. Its cross-platform accessibility ensures seamless work across devices. Figma is widely used by UI/UX designers for its ease of use and robust features.
MSSQL 	Microsoft SQL Server (MSSQL) is a relational database management system (RDBMS) developed by Microsoft. MSSQL supports advanced features like data analytics, business intelligence, and transaction processing. MSSQL is a powerful tool for database administrators and developers.
Visual studio (front end) 	Visual Studio is a comprehensive integrated development environment (IDE) by Microsoft, widely used for front-end development. It supports languages like HTML, CSS, JavaScript, and frameworks like Angular, React, and Vue.js. Visual Studio provides tools for debugging, code editing, and version control, making it a favorite among front-end developers.
Visual studio (back end) 	Visual Studio is also a powerful tool for back-end development, supporting languages like C#, VB.NET, and Python. It offers robust tools for building APIs, server-side applications, and database integration. Visual Studio provides debugging, testing, and deployment capabilities, ensuring efficient back-end development.

Postman	Postman is a popular API development and testing tool used by developers to design, test, and document APIs. It allows users to send HTTP requests, analyze responses, and automate testing workflows. Postman supports collaboration through shared workspaces and collections, making it ideal for team projects. Its user-friendly interface and extensive features simplify API development.
Eclipse	Eclipse is an open-source IDE primarily used for Java development but supports other languages through plugins. It provides tools for coding, debugging, and testing, making it a versatile choice for developers. Eclipse is known for its extensibility, with a vast ecosystem of plugins for additional functionalities. It is widely used in enterprise environments for its flexibility and robustness.
Simulide	SimulIDE is an open-source electronics circuit simulator used for designing and testing electronic circuits. It supports microcontrollers, PIC, AVR, and Arduino, making it ideal for hobbyists and educators. SimulIDE provides a user-friendly interface for simulating and debugging circuits in real-time. It is a lightweight tool that simplifies the process of learning and experimenting with electronics.
Proteus 8	Proteus 8 is a professional-grade software for electronic design automation (EDA), used for circuit simulation and PCB design. It combines schematic capture, simulation, and PCB layout tools in a single platform. Proteus supports microcontrollers, SPICE simulation, and real-time debugging, making it a comprehensive tool for engineers. It is widely used in industries for designing and testing complex electronic systems. Proteus 8 is a powerful solution for electronics design and development.

3.4 Hardware Tools

Hardware in a project refers to the physical components of a system, such as computers, devices, and other physical equipment. It encompasses the tangible parts that enable the project to function and achieve its goals. In essence, hardware is the "stuff" that makes the project work.

3.4.1 Microcontroller

In these diagrams, we show the hardware requirements of the system. The system must check the shipment, if it is input or output, and scan the warehouse ID, and scan the shipment ID, and check the number of empty shelves, and send this data to the Wi-Fi module, that is the NodeMCU ESP8266. Then, the second diagram shows the hardware requirements that will be used with the main AVR processor based microcontroller Atmega32, 6 IR sensors, and buzzer, and pushbutton, and 6 RGB LEDs, and an LCD. The third diagram shows the hardware used with the NodeMCU ESP8266. Two RFID, one in the input gate, and one on the output gate, and one buzzer. (show figure 3.8)[4]

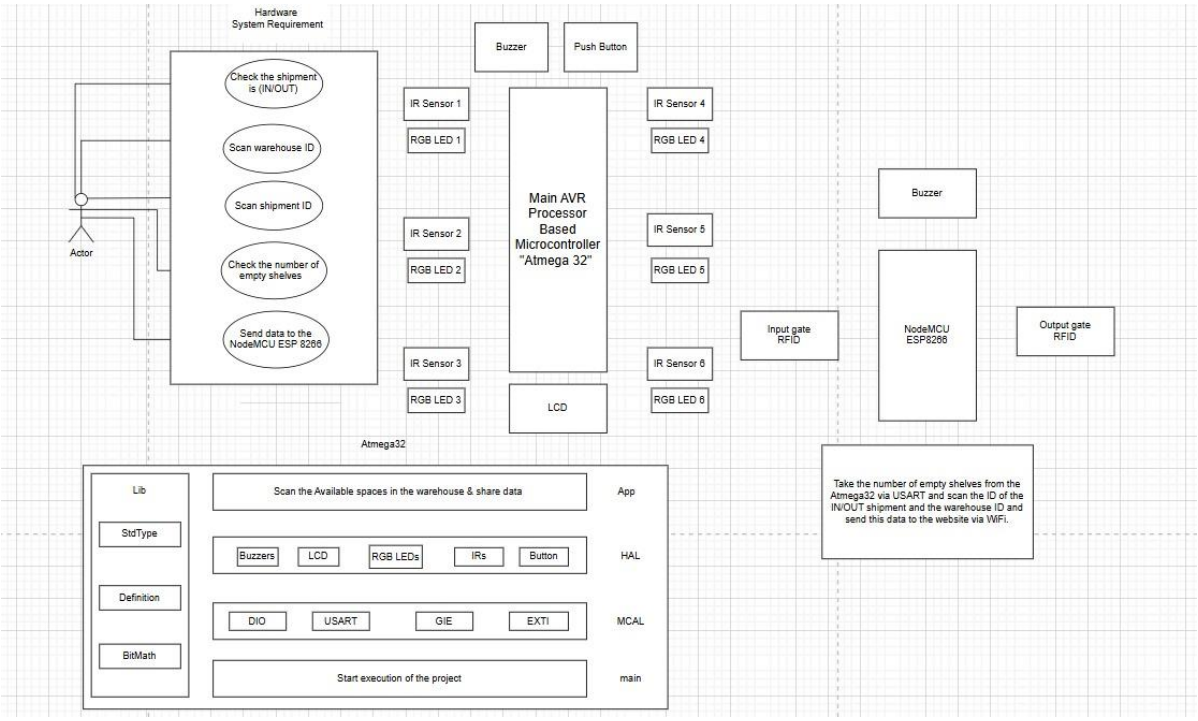


Figure 3.8: hardware requirements

All diagrams are uploaded to GitHub in good resolution

3.4.2 NodeMCU ESP 8266 WiFi Module

The NodeMCU ESP8266: is a WiFi-enabled microcontroller used for wireless communication in your inventory management system. It acts as a bridge between the ATmega32 microcontroller and a web server/cloud database, allowing real-time tracking of inventory data.[4]

- **Communication with ATmega32 (Serial/UART):** ATmega32 sends sensor data to NodeMCU via UART (TX/RX pins). NodeMCU processes and forwards this data over WiFi. Can also receive control signals from the server and send them to ATmega32.
- **Wireless Data Transmission (Cloud Communication):** Sends RFID product ID and shelf status (from IR sensors) to a web server/database.
- **Uses HTTP for communication:** Can host a simple web-based dashboard to display inventory status. (show figure 3.9)
-

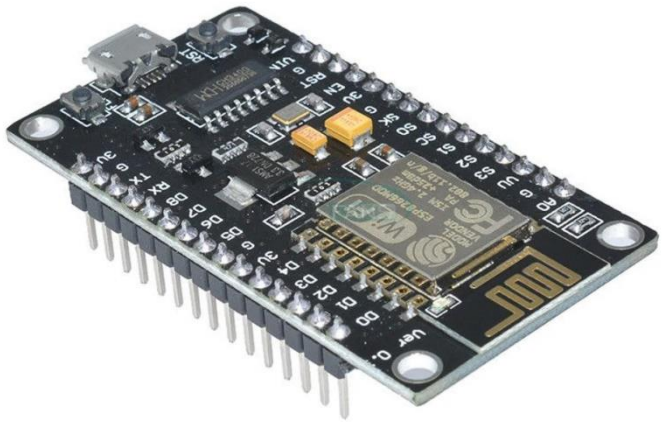


Figure 3.9: WIFI Module

3.4.3 Atmega32

Role of ATmega32 in the Project: The ATmega32 serves as the main controller of the smart inventory management system. It processes data from sensors, communicates with external modules, and controls system outputs such as LCD, buzzer, and LEDs.[4]

- **Monitoring Shelf Status (Stock Availability):** IR sensors detect whether a shelf is empty or occupied.
- **The ATmega32:** reads IR sensor inputs and updates the inventory status accordingly, If a product is placed or removed, the microcontroller processes this event.
- **Communicating with WiFi Module (ESP8266/NodeMCU):** The ATmega32 Kit sends RFID data and shelf status to an online web server/database using the ESP8266 WiFi module. Uses USART communication (TX/RX pins) to transmit data. Receives updates from the server if required.
- **Displaying Information (User Interface):** The LCD (16x2 or 20x4) displays real-time inventory data, such as: Shelf Status (Empty/Occupied) System Warnings or Alerts Uses 8-bit parallel communication to update the display.
- **Alerting Users (LEDs & Buzzer Notifications):** LED Indicators: Show system status (shelf full, error, or new scan detected). Buzzer: Alerts users when all the shelfs are full.
- **Handling User Inputs (If Any Buttons are Used):** push buttons are included (reset button, manual entry button), the ATmega32 handles user input to control certain functions. (show figure 3.10)

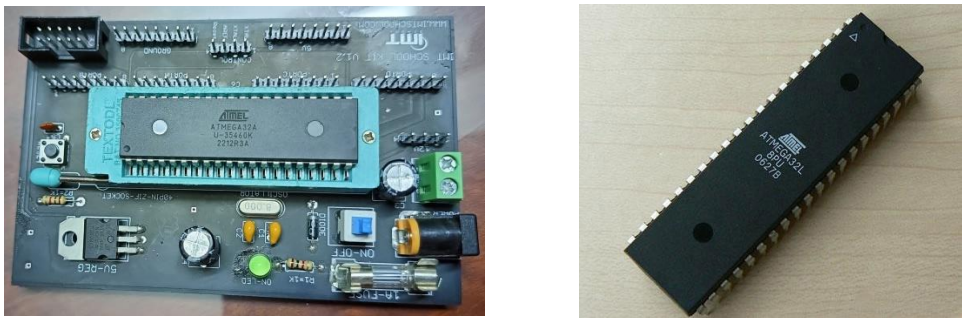


Figure 3.10: ATmega 32 Microcontroller

3.4.4 RC522 RFID Module

The RC522 RFID module is an alternative to the EM-18 RFID reader, used for reading RFID tags and identifying products in the inventory. Its primary functions include:

- Reading and Writing RFID Tags: Unlike the EM-18, the RC522 can both read and write data to RFID tags, making it useful for dynamic inventory updates.
- Communicating with the Microcontroller: It uses the SPI, I2C, or UART* interface to send scanned tag information to the Atmega 128.
- Efficient and Low Power: It operates at 13.56 MHz and consumes less power, making it suitable for embedded applications.

If you need two-way communication with RFID tags (reading and writing), the RC522 is a better choice than the EM-18, which only reads tags. (show figure 3.11)[5]

Figure 3.11: RC522 RFID Sensor

3.4.5 IR Sensor

The IR (Infrared) sensor is used to monitor the status of shelves in the inventory. Its primary functions include:

- Detecting whether a shelf is occupied or empty by sensing the presence or absence of products.
- Sending shelf status information to the Atmega 128 microcontroller for processing.
- Enabling real-time updates on the availability of storage space, which can be transmitted to the web server.

The IR sensor is essential for automating shelf monitoring, ensuring efficient space management and reducing the need for manual checks. (show figure 3.12)[5]

Figure 3.12: IR Sensors

3.4.6 Buzzer

The buzzer is used as an audio feedback mechanism in the smart inventory management system. Its primary functions include:

- Alerting users about specific events, such as successful or unsuccessful product scans, errors, or system notifications.
- Indicating critical conditions, such as empty shelves, misplaced products, or connectivity issues.
- Enhancing user interaction by providing audible signals for actions requiring attention.

