

# **PROJECT**

## **Warehouse Management System**

Submitted By,  
Shubham Lathiya (SCA24801030058),  
Priyank Mangaroliya (SCA24801030064)

Guided By,  
Prof. Abhijita Puhan,  
Dr. Kamlesh Meshram

For partial fulfillment of the requirements  
For the Degree of Master of Computer Application  
School of Engineering and Technology,  
Pimpri Chinchwad University.  
November, 2024.

## Acknowledgment

We extend our heartfelt gratitude to all those who contributed to the successful completion of our project on the Warehouse Management System.

We sincerely thank Prof. Abhijita Puhan, whose guidance and encouragement have been invaluable throughout the project. Their expertise and constructive feedback helped us navigate challenges and improve our work.

We are grateful to school of engineering and technology for providing the necessary resources and a conducive environment for carrying out this project. The infrastructure and support from the institution played a significant role in achieving our objectives. Lastly, we acknowledge the efforts of our team members and the support of our peers and family, whose cooperation and motivation inspired us to excel in this endeavour.

## Abstract

This Warehouse Management System (WMS) project aims to enhance the efficiency, accuracy, and productivity of warehouse operations by addressing challenges such as inaccurate inventory records, late order processing, and labor-intensive tasks. Designed for a single warehouse, the WMS enables real-time inventory tracking, streamlined order fulfillment, and optimized space utilization.

The project employs an Agile methodology, ensuring iterative development and client feedback integration. The system utilizes a robust tech stack, including Flask for the backend, MongoDB for scalable data storage, and HTML, CSS, and JavaScript for a responsive user interface. Key features include user management, inventory tracking, automated order processing, efficient picking and packing, shipping, and detailed reporting with analytics.

Expected outcomes include reduced errors, minimized costs, and data-driven decision-making, resulting in improved operational efficiency, accurate order fulfillment, and enhanced customer satisfaction.

# Index

1) Introduction .....	01
1.1) Problem Statement .....	01
1.2) Objectives .....	01
1.3) Scope .....	01
1.4) Organization of the Report .....	02
2) Literature Survey/Review .....	03
2.1) Overview of existing work .....	03
2.2) Projects, or products reviewed .....	03
2.3) Identification of gaps .....	03
3) System Analysis .....	04
3.1) Requirements Specification .....	04
3.1.1) Functional Requirements .....	04
3.1.2) Non-functional Requirements .....	06
3.2) Feasibility Study .....	07
3.3) Tools and Technologies .....	07
4) Design .....	08
4.1) System Architecture .....	08
4.2) Diagrams .....	08
4.3) Database Design .....	11
5) Implementation .....	15
5.1) Description of Modules/Components .....	15
5.2) Code Snippets .....	15
5.3) Screenshots of the Application/System .....	23
6) Testing .....	36
6.1) Test Cases and Testing Methodology .....	36
6.2) Results of Testing .....	37
6.3) Bug Fixes/Issues Resolved .....	40
7) Results and Discussion .....	41
7.1) Outcomes of the project .....	41

## Warehouse Management System

7.2) Comparison with existing solutions .....	41
7.3) Challenges Faced .....	42
8) Conclusion and Future Scope .....	43
8.1) Summary of the work .....	43
8.2) Limitations .....	43
8.3) Future Enhancements .....	43
References .....	45

## 1) Introduction

### 1.1) Problem Statement

- Managing warehouse operations effectively is challenging. Common issues include inaccurate inventory tracking, inefficient use of storage space, slow order processing, and a lack of real-time visibility into stock levels. These problems lead to higher costs, delays, and dissatisfied customers.
- A Warehouse Management System (WMS) is needed to automate tasks, improve inventory accuracy, optimize space utilization, and streamline order fulfillment. This will enhance efficiency, reduce costs, and improve customer satisfaction.

### 1.2) Objective

- This Warehouse Management System (WMS) is designed to help warehouse operations be more efficient, accurate, and productive. Using this system, user can track inventory in real time, streamline order fulfillment, and optimize warehouse space. WMS reduces operational costs, minimizes errors, and improves workflow by automating key tasks like inventory receiving, picking, packing, and shipping.
- We want to solve common problems such as inaccurate inventory records, late order processing, and labor-intensive operations. It'll also give businesses data-driven insights to make better decisions. In the end, we want to make sure orders are fulfilled accurately and on time, so we can maximize warehouse efficiency and profits.

### 1.3) Scope

- The scope of this Warehouse Management System (WMS) is limited to managing the operations of a single company's warehouse, focusing on inventory management, order processing, picking, packing, shipping, reporting and analytics. It will generate detailed reports and analytics for better decision-making.

- The project will not include multi-warehouse management, or complex integrations. It will not include return and replace options.

## 1.4) Organization of the Report

- **Introduction:** Describes warehouse management challenges, objectives of the proposed WMS, and its scope, focusing on improving efficiency, reducing errors, and optimizing operations for a single warehouse.
- **Methodology:** Explains the Agile development approach and the tech stack, including Flask for the backend, MongoDB for scalable storage, and a responsive UI using HTML, CSS, and JavaScript.
- **System Design and Features:** Covers system architecture and core functionalities like user management, real-time inventory tracking, automated order processing, shipping, and detailed analytics.
- **Implementation:** Highlights the development process, focusing on inventory management, order workflows, and reporting integration.
- **Expected Outcomes:** Discusses benefits such as enhanced efficiency, reduced costs, data-driven decisions, and improved customer satisfaction.
- **Limitations and Future Scope:** Outlines the exclusions, like multi-warehouse management and return processing, with recommendations for future enhancements.
- **Conclusion:** Summarizes the importance of the WMS in addressing operational challenges and achieving better performance.

## 2) Literature Survey/Review

### 2.1) Overview of existing

- Warehouse Management Systems (WMS) streamline inventory, order fulfillment, and logistics through various types of solutions. Legacy systems are robust but lack flexibility, while cloud-based WMS offers scalability, real-time updates, and accessibility. ERP-integrated WMS ensures unified business operations, and AI-powered systems optimize processes with automation and demand prediction. Industry-specific solutions address compliance and traceability, while open-source WMS provides cost-effective, customizable options.

### 2.2) Project Reviewed

- The project focuses on developing an efficient Warehouse Management System (WMS) to optimize inventory control, order processing, and logistics. It integrates advanced technologies like cloud computing for scalability, AI for automation and demand forecasting, and IoT for real-time tracking. Existing systems such as legacy, cloud-based, ERP-integrated, and industry-specific WMS were reviewed to emphasizing the need for a flexible, scalable, and cost-effective solution.

### 2.3) Identification of gaps

- Key gaps in existing WMS include limited scalability in legacy systems, high implementation costs, and challenges with integration and customization. Many systems lack real-time visibility, have complex adoption processes requiring technical expertise, and security concerns in cloud-based solutions. Addressing these gaps can enhance WMS accessibility, efficiency, and adaptability.