The buzzer ensures that users are promptly notified of system states or issues, improving usability and efficiency. (show figure 3.13)[5]

Figure 3.13: Buzzer

3.4.7 RGB LEDs

The RGB LEDs are used as visual indicators to convey the system's status and provide feedback to the users. Their primary functions include:

- Indicating the operational status of the system, such as power-on or connectivity.
- Signaling specific events, like successful product scans, empty shelves, or errors, using different colors or blink patterns.
- Providing quick and intuitive feedback to users for actions like placing products on shelves or confirming operations.

The RGB LEDs improve user interaction by offering clear and immediate status updates, enhancing the overall efficiency of the system. (show figure 3.14)[5]

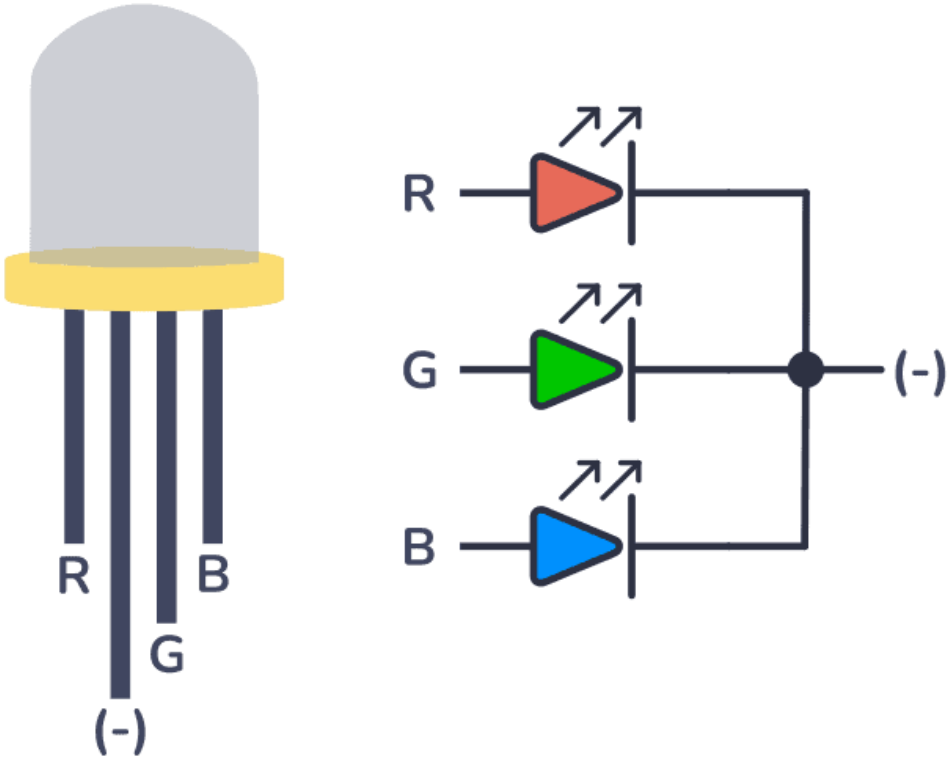


Figure 3.14: LEDS

3.4.8 LCD

The LCD (Liquid Crystal Display) is used as a visual feedback interface in the smart inventory management system. Its primary functions are:

- Displaying real-time information about the system, such as product IDs, shelf statuses, and system alerts.
- Showing user-friendly messages to guide inventory operators, like "Scan Successful," "Shelf Empty," or "Connection Error."
- Assisting in debugging and monitoring by displaying critical data during system operation.

The LCD enhances the usability of the system by providing clear and immediate visual feedback to users. (show figure 3.15)[5]

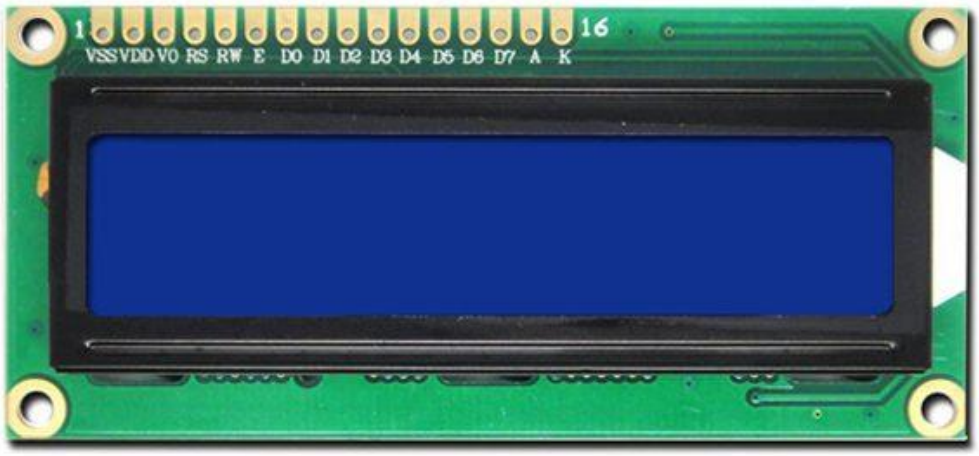


Figure 3.15: 16*2 LCD

3.4.9 Batteries / Voltage source

Batteries or Voltage source are used to power the various components of the smart inventory management system. Their primary functions include:

- Providing the necessary power for the microcontroller (Atmega 128), sensors (EM-18 RFID, IR), modules (ESP WiFi), and user interfaces (LCD, LEDs, buzzers).
- Ensuring portability and flexibility in system deployment, especially in areas where a stable power source might not be available.
- Allowing the system to function independently, maintaining operations even during power interruptions.

The batteries or Voltage source are crucial for ensuring continuous and reliable system performance, especially in a mobile or remote setup. (show figure 3.16)[5]

Figure 3.16: Batteries/Voltage source

3.4.10 Capacitors and Other Electronic Components

Capacitors and other electronic components play a supporting role in the stability and functionality of the smart inventory management system. Their primary functions include:

- **Capacitors:**
  - Stabilizing voltage and filtering noise in the power supply to ensure smooth operation of sensitive components like the microcontroller and sensors.
  - Providing temporary energy storage to smooth out power fluctuations, reducing the risk of system malfunctions.
- **Resistors:**
  - Controlling current flow to prevent damage to components by ensuring the correct voltage levels.
  - Protecting sensitive parts of the circuit by limiting excessive current.
- **Diodes:**
  - Preventing reverse current flow, which could damage components or cause incorrect behavior.
  - Used for rectification in circuits involving AC to DC conversion, ensuring proper power direction.
- **Transistors:**
  - Acting as switches to control power flow to different components like sensors, LEDs, or the buzzer.
  - Amplifying signals when necessary, especially in communication or sensor circuits.

These components ensure the proper functioning of the system, protect the hardware, and contribute to the overall reliability and longevity of the smart inventory management system. (show figure 3.17)[5]

Figure 3.17: Capacitors and Other Electronic Components

3.4.11 Breadboard

The breadboard is used as a temporary and flexible platform for assembling and testing the electronic components in the smart inventory management system. Its primary functions include:

- **Prototyping:**

Allowing quick and easy connections between components like the Atmega 128, RFID sensor, IR sensor, LEDs, and other parts without the need for soldering.

- **Testing:**

Enabling developers to test circuit designs and troubleshoot the system before creating a more permanent setup.

- **Flexibility:**

Providing the ability to modify the circuit layout easily, which is especially useful during the development and debugging phases.

The breadboard is a crucial tool for prototyping, testing, and refining the circuit before transitioning to a final design for the smart inventory management system. (show figure 3.18)[5]

Figure 3.18: Breadboard

3.4.12 USB ASP

The USB ASP is a USB-based programmer used for uploading code to microcontrollers, particularly for the Atmega 128 in your system. Its primary functions include:

- **Programming:**

Allowing the Atmega 128 microcontroller to be programmed with the necessary firmware or software via the ISP (In-System Programming) interface.

- **Debugging:**

Enabling direct access to the microcontroller for debugging and troubleshooting during development.

- **Flashing:**

Facilitating the uploading of new code or updates to the microcontroller's flash memory as needed.

The USB ASP is essential for developing, testing, and updating the microcontroller's program, ensuring the correct functionality of the entire system. (show figure 3.19)[5]

Figure 3.19: USB ASP

3.4.13 Jumper Wires

Jumper wires are used to make temporary electrical connections between components on the breadboard or between the breadboard and other parts of the system. Their primary functions include:

- **Wiring Connections:**

Establishing quick and flexible connections between components like the Atmega 128, sensors, and modules.

- **Prototyping:**

Facilitating easy reconfiguration of circuits during testing and debugging without the need for soldering.

- **Troubleshooting:**

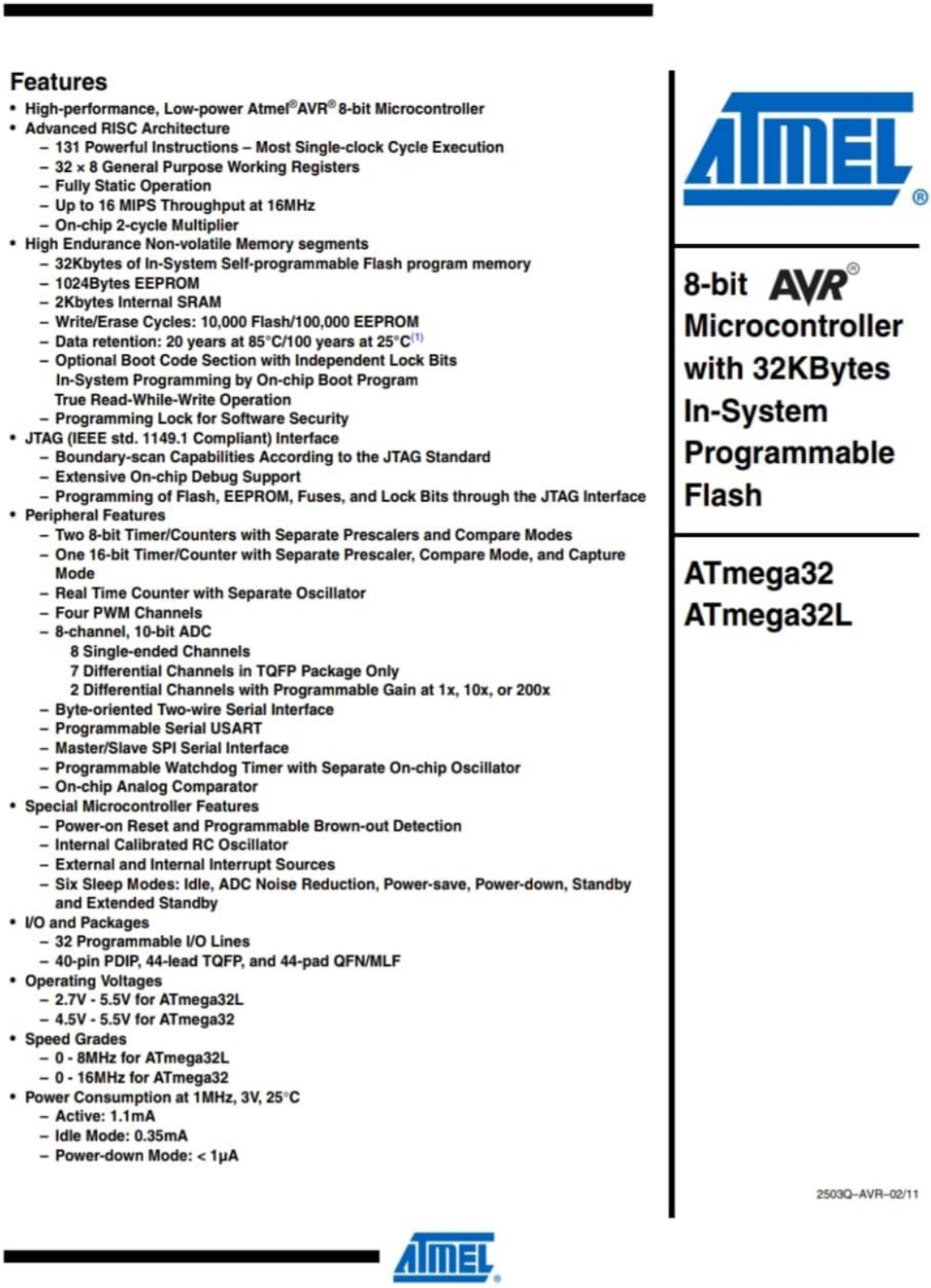
Allowing developers to change or swap connections quickly to identify issues or test different configurations.