### 3) System Analysis

#### 3.1) Requirements Specification

##### 3.1.1) Functional Requirements

###### **Admin and Manager:**

- 1) Login:** Allows the admin and manager to securely access the system using authentication.
- 2) View Dashboard:** Displays an overview of key warehouse metrics and activities in a user-friendly interface.
- 3) Manage Users:** Enables the admin to add, update, and delete user accounts and roles within the system.
- 4) Manage Products:** Provides functionality to add, update, and delete product details.
- 5) Manage Stock:** Allows the user to track inventory levels and update stock details in real-time.
- 6) View Clients:** Displays a list of clients associated with the warehouse operations.
- 7) Manage Orders:** Facilitates tracking and delivery of orders, ensuring timely and accurate order fulfillment.
- 8) View Reports:** Generates detailed analytics and reports for decision-making.
- 9) View Profile:** Lets the user update their profile information and reset passwords.
- 10) Manage Permission:** Enables the admin to assign and manage user access permissions within the system.

**Employee:**

- 1) Login:** Allows the employee to securely access the system using authentication.
- 2) View Orders:** Allows the employee to check details of received orders, including status and packing orders.
- 3) Manage Tasks:** Employees can view assigned tasks, update task status, and mark tasks as completed to streamline daily operations.
- 4) View Profile:** Lets the user update their profile information and reset passwords.

**Supplier:**

- 1) Login:** Allows the supplier to securely access the system using authentication.
- 2) View Orders:** Allows the supplier to check details of received orders, including status and delivery requirements.
- 3) Deliver Order:** Provides functionality for suppliers to mark orders as delivered after completing the shipment.
- 4) Receive Payment:** Enables the supplier to confirm and track payments received for delivered orders.
- 5) Print Invoice:** Allows the supplier to generate and print invoices for delivered orders.
- 6) View Profile:** Lets the user update their profile information and reset passwords.

**Client:**

- 1) Register:** Clients must register an account by providing details and verifying their identity through an OTP (One-Time Password).

- 2) Login:** Clients log in securely using their credentials, with authentication ensuring authorized access.
- 3) View Products:** Clients can browse and view a list of available products, including details like quantity and price.
- 4) Add Orders:** Clients can place new orders by selecting products and specifying quantities.
- 5) Track Order:** Clients can monitor the status and delivery progress of their orders.
- 6) View Order History:** Clients can access a detailed history of past orders, including statuses and payment details.
- 7) View Profile:** Lets the user update their profile information and reset passwords.

### **3.1.2) Non-functional Requirements**

- 1) Security and Data Protection:** All sensitive data, including user details, orders, and stock information, must be encrypted during storage and transmission. The system should implement role-based access control (RBAC) to ensure only authorized users can perform specific actions.
- 2) Email Service:** The system should send email notifications to clients upon successful order placement, including order details and tracking links. Users should receive emails during registration, account updates, or password resets.
- 3) Usability:** The interface should be intuitive and accessible, allowing users to perform tasks (e.g., tracking orders, managing stock) with minimal training.
- 4) Availability:** The system must be available 99.9% of the time, ensuring continuous service with minimal downtime.

### 3.2) Feasibility Study

- **Technical Feasibility:** Python, MongoDB, and web frameworks (Flask) are used to build robust and scalable systems as part of the project. Using tools like PyCharm and MongoDB Compass speeds up the development process by enabling efficient coding, debugging, and data analysis.
- **Economic Feasibility:** Automating order processing and inventory management will improve warehouse efficiency, reduce operational costs, and enhance customer satisfaction. Open-source tools (Python, MongoDB) reduce development costs.
- **Operational Feasibility:** WMS features an intuitive interface designed for warehouse personnel and managers, enabling quick adoption. Role-based access control ensures proper and secure use of system features, minimizing operational risks.

### 3.3) Tools and Technologies

#### Technologies:

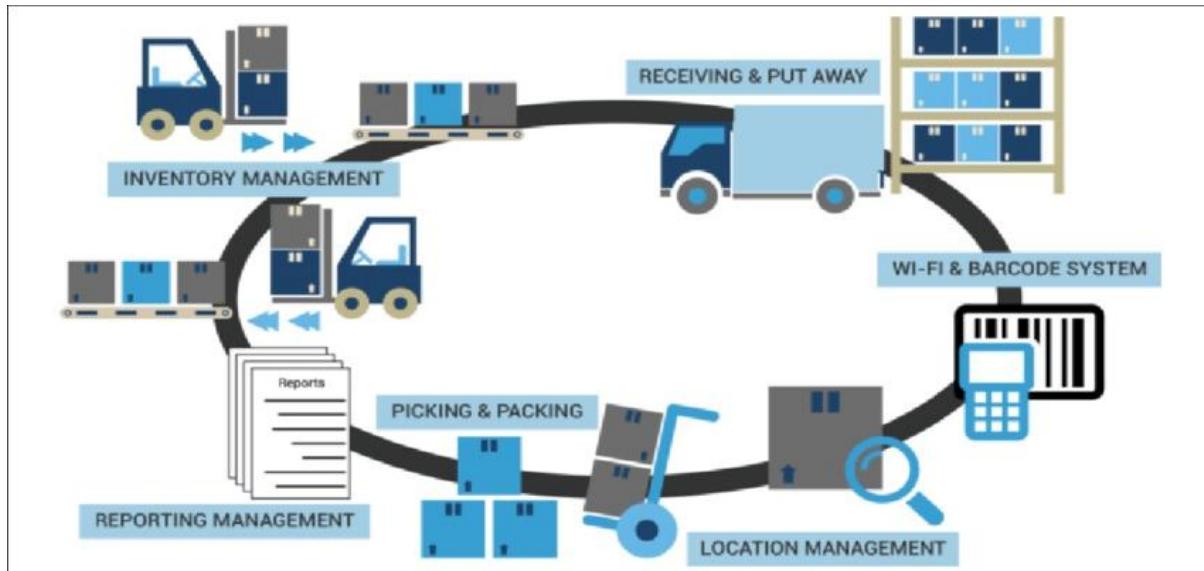
- **Frontend:** HTML, CSS, and JavaScript will be used to create a responsive and interactive user interface.
- **Backend:** The backend will be handled by Python using Flask, providing a robust environment for API integrations and server-side logic.
- **Database:** Data will be stored in MongoDB for scalability, quick data retrieval, and inventory data handling.

#### Tools:

- **PyCharm:** The PyCharm IDE is an integrated development environment (IDE) for writing, testing, and debugging Python code. The WMS backend will be developed using PyCharm's features for efficient coding.
- **MongoDB:** The MongoDB Compass graphical user interface (GUI) allows users to query, aggregate, and analyze data visually. A user-friendly interface provides an intuitive environment for exploring and interacting with data.