Jumper wires are indispensable for building and modifying circuits, especially during the prototyping phase, enabling easy adjustments and efficient testing. (show figure 3.20)[5]

Figure 3.20: Jumper Wires

3.4.14 Datasheet

- The ATmega32 datasheet provides detailed technical specifications for the microcontroller used in embedded systems. It describes the microcontroller's core features, including a 32KB flash memory for program storage, 2KB SRAM, and 1KB EEPROM. The chip operates at clock frequencies up to 16 MHz and supports 32 general-purpose I/O pins. It includes multiple peripherals such as ADC (8-channel 10-bit), USART, SPI, I2C (TWI), timers, and PWM modules. The datasheet also outlines the power consumption, pin configuration, and electrical characteristics, helping developers understand how to interface it with sensors, actuators, and communication modules. It is a key reference for designing and programming systems like smart inventory management, where real-time control and reliable performance are essential. (show figure 3.21)[6]



Features

- High-performance, Low-power Atmel® AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 × 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 32Kbytes of In-System Self-programmable Flash program memory
  - 1024Bytes EEPROM
  - 2Kbytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7V - 5.5V for ATmega32L
  - 4.5V - 5.5V for ATmega32
- Speed Grades
  - 0 - 8MHz for ATmega32L
  - 0 - 16MHz for ATmega32
- Power Consumption at 1MHz, 3V, 25°C
  - Active: 1.1mA
  - Idle Mode: 0.35mA
  - Power-down Mode: < 1µA

8-bit AVR® Microcontroller with 32KBytes In-System Programmable Flash

ATmega32
ATmega32L

2503Q–AVR–02/11

Figure 3.21: Datasheet

All diagrams are uploaded to GitHub in good resolution

3.5 User Experience (UX)

UX, or User Experience, refers to a user's overall experience when interacting with a product, system, or service. It encompasses all aspects of the user's interaction, including their perceptions of usability, ease of use, and efficiency. A good UX aims to be intuitive, enjoyable, and meet the user's needs.[7]

3.5.1 Problem statement

This is a **Problem Statement** for a smart inventory management system project. The text describes the current challenges in inventory management in large warehouses, such as human errors and time loss due to reliance on manual methods. The goal of the project is to develop a system based on modern technology (such as barcodes, RFID, and sensors) to automate the process of inventory tracking and updating data in real time. The system will operate 24/7 and will provide accurate and immediate information, which will reduce errors, improve operational efficiency, and ensure that orders are completed quickly and accurately. (show figure 3.22)[7]

Figure 3.22: Problem statement

All diagrams are uploaded to GitHub in good resolution

3.5.2 Persona

This is a **User Persona** for a warehouse manager facing issues with inventory accuracy due to human errors and delays.The image outlines his goals, frustrations, motivations, and daily activities, helping in designing a system that meets his needs. (show figure 3.23)[7]

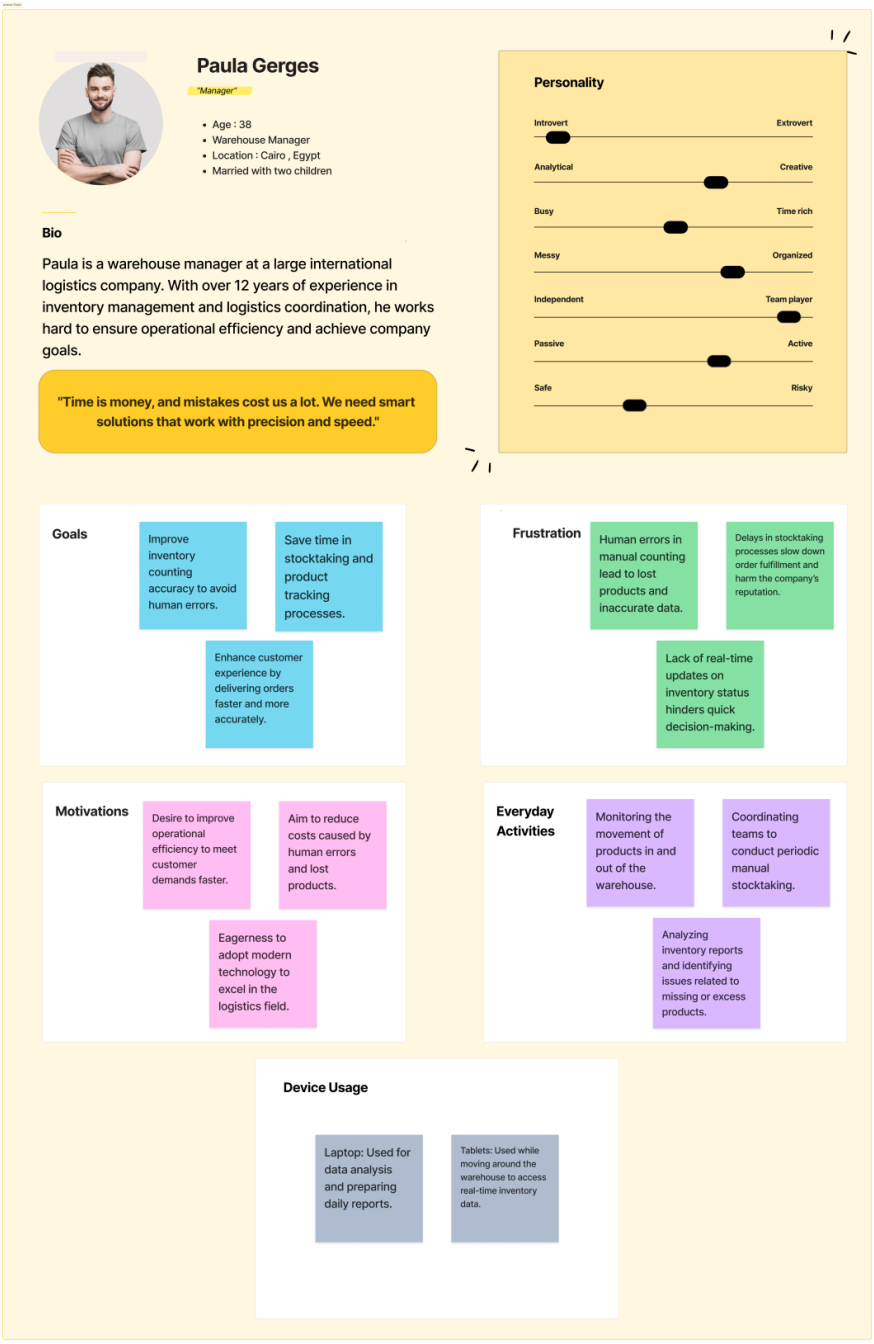


Figure 3.23: Persona

All diagrams are uploaded to GitHub in good resolution

3.5.3 Empathy map

This is an **Empathy Map** for a warehouse manager, outlining his thoughts, feelings, actions, and concerns regarding inventory management. It highlights his frustrations with manual stocktaking, his need for real-time updates, and his interest in automated solutions to reduce errors and improve efficiency. (show figure 3.24)[7]

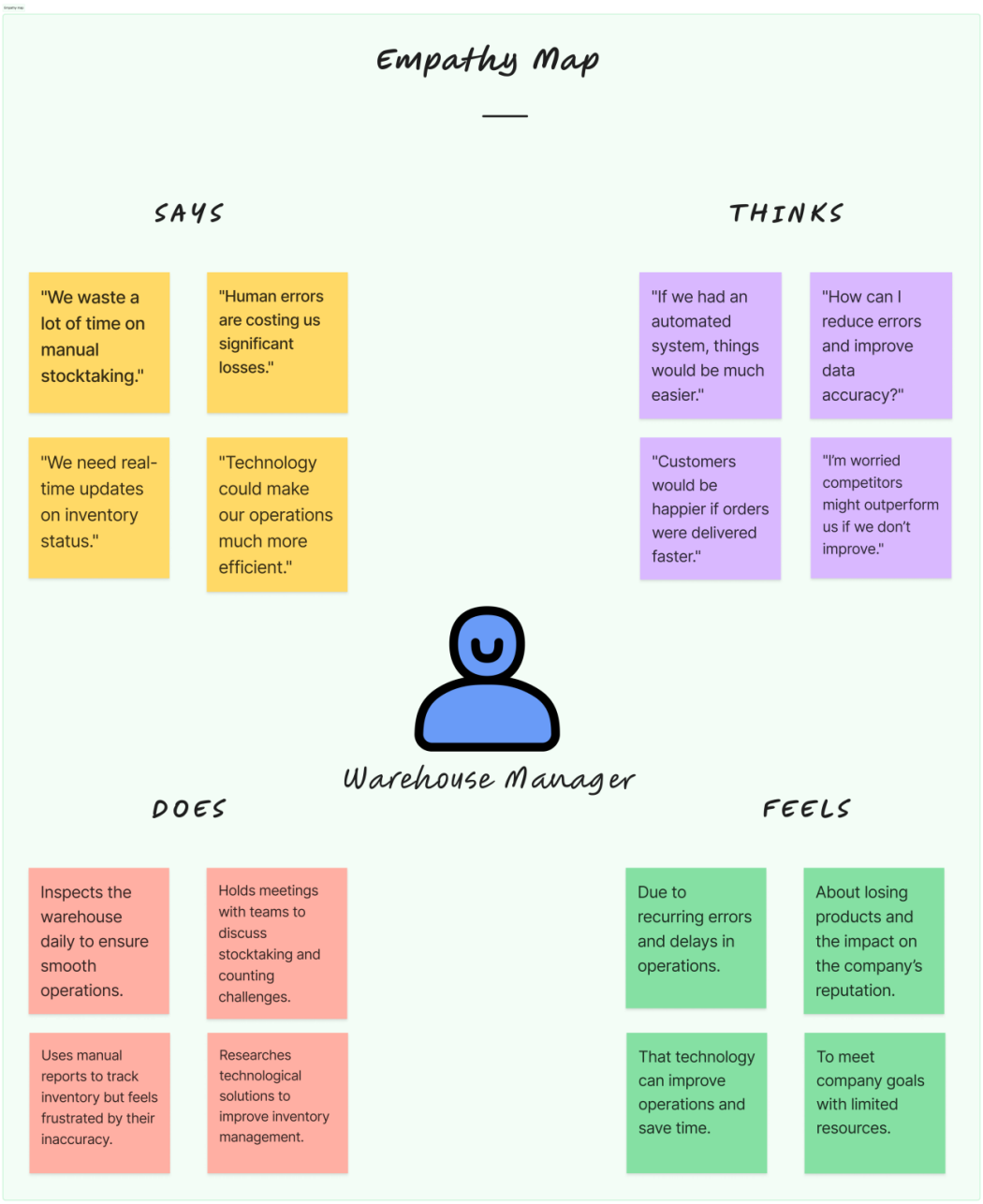


Figure 3.24: Empathy map

All diagrams are uploaded to GitHub in good resolution

3.5.4 User journey map

This is an **User journey map** of a warehouse manager in inventory management, from planning to searching for solutions. He faces challenges such as manual errors and delays, leading to frustration and stress. During execution, he monitors teams and records errors but feels exhausted. When analyzing data, he struggles with accuracy issues, causing disappointment. In decision-making, he feels pressured but remains optimistic about improvements. Finally, he searches for technological solutions to streamline operations and enhance accuracy, bringing him hope and excitement. (show figure 3.25)[7]

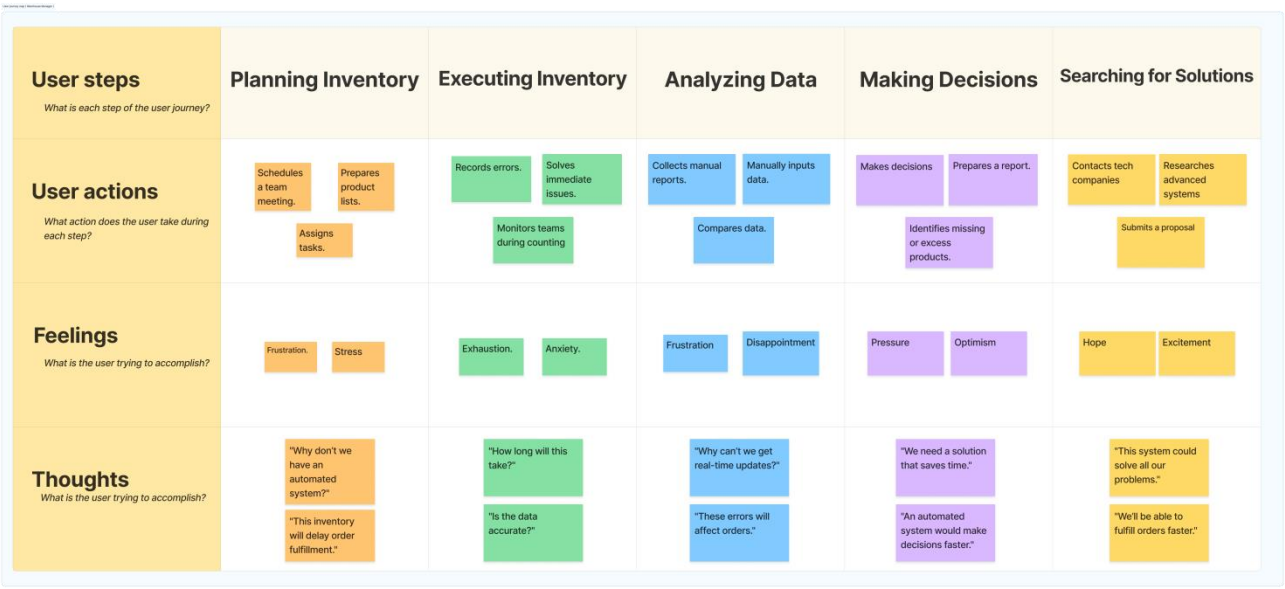


Figure 3.25: User journey map

All diagrams are uploaded to GitHub in good resolution

3.5.5 Information Architecture (IA)

This is an **information architecture** of a warehouse management system. It starts from the **Home Page**, leading to various modules such as **Login/Register**, **Dashboard**, **Inventory Items**, **Products**, **Categories/Shelves/Warehouses**, **User Management**, and **Faulty Devices**. Each section contains specific functionalities—for example, the **Dashboard** provides an overview of inventory, users, and assets, while the **Products** and **Inventory Items** sections allow for managing stock details. The system also includes user roles and permissions under **User Management**, ensuring controlled access. Lastly, **Faulty Devices** tracking helps maintain inventory efficiency by flagging defective items. (show figure 3.26)[7]

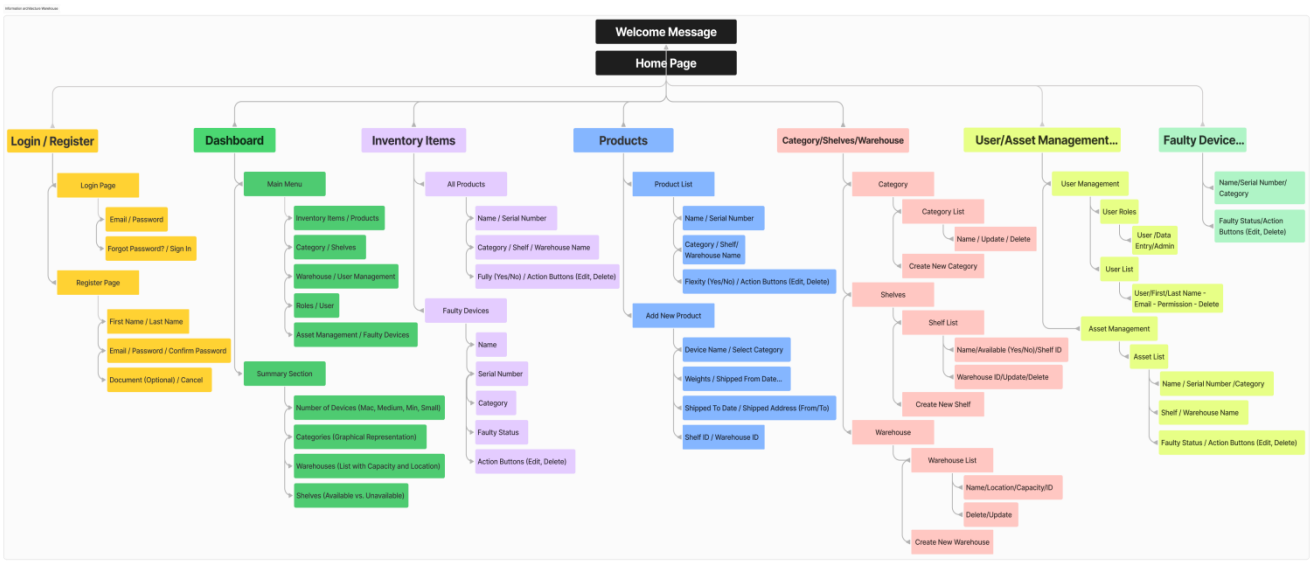


Figure 3.26: Information Architecture (IA)

All diagrams are uploaded to GitHub in good resolution

3.5.6 User Flow

The is an **user flow** of a warehouse management **system**. It starts with users opening the website and logging in. If they are new, they must register; if they forget their password, they can reset it. After successful login, they access the **Dashboard**, which provides options for managing **inventory items, products, categories, shelves, and warehouses**. Users can also handle **user management, and faulty devices**. Each section allows actions like **adding, updating, and viewing details**. The flow ensures smooth navigation from login to inventory and user management, leading to successful transaction completion. (show figure 3.27)[7]

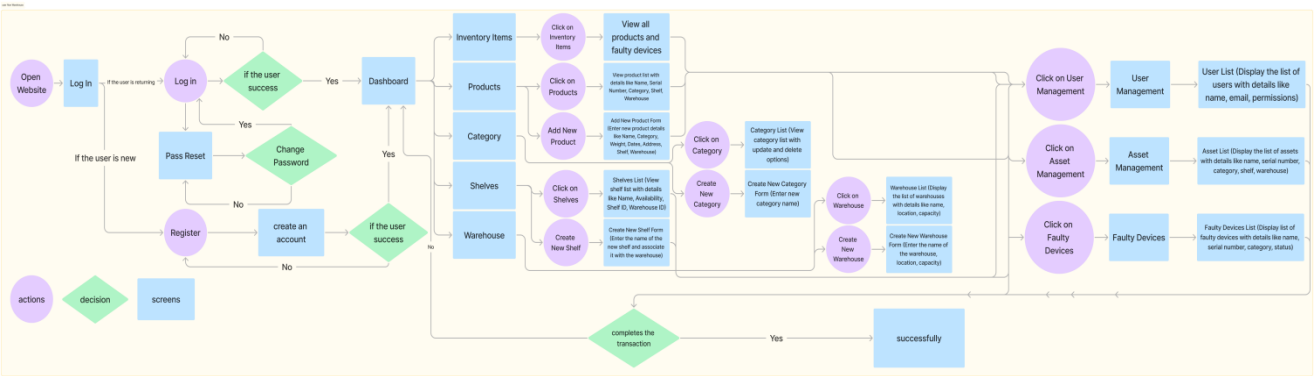


Figure 3.27: User Flow

All diagrams are uploaded to GitHub in good resolution

3.5.7 Wireframes

Wireframes are visual representations of a website or application's structure and layout, essentially blueprints for digital products. They focus on functionality, content placement, and user flow, omitting visual details like colors and fonts to emphasize the core structure. Think of them as the architectural plans for a building, outlining the placement of rooms, doors, and windows before any interior design is considered.[8]

3.5.7.1 Low-fidelity wireframes

Low-fidelity (Lo-Fi) wireframes are simple, sketch-like representations of a website or application layout. They focus on structure, functionality, and user flow rather than design details. Typically, they use boxes, lines, and placeholders for text and images.[8]

How They Are Created:

1. **Hand-drawn sketches** – Quickly drafted on paper or whiteboards.
2. **Basic grayscale design** – Avoids colors, detailed typography, and images, focusing only on layout and navigation.

Lo-Fi wireframes are ideal for brainstorming, gathering initial feedback, and making quick changes before committing to a high-fidelity design. (show figure 3.28,3.29,3.30)[8]