## 4) Design

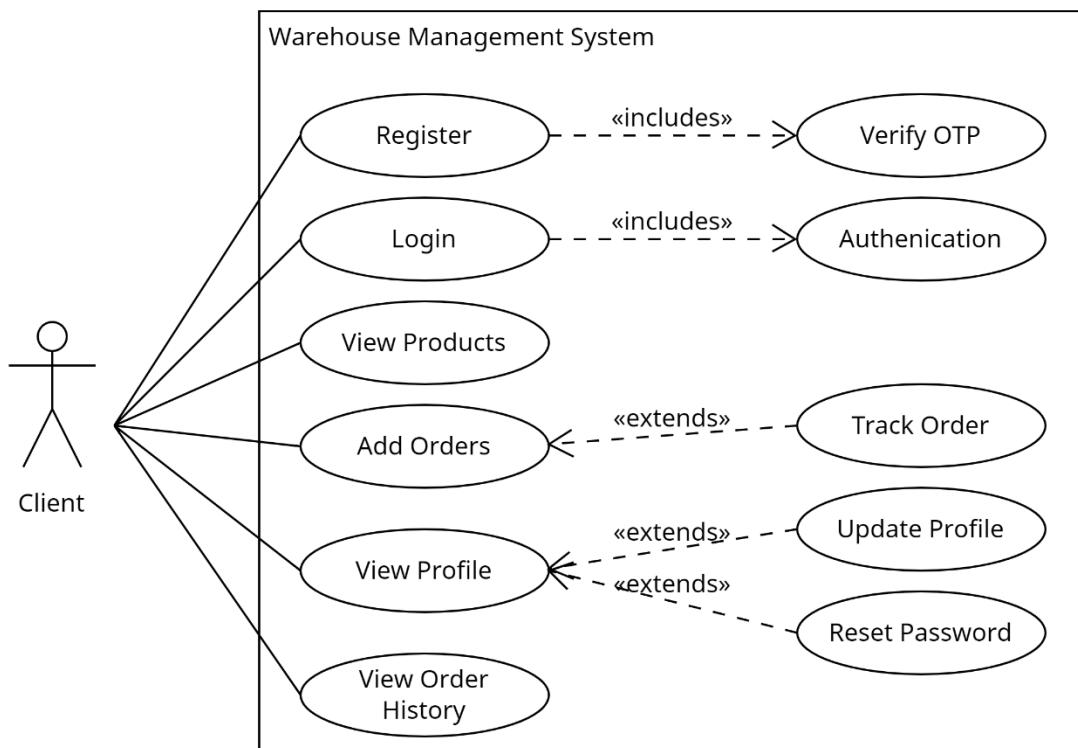
### 4.1) System Architecture



Warehouse Management System

### 4.2) Diagrams

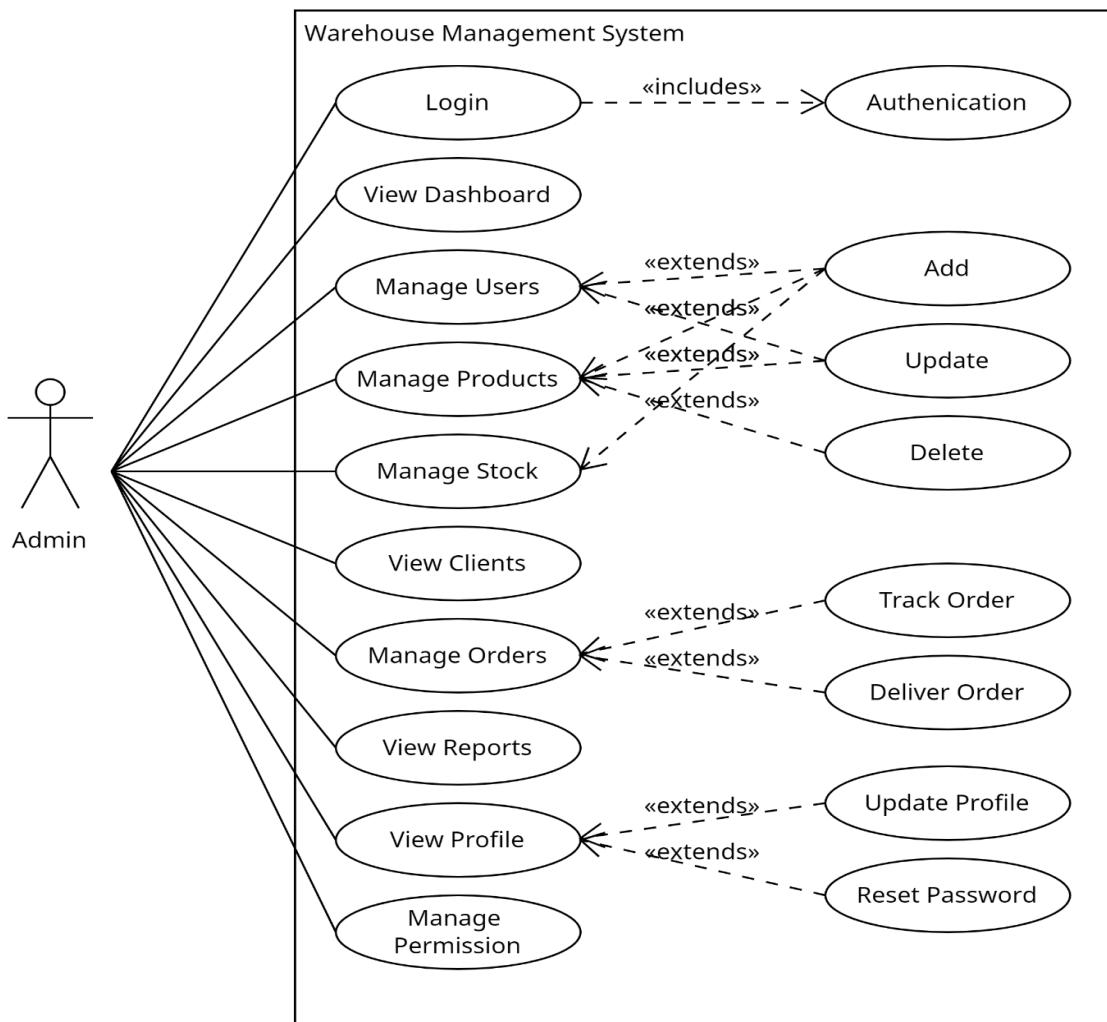
Client:



Client Use Case Diagram

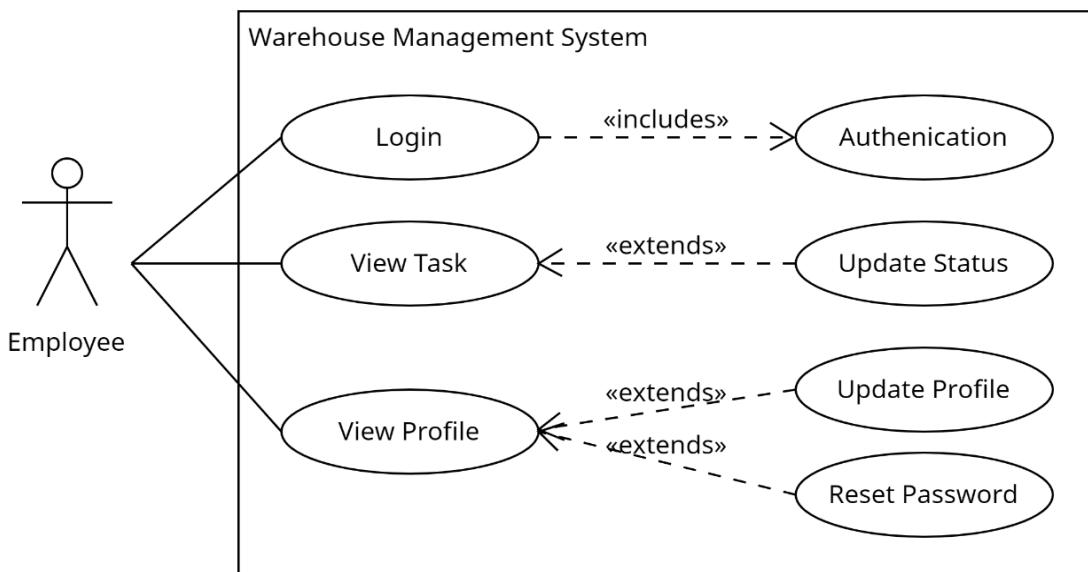
## Warehouse Management System

**Admin:**



Admin Use Case Diagram

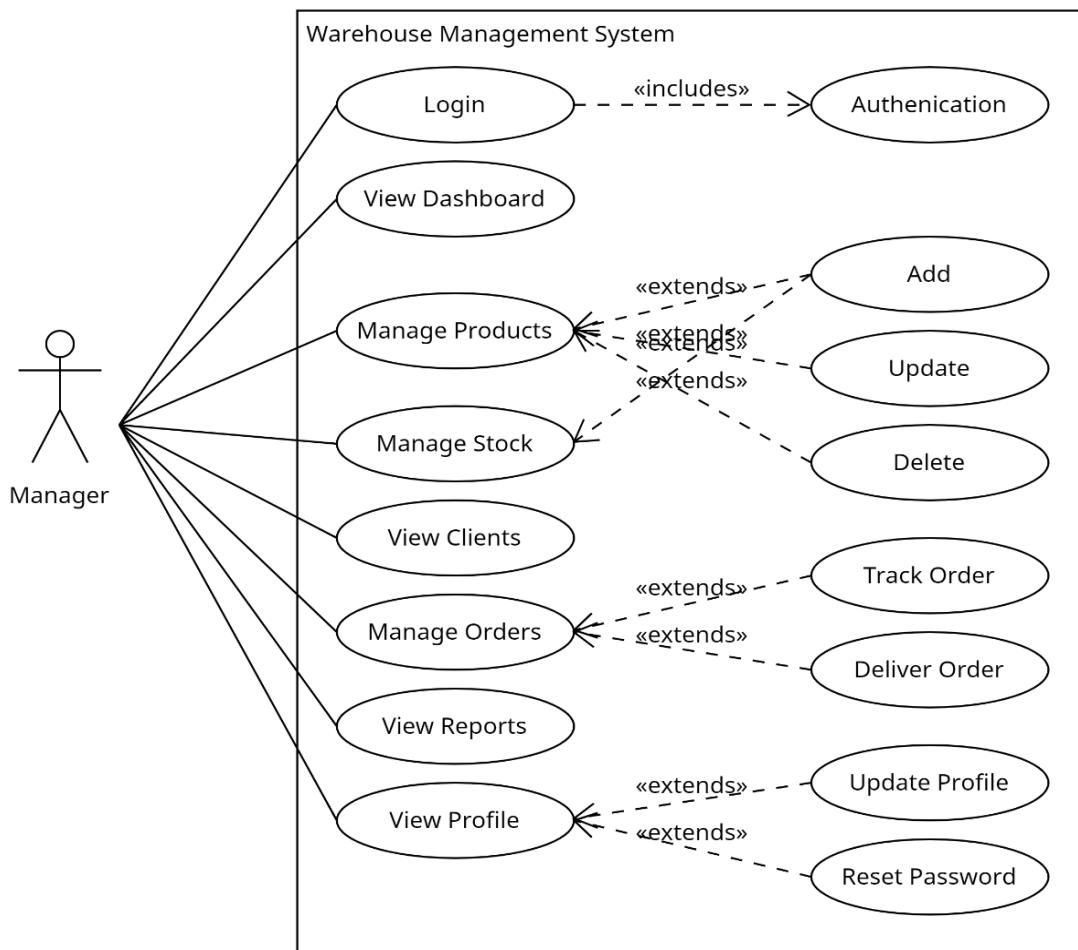
**Employee:**



Employee Use Case Diagram

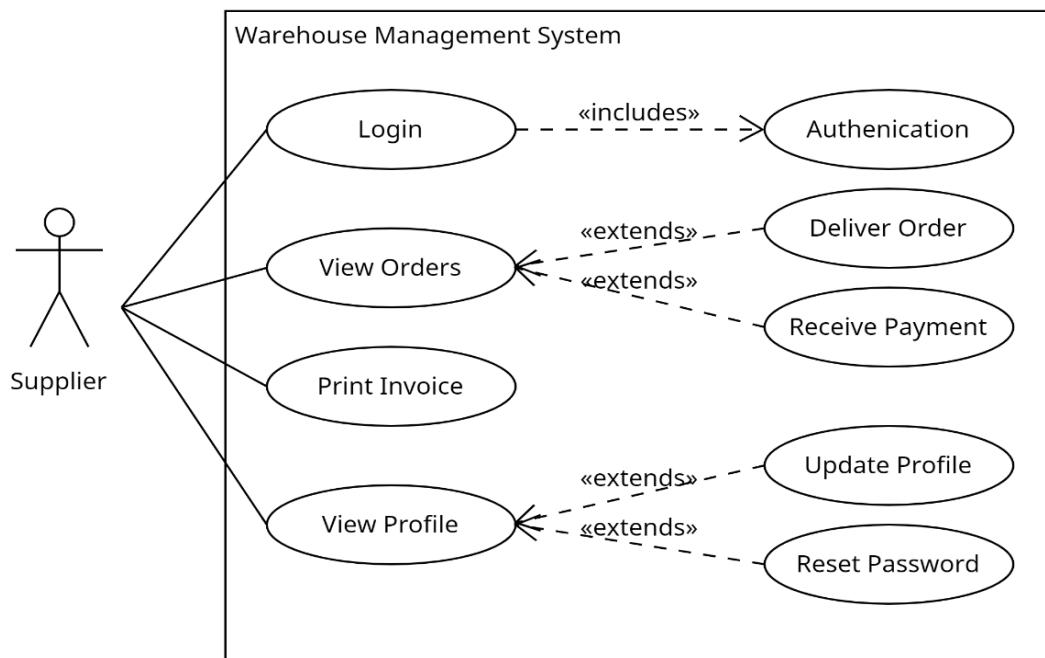
## Warehouse Management System

**Manager:**



**Manager Use Case Diagram**

**Supplier:**



**Supplier Use Case Diagram**

### 4.3) Database Design

#### user

Attribute	Data Type	Constraints
_id	ObjectId	Primary Key, Not Null, Unique
full_name	String	Not Null
mobile	String	Not Null, Unique
email	String	Not Null, Unique
password	String	Not Null
role	String	Not Null
status	String	Not Null
city	String	Not Null
area	String	Not Null