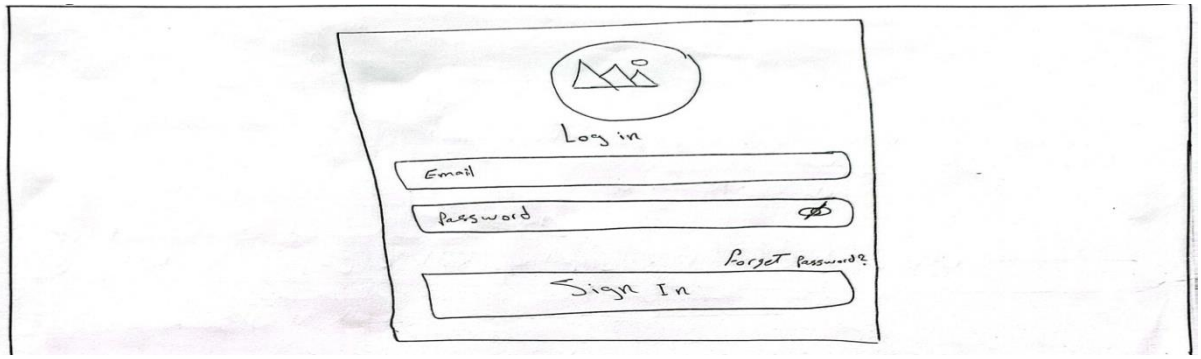


Figure 3.28: Low-fidelity wireframes – Log in

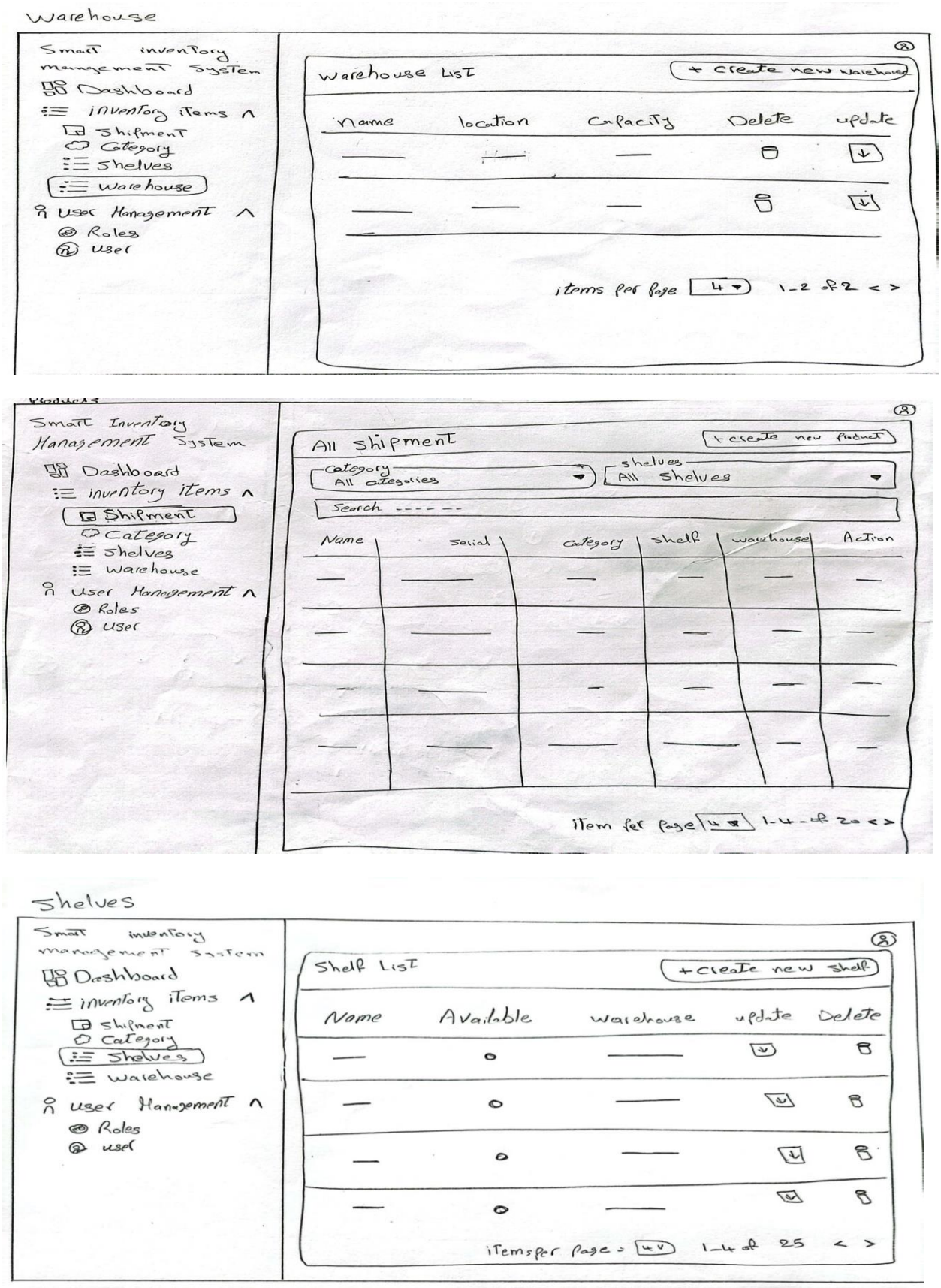


Figure 3.29: Low-fidelity wireframes – warehouse, shipment, shelves

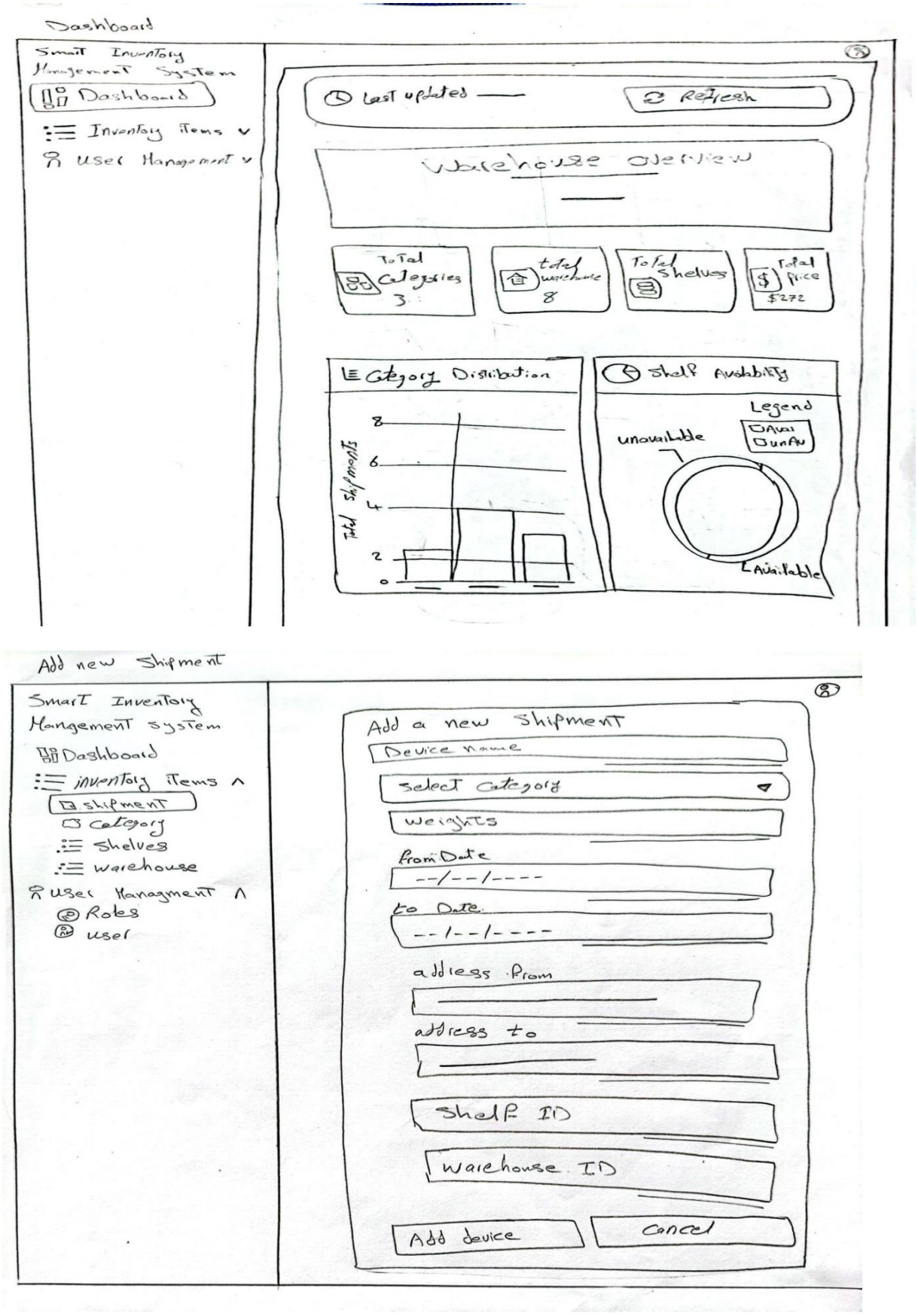


Figure 3.30: Low-fidelity wireframes – dashboard, add new shipment

3.5.7.2 High-fidelity wireframes

High-fidelity (Hi-Fi) wireframes are detailed and visually refined, closely resembling the final product. They include real content, proper typography, and sometimes interactive elements to simulate user experience.[8]

How They Are Created:

1. **Using design software** – Tools like Figma, Adobe XD, or Sketch provide advanced design capabilities.
2. **Incorporating real UI elements** – Includes actual buttons, menus, images, and text instead of placeholders.

Hi-Fi wireframes are essential for stakeholder presentations, usability testing, and providing a clear development guide for designers and developers. (show figure 3.31,3.32,3.33)[8]