#### product

Attribute	Data Type	Constraints
_id	ObjectId	Primary Key, Not Null, Unique
product_name	String	Not Null
sku	Long	Primary Key, Not Null, Unique
unit	Integer	Not Null
min	Integer	Not Null
max	Integer	Not Null
price	Float	Not Null
status	String	Not Null
description	String	Not Null
created_by	String	Not Null
area_id	ObjectId	Not Null, Foreign Key (References area(area_id))
image	String	Not Null
created_at	Date	Not Null

**area**

<b>Attribute</b>	<b>Data Type</b>	<b>Constraints</b>
_id	ObjectId	Primary Key, Not Null, Unique
area_name	String	Not Null, Unique
no_box	Integer	Not Null
status	String	Not Null

**stock**

<b>Attribute</b>	<b>Data Type</b>	<b>Constraints</b>
_id	ObjectId	Primary Key, Not Null, Unique
sku	Long	Not Null, Unique, Foreign Key (References products(sku))
date	Date	Not Null
+qty	Integer	Not Null
-qty	Integer	Not Null
total_qty	Integer	Not Null

**order**

<b>Attribute</b>	<b>Data Type</b>	<b>Constraints</b>
_id	ObjectId	Primary Key, Not Null, Unique
user_id	ObjectId	Not Null, Foreign Key (References users(user_id))
products	Array of Object	Not Null
total_amount	Float	Not Null
payment_type	String	Not Null
order_date	Date	Not Null
status	Array of Object	Not Null
otp	Integer	Not Null

**transactions**

<b>Attribute</b>	<b>Data Type</b>	<b>Constraints</b>
_id	ObjectId	Primary Key, Not Null, Unique
order_id	ObjectId	Not Null, Foreign Key (References orders(order_id))
amount	Float	Not Null
payment_type	String	Not Null
payment_status	String	Not Null
transaction_date	Date	Not Null

**case\_collections**

<b>Attribute</b>	<b>Data Type</b>	<b>Constraints</b>
_id	ObjectId	Primary Key, Not Null, Unique
order_id	ObjectId	Not Null, Foreign Key (References orders(order_id))
amount	Float	Not Null
submission_date	Date	Not Null
supplier_id	ObjectId	Not Null, Foreign Key (References suppliers(supplier_id))
status	String	Not Null

**page\_visibility**

<b>Attribute</b>	<b>Data Type</b>	<b>Constraints</b>
_id	ObjectId	Primary Key
page_name	String	Not Null, Unique,
roles	Object	Not Null

**assign\_tasks**

Attribute	Data Type	Constraints
_id	ObjectId	Primary Key, Not Null, Unique
order_id	ObjectId	Not Null, Foreign Key (References orders(order_id))
employee_id	ObjectId	Not Null, Foreign Key (References users(user_id))
assigned_date	Date	Not Null
status	String	Not Null
supplier_id	ObjectId	Not Null, Foreign Key (References suppliers(supplier_id))

## 5) Implementation

### 5.1) Description of Modules/Components

- **User Management:** Admins can create, manage, and assign roles to users, such as warehouse managers and employees, using the User Management module of the Warehouse Management System (WMS).
- **Inventory Management:** The module handles inventory tracking, management, and updates within a warehouse. It allows real-time stock monitoring, keeping track of product quantities and locations, as well as stock adjustments and low stock alerts.
- **Order Processing:** This module streamlines order handling, from order creation to fulfillment. This module automates order processing, so warehouse employees can get orders, allocate inventory, update order statuses, and prioritize order processing based on urgency and customer needs.
- **Picking and Packing:** A module that enables efficient picking and packing of orders. It guides warehouse staff through the most efficient routes based on inventory location and order priority.
- **Shipping and Dispatch:** Order fulfillment includes preparing shipments, generating shipping labels, and dispatching products to customers.
- **Reporting and Analytics:** A warehouse admin can generate detailed reports on inventory levels, order processing times, picking efficiency, and overall warehouse productivity with this module, which provides detailed reports and analytics to aid in data-driven decision-making.

### 5.2) Code Snippets

#### 1) Mail Sent Code:

```
# Flask-Mail Configuration
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'shubhamlathiya2021@gmail.com'
app.config['MAIL_PASSWORD'] = 'tquerujnjzuvgdjho'
```

```
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True

# Initialize the mail object
mail = Mail(app)

from email.mime.text import MIMEText
from flask_mail import Message, Mail
from flask import current_app

mail = Mail()

def send_email(subject, recipient_email, body_html):
    try:
        # Create the Flask-Mail message object
        msg = Message(subject=subject,
                      sender=current_app.config['MAIL_USERNAME'],
                      recipients=[recipient_email])

        # Attach the HTML body to the message
        msg.html = body_html

        # Send the email using Flask-Mail
        mail.send(msg)

    return True
    except Exception as e:
        print(f"Error sending email: {e}")
        return False
```

## 2) Payment Code:

```
const razorpayOptions = {
    key: 'rzp_test_51mQEdclvr946E', // Replace with your
Razorpay key
    amount: data.amount, // Amount in paise (multiply by 100
to convert INR to paise)
    currency: 'INR',
```

```

name: 'Your Store Name',
description: 'Order Payment',
order_id: data.razorpay_order_id, // Razorpay order ID
from backend
success
    handler: function (response) {
        // Step 3: Submit order data directly after payment
        fetch('/client/submitOrder', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify({
                products: orderList, // Array of products in the
                order
                totalAmount: orderData.totalAmount,
                paymentType: 'Razorpay', // Payment type is
                Razorpay
                razorpay_payment_id:
                response.razorpay_payment_id, // Razorpay Payment ID
                razorpay_order_id: response.razorpay_order_id, //
                Razorpay Order ID
                razorpay_signature: response.razorpay_signature // //
                Razorpay Signature
            })
        })
        .then(orderRes => orderRes.json())
        .then(orderData => {
            if (orderData.status === 'success') {
                alert('Order placed successfully!');
                window.location.href = orderData.url; // Redirect
                to the success page
            } else {
                alert(orderData.message); // Error handling if
            }
        })
    }
}

```

```

    .catch(error => {
      console.error('Error submitting order:', error);
      alert('There was an error placing your order!');
    });
  },
  prefill: {
    name: 'Customer Name',
    email: 'customer@example.com',
    contact: '9999999999'
  },
  notes: {
    address: 'Customer Address'
  },
  theme: {
    color: '#F37254'
  }
};

// Open Razorpay payment gateway
const razorpay = new Razorpay(razorpayOptions);
razorpay.open();