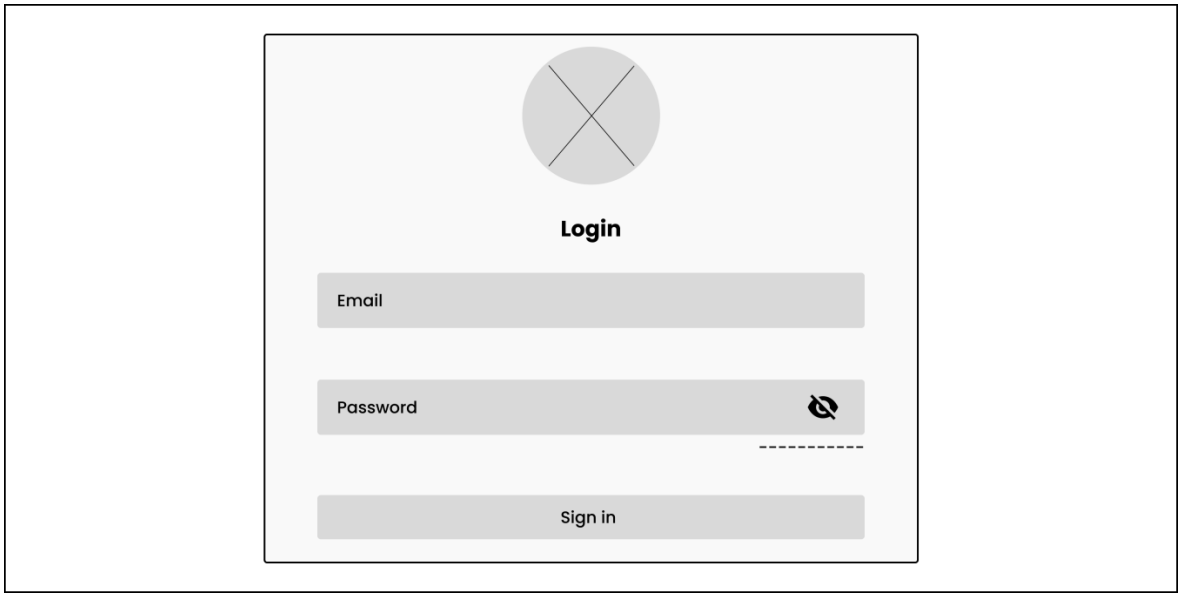


Figure 3.31: High-fidelity wireframes – Log in

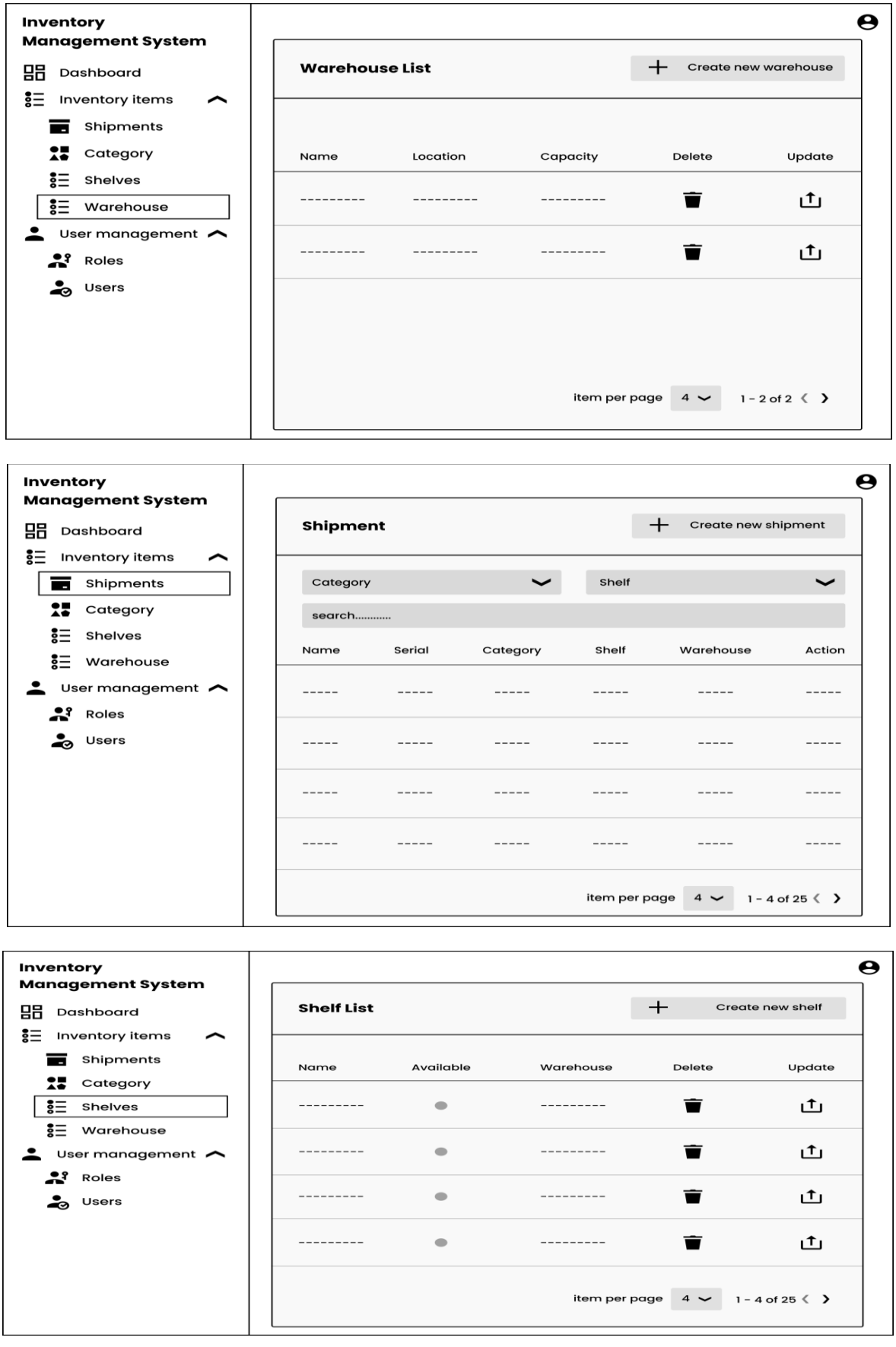
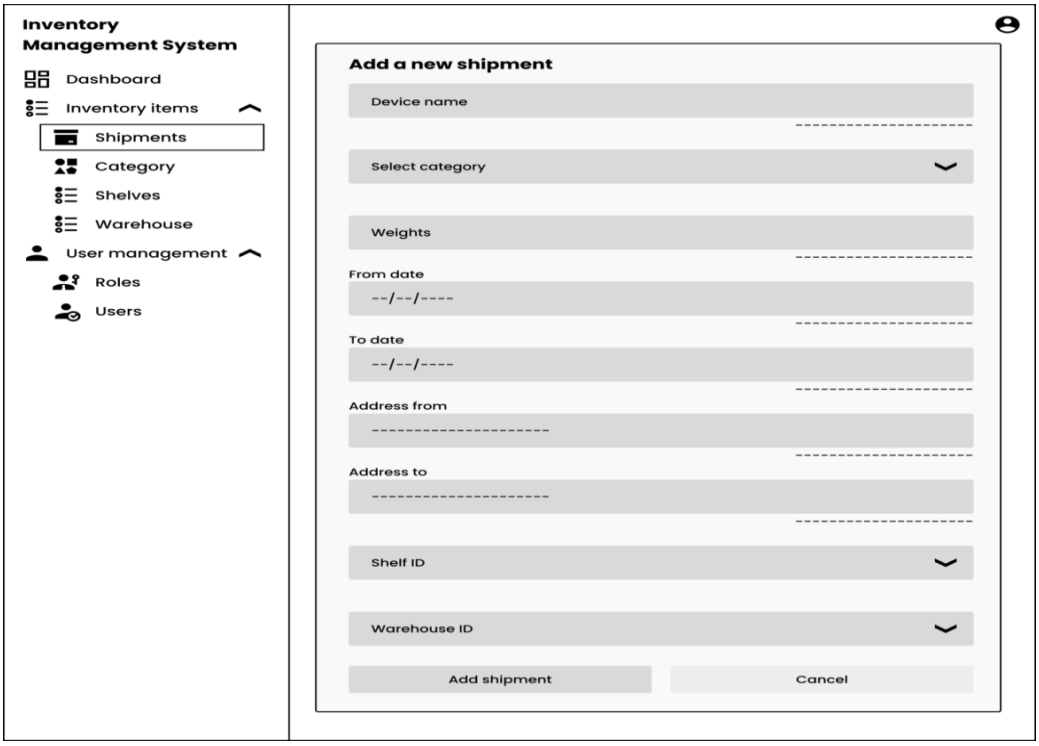


Figure 3.32: High-fidelity wireframes – warehouse, shipment, shelves



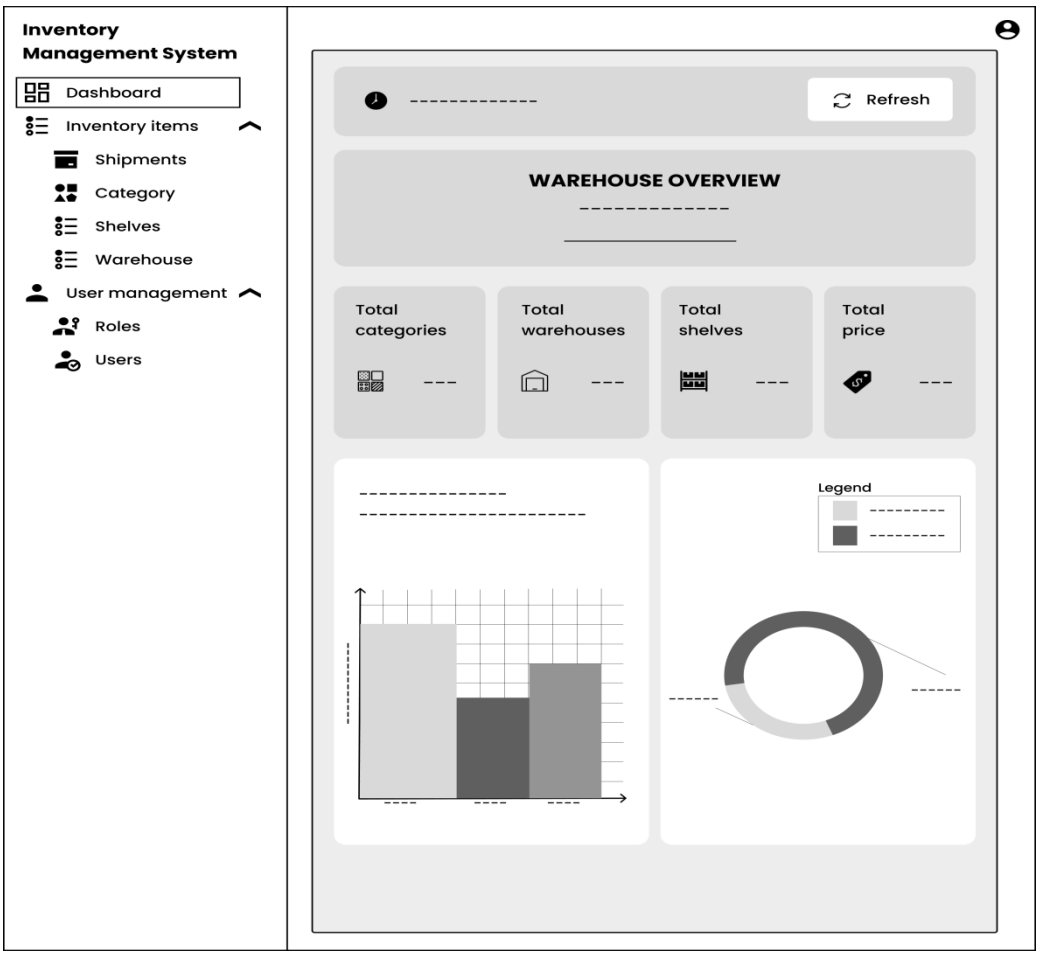


Figure 3.33: High-fidelity wireframes – dashboard, add new shipment

3.6 User Interface (UI)

- It is the visual part of an application that users interact with, such as buttons, menus, and text. The goal is to make the app easy to use and visually appealing.[9]

- **Steps to Design UI in Figma:**

  Create a New Project → Open Figma and start a new file.

  Add Screens (Frames) → Select the screen size (mobile, web, etc.).

  Insert Elements → Buttons, input fields, icons, images.

  Set Colors and Fonts → Use a consistent color scheme and clear fonts.

  Create Reusable Components → For easy modifications later.
  (show figure 3.34,3.35)[9]

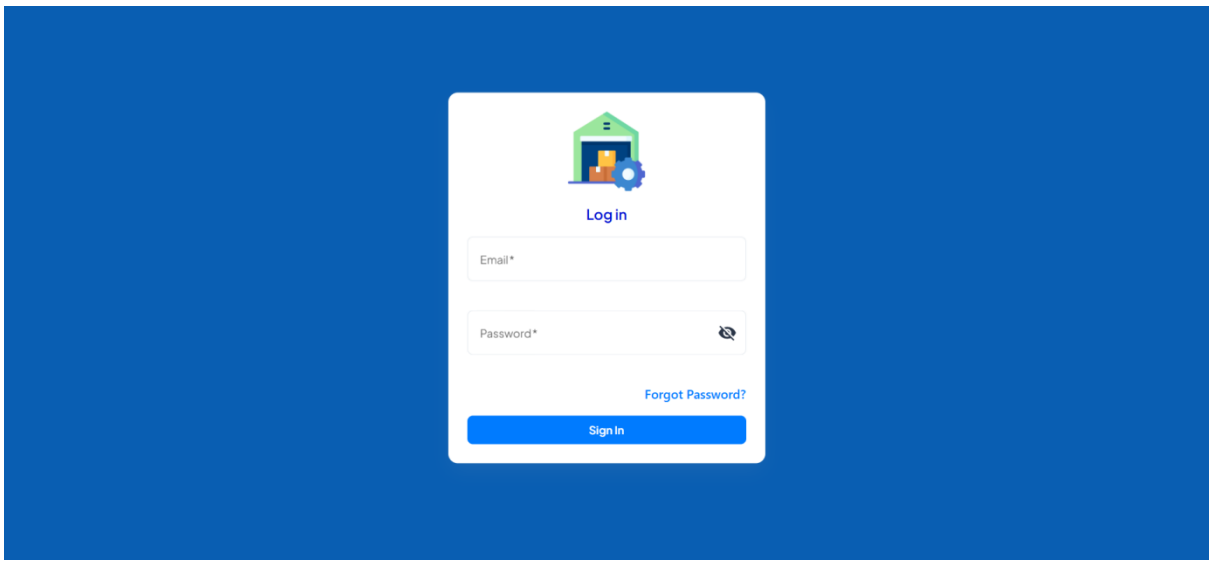


Figure 3.34: User Interface (UI) – log in

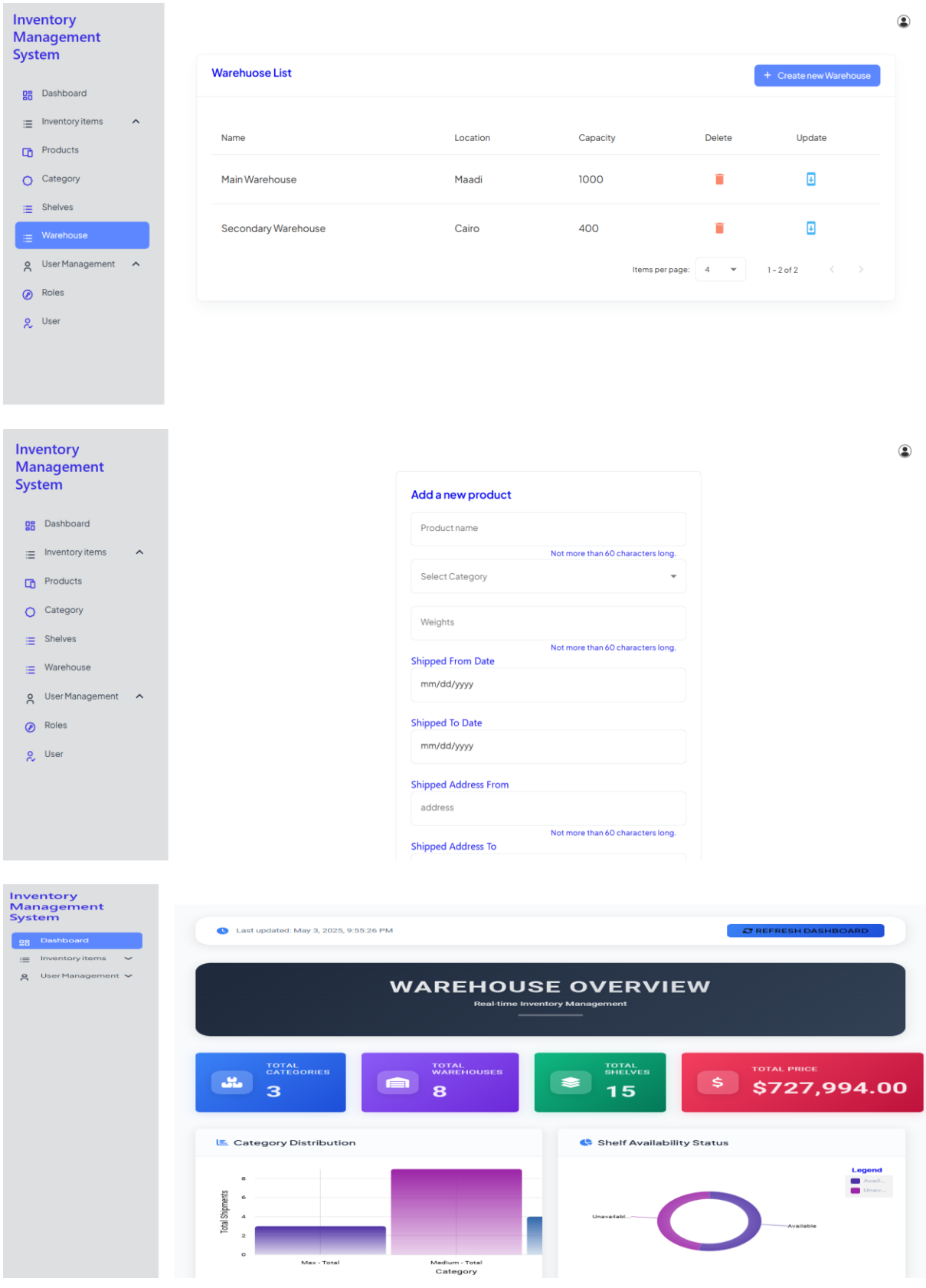


Figure 3.35: User Interface (UI) – warehouse, add new shipment, dashboard

Chapter 4
Implementation

4.1 Introduction (Frontend _Backend)

The Inventory Management System is designed to help businesses efficiently manage their shipments, categories, shelves, and warehouses. This system is built with:

- **Backend**: .NET Core API
- **Frontend**: Angular
- **Database**: SQL Server

The system provides robust features to track inventory, organize products, categorize them, and manage warehouse storage efficiently.

System Architecture

1. **Backend**
    - **Framework**: .NET Core 8 API
    - **Pattern**: Clean Architecture with Domain, Application, Infrastructure, and API layers.
    - **Authentication**: JWT-based authentication.
    - **Database Interaction**: Entity Framework Core.
2. **Frontend**
    - **Framework**: Angular
    - **Design System**: Angular Material for UI components.
3. **Database**
    - Relational database (SQL Server).
    - Schema includes tables for shipments, Categories, Shells, Warehouses, AspNetUsers, AspNetUserLogins, AspNetUserRoles, AspNetUserClaims, AspNetUserTokens, AspNetRoles, AspNetRoleClaims, _EFMigrationsHistory.

4.2 Software Implementation

Software implementation is the process of putting a software system into operation within an organization. It encompasses more than just installing the software; it involves integrating it into existing workflows, migrating data, training users, and ensuring it aligns with the organization's needs. Effective implementation is crucial for realizing the benefits of new software, such as increased efficiency, improved data management, and better compliance.

4.2.1 Front End

The frontend of the Inventory Management System is built using **Angular**, a popular framework for building dynamic and responsive web applications. The frontend is responsible for providing a user-friendly interface for managing shipments, categories, shelves, warehouses, and user roles.

Frontend Architecture

- **Framework**: Angular
- **UI Components**: Angular Material for a consistent and modern design.
- **API Integration**: The frontend communicates with the backend via RESTful API endpoints to fetch and manipulate data.

4.2.2 Back End

The backend of the Inventory Management System is built using **.NET Core 8 API**, which provides a robust and scalable foundation for handling business logic, data processing, and security. The backend is responsible for managing data, processing requests from the frontend, and ensuring secure access to the system.

Backend Architecture

- **Framework**: .NET Core 8 API
- **Architecture Pattern**: Clean Architecture with Domain, Application, Infrastructure, and API layers.
- **Authentication**: JWT-based authentication for secure API access.
- **Database Interaction**: Entity Framework Core for interacting with the SQL Server database.

Database Schema

The database is a relational database (SQL Server) with the following tables:

- shipments
- Categories
- Shelves
- Warehouses
- AspNetUsers
- AspNetUserLogins
- AspNetUserRoles
- AspNetUserClaims
- AspNetUserTokens

- AspNetRoles
- AspNetRoleClaims
- _EFMigrationsHistory

API Endpoints

1. **Shipment Endpoints**
   - GET /api/ shipments - Retrieve all shipments.
   - GET /api/ shipments /{id} - Retrieve a specific shipments by ID.
   - GET /api/ shipments /count - Count of shipments.
   - POST /api/ shipments - Create a new shipments.
   - PUT /api/ shipments /{id} - Update a shipments.
   - DELETE /api/ shipments /{id} - Delete a shipments.

2. **Category Endpoints**
   - GET /api/categories - Retrieve all categories.
   - GET /api/categories/{id} - Retrieve a specific category by ID.
   - POST /api/categories - Create a new category.
   - PUT /api/categories/{id} - Update a category.
   - DELETE /api/categories/{id} - Delete a category.

3. **Shelf Endpoints**
   - GET /api/shelfs - Retrieve all shelves.
   - GET /api/shelfs/{id} - Retrieve a specific shelf by ID.
   - GET /api/shelfs/count - Count of shells.
   - POST /api/shelfs - Create a new shelf.
   - PUT /api/shelfs/{id} - Update a shelf.
   - DELETE /api/shelfs/{id} - Delete a shelf.

4. **Warehouse Endpoints**
    - GET /api/warehouses - Retrieve all warehouses.
    - GET /api/warehouses/{id} - Retrieve a specific warehouse by ID.
    - POST /api/warehouses - Create a new warehouse.
    - PUT /api/warehouses/{id} - Update a warehouse.
    - DELETE /api/warehouses/{id} - Delete a warehouse.

5. **Roles Endpoints**
    - GET /api/roles - Retrieve all roles.
    - GET /api/roles/{roleId} - Retrieve a specific role by ID.
    - POST /api/roles - Create a new role.
    - PUT /api/roles/{roleId} - Update a role.
    - DELETE /api/roles/{id} - Delete a role.

6. **Accounts Endpoints**
    - GET /api/accounts/EmailConfirmation
    - GET /api/accounts
    - GET /api/accounts/{id}
    - PUT /api/accounts/{id}
    - DELETE /api/accounts/{id}
    - POST /api/accounts/Registration
    - POST /api/accounts/Login
    - POST /api/accounts/ForgotPassword
    - POST /api/accounts/ResetPassword

7. **Authentication and Authorization**
    - **JWT Tokens**: Used for secure API access.
    - **Roles**: Admin, Data Entry, and User roles are assigned to control access to different parts of the system.

4.3 Extract-Parameter Endpoint

This endpoint (extract-parameter) processes a structured string of space-separated values to extract details for assigning or unassigning a product to a shelf in a warehouse. Here's a breakdown in 10 key points:

1. **Receives a Request Body**
    - The request must contain Data (a space-separated string) and a Product object.
2. **Validates the Input**
    - Checks if Data is null or empty.
    - Ensures Data has at least 18 space-separated parts.
3. **Extracts Key Parameters from Data**
    - First Value -> inOrOut (1 = check-in, 0 = check-out).
    - Second Value -> warehouse (a single-character identifier).
    - Next 16 Values -> Represent a GUID in hexadecimal format.
4. **Converts Serial Number to GUID**
    - Joins the 16 hex parts into a 32-character string.
    - Converts it into a byte array.
    - Reorders bytes correctly for GUID formatting.
    - Constructs the GUID from the byte array.
5. **Retrieves Product from Database**
    - Queries the product using the extracted serial number.
6. **Calls AssignShelfToProduct**
    - Passes extracted parameters to another endpoint for processing.

- Inside AssignShelfToProduct.
- This method assigns or unassigns a shelf for the product.

7. **Validates Input Again**
   - Checks if the product is already assigned (In_Out == true).
   - Ensures the serial number is valid.
   - Retrieves the matching warehouse based on the given character.
8. **Finds Available Shelf in Warehouse**
   - Gets all shelves and finds an available one in the selected warehouse.
9. **Assigns or Unassigns the Shelf**
   - If inOrOut = true (Check-In):
   - Assigns the first available shelf.
   - Updates the product and shelf status.
   - Saves changes to the database.
   - Returns 'Product assigned to a shelf.'
   - If inOrOut = false (Check-Out):
   - Frees up the currently assigned shelf.
   - Updates the product to mark it as not assigned.
   - Saves changes.
   - Returns 'Product checked out from shelf.'
10. **Error Handling**
    - Logs errors and returns appropriate responses (400, 404, or 500).

This endpoint ensures that products are properly assigned to and removed from warehouse shelves while enforcing strict validation. (show figure 4.1)

```
public class ExtractParameterRequest
{
    public string Data { get; set; }
    public ProductForUpdateDto Product { get; set; } = new ProductForUpdateDto();
}

[HttpPost("extract-parameter")]
public async Task<IActionResult> ExtractParameter([FromBody] ExtractParameterRequest request)
{

    if (request == null || string.IsNullOrEmpty(request.Data))
    {
        return BadRequest("Request body or data is required.");
    }

    string data = request.Data;

    // Split the data by spaces
    string[] parts = data.Split(' ', StringSplitOptions.RemoveEmptyEntries);

    if (parts.Length < 18) // Must be at least 19 parts: 1 + 1 + 16 (GUID) + 1
    {
        return BadRequest("Data format is incorrect.");
    }

    // Parse inOrOut (first value)
    bool inOrOut;
    if (parts[0] == "1")
        inOrOut = true;
    else if (parts[0] == "0")
        inOrOut = false;
    else
        return BadRequest("Invalid inOrOut parameter. Expected '1' or '0'.");

    // Parse warehouse (second value, should be a single character)
    if (parts[1].Length != 1)
    {
        return BadRequest("Invalid warehouse parameter. Expected a single character.");
    }
    char warehouse = parts[1][0];

    // Extract the 16-byte serial number (from index 2 to index 17)
    string serialHexString = string.Join("", parts.Skip(2).Take(16)); // Remove spaces and join

    if (serialHexString.Length != 32) // A valid GUID must have 32 hex digits
    {
        return BadRequest("Invalid serial number length. Expected 16 hexadecimal values.");
    }

    // Convert hex string to byte array
    byte[] serialBytes = new byte[16];
    for (int i = 0; i < 16; i++)
    {
        if (!byte.TryParse(parts[i + 2], System.Globalization.NumberStyles.HexNumber, null, out
serialBytes[i]))
        {
            return BadRequest($"Invalid hex value at position {i + 2}: {parts[i + 2]}");
        }
    }

    // Convert byte array to GUID by ensuring correct endianess
    Guid serialNumber = new Guid(
        new byte[]
        {
            serialBytes[3], serialBytes[2], serialBytes[1], serialBytes[0], // Data1 (4 bytes,
reversed)
            serialBytes[5], serialBytes[4], // Data2 (2 bytes, reversed)
            serialBytes[7], serialBytes[6], // Data3 (2 bytes, reversed)
            serialBytes[8], serialBytes[9], serialBytes[10], serialBytes[11], // Data4 (big-
endian, unchanged)
            serialBytes[12], serialBytes[13], serialBytes[14], serialBytes[15]  // Data5 (big-
endian, unchanged)
        }
    );
    var productEntity = await _repository.Product.GetProductBySerialNumberAsync(serialNumber,
trackChanges: false);
    try
    {
        return await AssignShelfToProduct(inOrOut, warehouse, serialNumber,
_mapper.Map(request.Product, productEntity));
    }
    catch (Exception ex)
    {
        _logger.LogError($"Error in ExtractParam calling AssignShelfToProduct: {ex.Message}");
        return StatusCode(500, "Internal server error.");
    }
}
```