```

**3) QR Code Scanner Code:**

```

import cv2
from pyzbar.pyzbar import decode
import os

# Beep sound parameters (only for Windows)
def play_beep():
  if os.name == 'nt': # Only play beep on Windows
    import winsound
    frequency = 1000 # Set Frequency To 1000 Hertz
    duration = 500 # Set Duration To 500 ms (0.5 seconds)
    winsound.Beep(frequency, duration)
  else:
    # Alternative sound or print message on non-Windows systems
    print("\a") # ASCII Bell character for simple beep sound

```

```
# Capture webcam
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Error: Could not open camera.")
    exit()

while cap.isOpened():
    success, frame = cap.read()
    if not success:
        print("Error: Could not read frame from camera.")
        break

    # Flip the image like a mirror image
    frame = cv2.flip(frame, 1)

    # Detect the barcode
    detectedBarcode = decode(frame)

    for barcode in detectedBarcode:
        # Decode the barcode data
        barcode_data = barcode.data.decode('utf-8')
        if barcode_data:
            print(f"{barcode_data}")
            # Display the barcode data on the frame
            cv2.putText(frame, barcode_data, (50, 50),
                       cv2.FONT_HERSHEY_COMPLEX, 2, (0, 255, 255), 2)

        # Play beep sound
        play_beep()

    # Save the frame containing the barcode as an image
    cv2.imwrite("code.png", frame)

    # Exit the program after detecting a barcode
    cap.release()
    cv2.destroyAllWindows()
```

```
exit()

# Show the video feed with the scanner window
cv2.imshow('scanner', frame)

# Press 'q' to quit the program manually
if cv2.waitKey(1) == ord('q'):
    break

# Release the webcam and close windows
cap.release()
cv2.destroyAllWindows()
```

#### 4) Monitor Stock Level:

```
def monitor_stock_levels():
    from app import mongo, app
    print("Scheduler started.")

    # Fetch all products from the products collection
    products = mongo.db.products.find({})

    for product in products:
        # Fetch the latest stock entry for this product
        stock_record = mongo.db.stock.find_one(
            {'sku': product['sku']}, sort=[('date', -1)]
        )

        if not stock_record:
            # If there's no stock record, continue to the next product
            continue

        current_stock_qty = stock_record['total_qty']
        min_stock_threshold = product.get('min', 10) # Fetch min threshold or
        default to 10
        notification_sent = product.get('notification_sent', False) # Check if
        notification has already been sent
        # Check if current stock is below the threshold
```

```

if current_stock_qty < min_stock_threshold:
    if not notification_sent:
        # Send notification only if it hasn't been sent already
        send_notification(product, current_stock_qty, min_stock_threshold)
        # Update the product to mark that the notification has been sent
        mongo.db.products.update_one(
            {'_id': ObjectId(product['_id'])},
            {'$set': {'notification_sent': True}}
        )
        print(f"Low stock notification sent for {product['product_name']}"
        (SKU: {product['sku']})) (stock : {current_stock_qty})
    else:
        print(f"Notification already sent for {product['product_name']}"
        (SKU: {product['sku']}"))
    else:
        # If stock is replenished, reset the notification_sent field
        if notification_sent:
            mongo.db.products.update_one(
                {'_id': ObjectId(product['_id'])},
                {'$set': {'notification_sent': False}}
            )
            print(f"Stock replenished for {product['product_name']} (SKU:
            {product['sku']}), notification flag reset.")
        else:
            print(f"Stock is sufficient for {product['product_name']} (SKU:
            {product['sku']})")
            print("Scheduler stopped.")

```

## 5) Assign Orders for Employee:

```

def assign_order_to_employee(order_id):
    # Fetch the queue of employees who have role "employee" and status
    "true"
    employees = list(mongo.db.users.find(
        {'role': 'employee', 'status': 'true'})) # Assuming the role and status are
    in 'users' collection
    if not employees:

```

```
return None

# Fetch the current queue pointer (index of the last assigned employee)
queue_data = mongo.db.queue.find_one({}) # Assuming the queue
information is stored in a 'queue' collection

if not queue_data:
    # If no queue data exists, initialize it with the first employee
    queue_data = {
        'current_employee_index': 0
    }
    mongo.db.queue.insert_one(queue_data)
# Get the current employee index
current_employee_index = queue_data['current_employee_index']

# Find the next employee in the queue
next_employee_index = (current_employee_index + 1) % len(employees)
# Round-robin logic

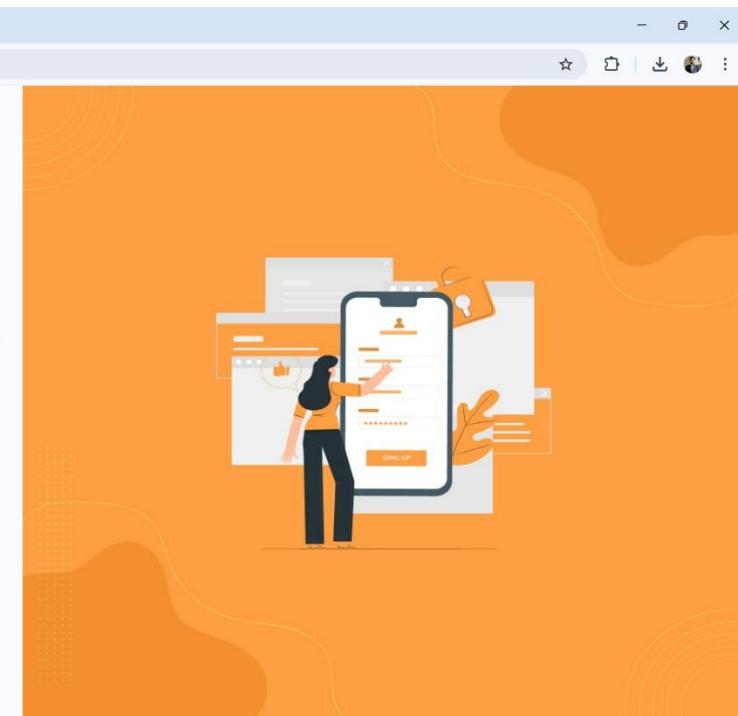
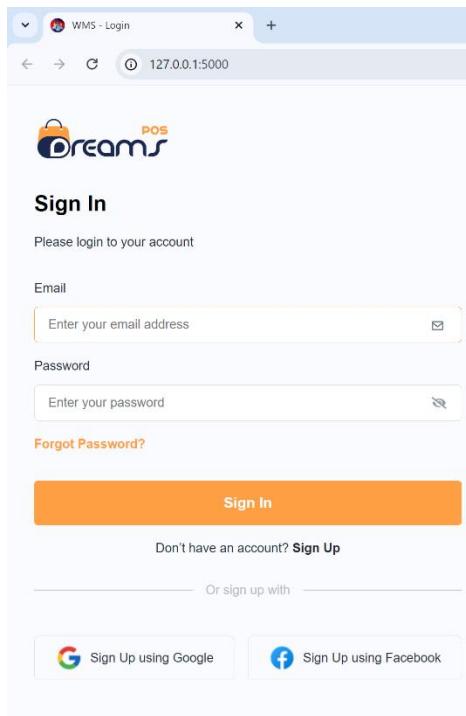
# Assign the next employee
assigned_employee = employees[next_employee_index]

# Update the queue pointer to the new employee
mongo.db.queue.update_one({}, {'$set': {'current_employee_index':
next_employee_index}})

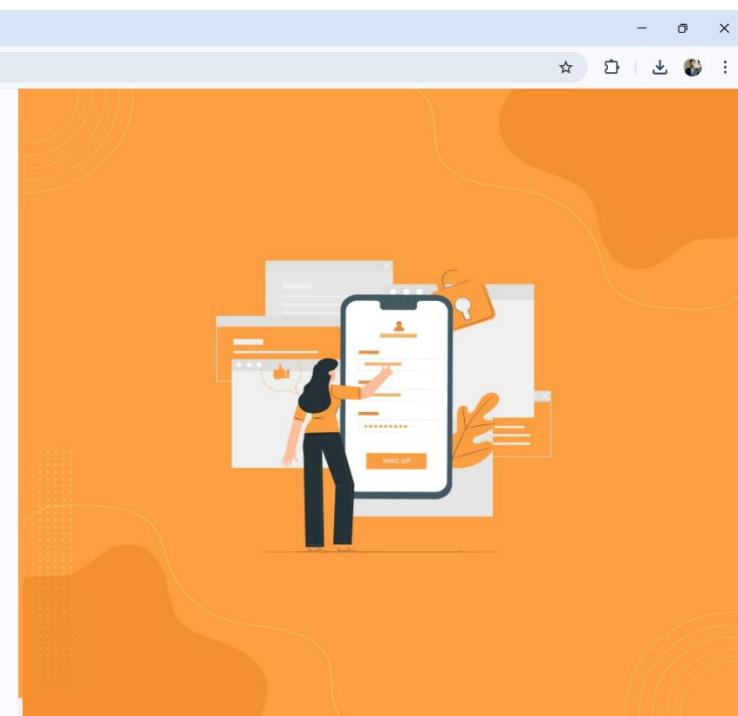
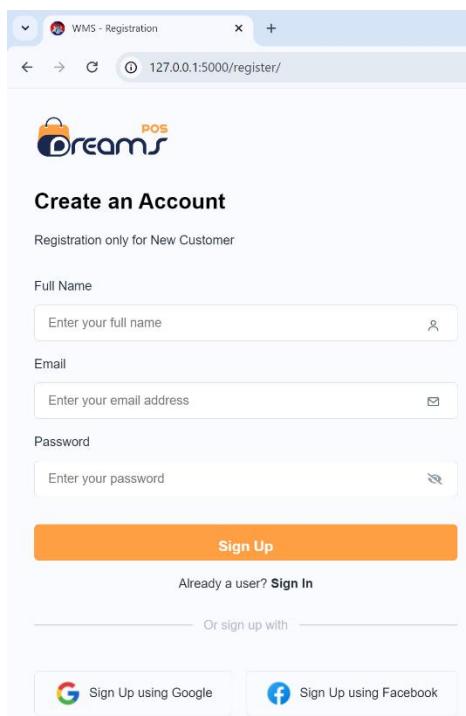
# Create an assignment record in the 'assignments' collection
assignment = {
    'order_id': ObjectId(order_id),
    'employee_id': assigned_employee['_id'],
    'assigned_date': datetime.now(),
    'status': 'Assigned'
}
mongo.db.assigned_tasks.insert_one(assignment)
return assigned_employee
```