Figure 4.1: Extract-Parameter Endpoint

All diagrams are uploaded to GitHub in good resolution

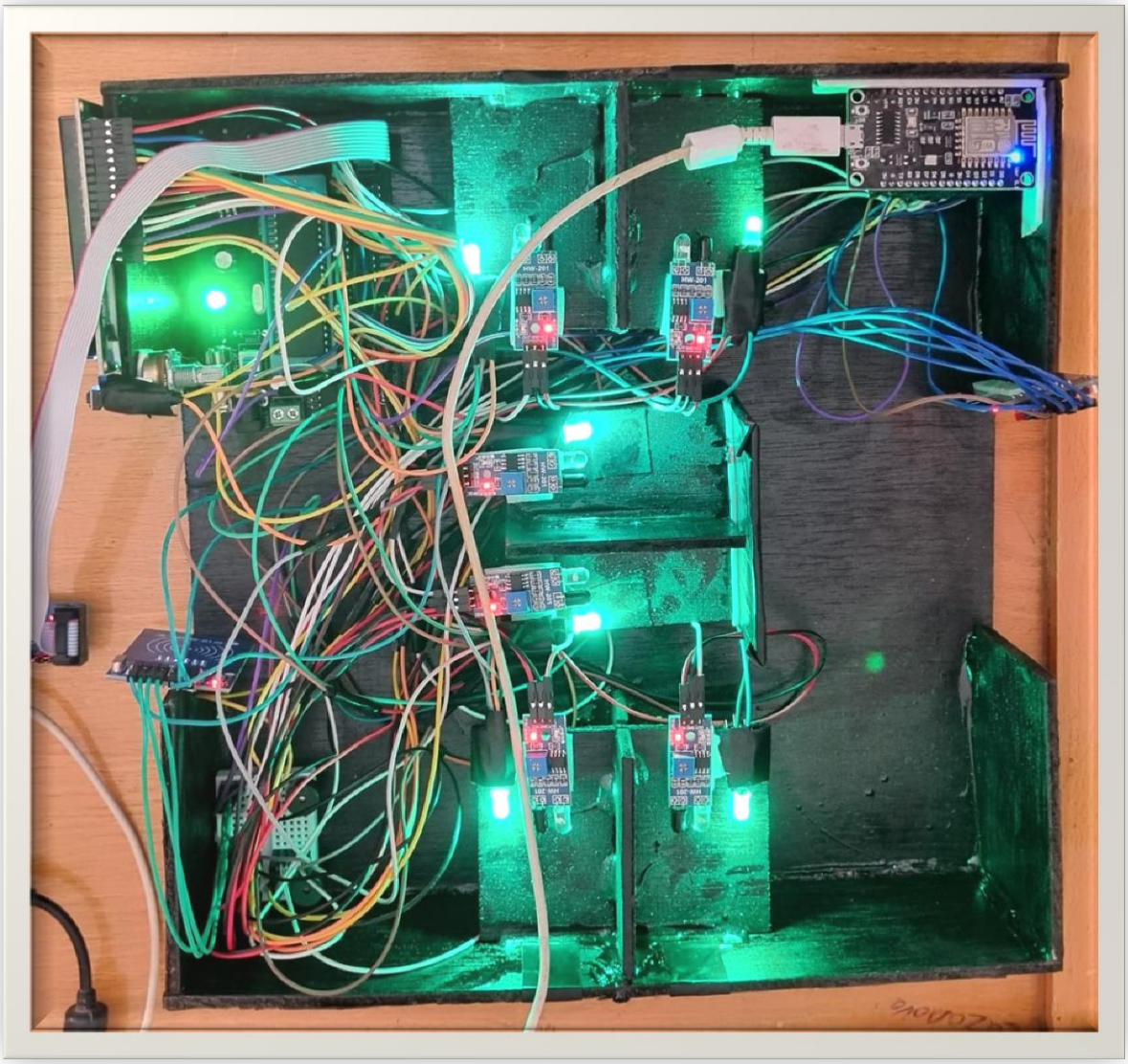


Figure 4.2: Visualizing the project

Conclusion

Key Achievements:

- Reduction of Human Errors: Automation has eliminated errors caused by manual counting, leading to more accurate inventory data.
- Time Savings: Automated processes have significantly reduced the time required for inventory management, increasing productivity.
- Real-Time Updates: The system provides accurate and continuous information about inventory status, enabling managers to make faster and more effective decisions.
- Improved Customer Satisfaction: By ensuring inventory accuracy and faster order fulfillment, the overall customer experience has been enhanced.

Impact on Business:

- Reduced Operational Costs: By minimizing errors and saving time, costs have been significantly reduced.
- Increased Operational Efficiency: The system has improved workflow within warehouses, leading to higher productivity.
- Enhanced Competitiveness: Companies using this system can now meet customer demands faster and more accurately, strengthening their market position.

Value Added:

This system does not just offer technical improvements; it redefines how inventory is managed in large warehouses. Through automation and real-time updates, warehouses can now operate more efficiently, reduce waste, and ensure customer satisfaction. This project represents a major step toward a smarter and more efficient future in supply chain and inventory management.

Key Features of the System:

1. **Automatic Shipment Tracking:**
    - Use of barcodes and “RFID” to track shipments as they enter or exit the warehouse.
    - Automatic updating of data in the central system.
2. **Real-Time Shelf Monitoring:**
    - Sensors on shelves to determine whether they are full or empty.
    - Sending real-time data to the system to identify available storage spaces.
3. **Interactive Web Interface:**
    - A dedicated website to display inventory data clearly and user-friendly for warehouse staff.
    - Access to data anytime, anywhere.
4. **Comprehensive Inventory Management:**
    - Add, update, and delete products, categories, shelves, and warehouses.
    - Track and manage faulty products effectively.
5. **24/7 Operation:**
    - The system operates around the clock, allowing inventory tracking at any time.
6. **Real-Time Reporting:**
    - Generate instant reports on inventory status, available products, and faulty devices.

Future Work

i. Intelligent Automated Gates and Tracking Automated gates equipped with barcode scanners and RFID readers enhanced with computer vision AI for visual verification AI-powered object recognition to identify products even without clear barcode visibility Machine learning algorithms that learn from scanning patterns to improve accuracy over time Anomaly detection AI to flag suspicious activities or unusual inventory movements.

ii. Smart Sensor Networks with AI Analytics Sensors installed on shelves with AI-powered space optimization algorithms Predictive analytics to forecast when shelves will be full or empty Machine learning models that optimize storage allocation based on product characteristics and demand patterns AI-driven maintenance prediction for sensor networks to prevent failures.

iii. Intelligent Data Processing and Analytics Real-time AI processing of inventory data with pattern recognition Predictive inventory management using machine learning to forecast demand AI-powered supply chain optimization recommendations Intelligent alert system that prioritizes notifications based on business impact.

iv. Computer Vision Integration AI-powered visual inspection for product quality control Automated damage detection using image recognition algorithms Smart product categorization through visual analysis Facial recognition for personnel access control and activity tracking.

References

[1] https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-9.0

[2] https://dotnettutorials.net/lesson/automapper-in-c-sharp/

[3] https://learn.microsoft.com/en-us/aspnet/core/security/authorization/roles?view=aspnetcore-9.0

[4] https://handsontec.com/dataspecs/module/esp8266-V13.pdf

[5] https://randomnerdtutorials.com/esp8266-nodemcu-http-get-post-arduino/

[6] https://www.alldatasheet.com/datasheet-pdf/view/255854/ATMEL/ATMEGA32.html

[7] Krug, S. (2014). Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability (3rd ed.). New Riders

[8] https://dribbble.com/

[9] https://www.smashingmagazine.com/category/ux/

Appendix: GitHub Link

- https://github.com/csgraduationproject602/Smart-Inventory-Management-System-Graduation-Project-602-CS

ملخص المشروع باللغة العربية

في المستودعات الكبيرة، يتم عد المخزون يدويًا بواسطة البشر، مما يتسبب في العديد من المشكلات مثل الأخطاء البشرية، وفقدان الوقت، وعدم الدقة في معرفة الكميات الفعلية المتاحة أو المفقودة من الشحنات. تتفاقم هذه المشكلات في المستودعات التي تعمل على مدار الساعة وتتعامل مع أحجام كبيرة من المخزون، مثل تلك المستخدمة من قبل شركات مثل أمازون وأرامكس. المستفيدون الرئيسيون من هذا النظام هم مالكو ومديرو المستودعات الكبيرة الذين يواجهون تحديات في تتبع حركة الشحنات داخل وخارج المستودع. يستهدف النظام الشركات الكبيرة التي تعتمد على مستودعات ضخمة وتحتاج إلى الدقة والسرعة في عد وإدارة المخزون.

النظام الذي يتم تطويره هو حل شامل يهدف إلى القضاء على المشكلات الناتجة عن العد اليدوي. يعتمد النظام على أتمتة العملية باستخدام تقنيات حديثة مثل البوابات، وأجهزة الاستشعار، والباركود، وأجهزة الاستشعار بالأشعة تحت الحمراء أو بالموجات فوق الصوتية أو تقنية .RFID ستقوم هذه الأدوات بتتبع الشحنات تلقائيًا عند دخولها أو خروجها من المستودع وتحديث البيانات في النظام دون الحاجة إلى تدخل بشري. سيتم تنفيذ هذا النظام في أي مستودع كبير يتعامل مع كميات هائلة من المنتجات، بغض النظر عن نوع الشحنة أو حجم المستودع.

سيعمل النظام عند بوابات الدخول والخروج للمستودع، وكذلك داخل المستودع على الأرفف نفسها لتتبع الشحنات المتاحة وتحديد المساحات الفارغة. سيكون النظام جاهزًا للعمل على مدار الساعة، مما يسمح بتتبع أي منتج يدخل أو يغادر المستودع في أي وقت خلال اليوم. ستكون العملية بأكملها مؤتمتة وستحدث في الوقت الفعلي، مما يعني أن عمال المستودع سيحصلون على تحديثات مستمرة حول المخزون في أي لحظة.

العد اليدوي للمخزون غير فعال ويتسبب في العديد من المشكلات مثل الأخطاء البشرية، والتأخير، وعدم الدقة في تحديد الكميات الفعلية للمخزون. هذا يؤدي إلى عدم الكفاءة التشغيلية والتأخير في تنفيذ الطلبات أو حتى فقدان الشحنات. أتمتة عملية العد ستوفر الوقت، وتضمن الدقة، وتقلل من الحاجة إلى التدخل البشري، مما يحسن بشكل كبير الكفاءة التشغيلية العامة.

كيفية عمل النظام :

سيعتمد النظام على بوابات عند مدخل ومخرج المستودع، مزودة بأجهزة استشعار، وماسحات باركود، ستقرأ هذه الأجهزة الباركود الخاص بكل شحنة عند الدخول أو الخروج وترسل هذه .RFID وتقنية المعلومات إلى النظام المركزي.

بالإضافة إلى ذلك، ستكون الأرفف داخل المستودع مجهزة بأجهزة استشعار لقياس ما إذا كانت ممتلئة أو فارغة. ستقوم هذه المستشعرات بإرسال بيانات في الوقت الفعلي لتحديد المساحات التخزينية المتاحة.

عند مغادرة شحنة المستودع، سيتم تحديث الكميات في النظام، وستخبر المستشعرات النظام بأن الرف أصبح فارغًا. سيتم عرض جميع هذه البيانات على موقع مخصص لموظفي المستودع.

سيستخدم النظام بروتوكولات اتصال لربط الأنظمة المدمجة التي تتحكم في المستشعرات والبوابات بالنظام المركزي لعرض البيانات على الويب.

الفوائد المتوقعة :

تقليل الأخطاء البشرية : من خلال الأتمتة، سيتم تجنب الأخطاء الناتجة عن العد اليدوي.

توفير الوقت : العمليات المؤتمتة ستقلل من الوقت المطلوب لإدارة المخزون.

تحسين الكفاءة : تحديثات البيانات في الوقت الفعلي ستساعد في اتخاذ قرارات أسرع وأكثر دقة.

زيادة رضا العملاء : من خلال ضمان دقة المخزون وسرعة تنفيذ الطلبات.

.