## 5.3) Screenshots of the Application/System

### 1) User:

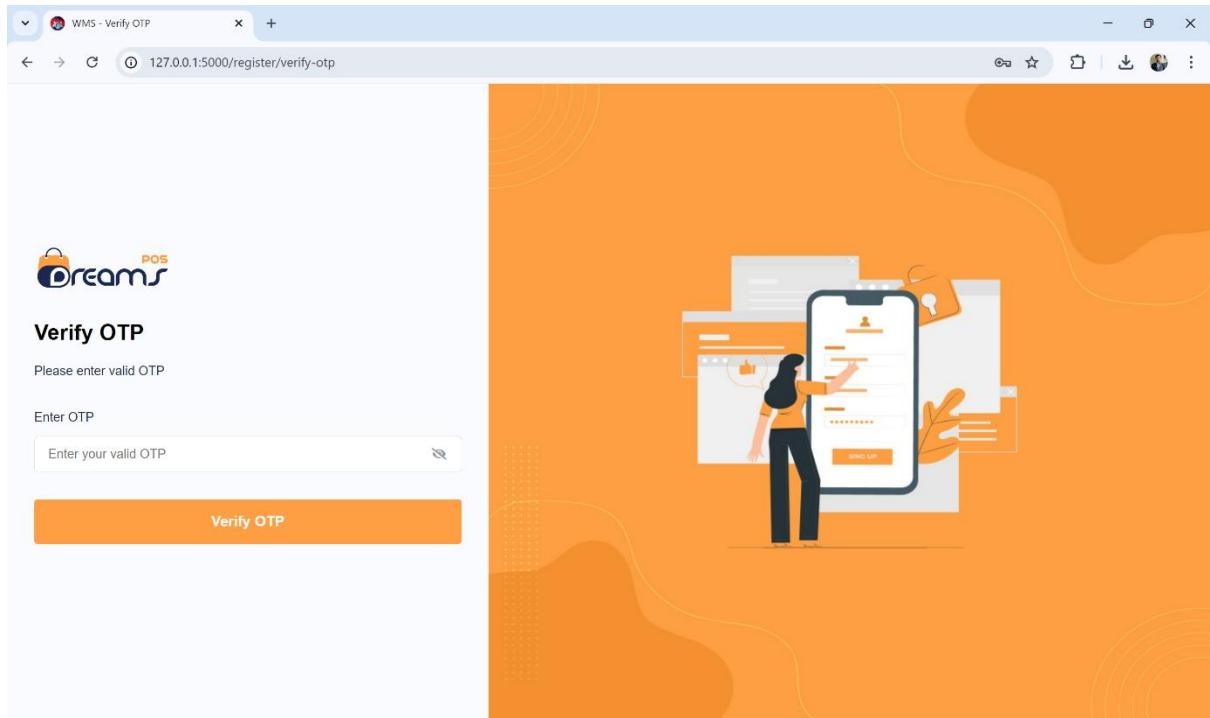


Sign In

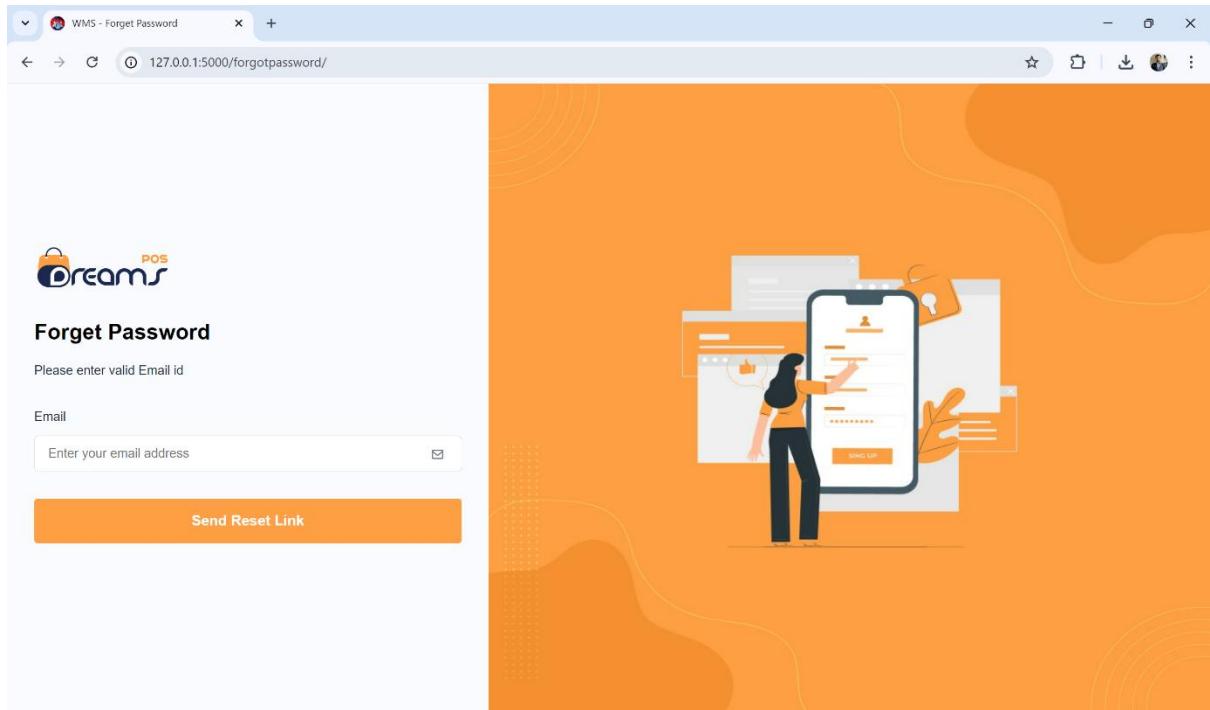


Sign Up

# Warehouse Management System



## OTP Verify



## Forget Password

# Warehouse Management System

The screenshot shows the WMS - WMS - Profile page at 127.0.0.1:5000/register/profile. The interface includes a sidebar with links for Dashboard, Users, Product, Stock, Clients, Orders, and Report. The main content area displays a user profile for 'Shubham'. It features a placeholder for a profile picture, the name 'Shubham', and a note 'Updates Your Photo and Personal Details.' Below this are input fields for First Name (Shubham), Mobile No (7041138931), Email (shubhamlathiya2004@gmail.com), City (Surat), Area (Nana Varachha), and Address. A large orange 'Submit' button is at the bottom.

## View Profile

## 2) Admin and Manager:

The screenshot shows the WMS - Dashboard page at 127.0.0.1:5000/admin/dashboard. The sidebar has links for Dashboard, Users, Product, Stock, Clients, Orders, and Report. The main dashboard features four summary boxes: 'Total Purchase Due' (\$307144), 'Total Sales Due' (\$4385), 'Total Sale Amount' (\$385656.5), and 'Total Sale Amount' (\$40000). Below these are four cards showing counts: 9 Customers, 2 Suppliers, 4 Employees, and 2 Total orders. Two tables follow: 'Recently Added Products' and 'Recently Added Stocks'. The 'Recently Added Products' table lists four items: 1. Masala Sev Murruma (Price: 180.0), 2. Masala Shing (Price: 190.0), 3. Crunchex – Simply Salted (Price: 110.0), and 4. Crunchex – Mirch Masala (Price: 130.0). The 'Recently Added Stocks' table lists four items: 1. CP – Masala Masti (Stocks: 50), 2. Banana Wafers – Masala (Stocks: 80), 3. Crunchem – Pizzy Masala (Stocks: 100), and 4. Crunchex – Mirch Masala (Stocks: 900).

## Admin Dashboard

# Warehouse Management System

The screenshot shows the 'User List' page of the Warehouse Management System. The page title is 'WMS - View Users'. The URL in the browser is 127.0.0.1:5000/user/viewuser. The left sidebar has a 'Users' dropdown menu with 'New User' (radio button), 'Users List' (radio button, selected), and 'Supplier List' (radio button). Other menu items include 'Product', 'Stock', 'Clients', 'Orders', and 'Report'. The main content area is titled 'User List' and 'Manage Users'. It features a search bar and a table with the following data:

No	First name	Phone	Email	City	Role	Status	Action
1	Shubham	7041138931	shubhamlathiya2004@gmail.com	Surat	admin	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
2	Rutvik	7862016740	rutveeksavaliya223@gmail.com	Banglore	employee	<input type="checkbox"/>	<a href="#">Edit</a>
3	Priyank	7451289301	priyankmangroliya1@gmail.com	Pune	employee	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
4	Shubham lathiya	9876543210	shubhamlathiya2021@gmail.com	Surat	manager	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
5	Suzanne	9329646346	tsuzanne376@gmail.com	Surat	employee	<input checked="" type="checkbox"/>	<a href="#">Edit</a>
6	Rutvik	7458120369	rutusavaliya5@gmail.com	Junagadh	employee	<input checked="" type="checkbox"/>	<a href="#">Edit</a>

Show per page: 10 1 - 6 of 6 items

## View Users

The screenshot shows the 'User Management' page for adding a new user. The page title is 'WMS - Add User'. The URL in the browser is 127.0.0.1:5000/user/adduser. The left sidebar has a 'Users' dropdown menu with 'New User' (radio button, selected), 'Users List' (radio button), and 'Supplier List' (radio button). Other menu items include 'Product', 'Stock', 'Clients', 'Orders', and 'Report'. The main content area is titled 'User Management' and 'Add User'. It features a form with the following fields:

First Name	Password	
<input type="text"/>	<input type="password"/> <a href="#">Show</a>	
Mobile No	Email	
<input type="text"/>	<input type="text"/>	
City	Area	Role
<input type="text"/>	<input type="text"/>	<input type="text"/> Manager

Buttons: [Submit](#) [Reset](#)

## Manage User

# Warehouse Management System

The screenshot shows the 'Product Management' section of the WMS. On the left, a sidebar menu is open under the 'Product' category, showing options like 'New Product', 'Product List', and 'Manage Area'. The main area is titled 'Product Management' and 'Product List'. It features a search bar and a table with columns: No, Product Image, Product Name, SKU, Price, Min Qty, Max Qty, Area, Status, and Action. Three product entries are listed:

No	Product Image	Product Name	SKU	Price	Min Qty	Max Qty	Area	Status	Action
1		Crunchem – Pizzi Masala	704113893101	120.0	10	100	A2	available	
2		Crunchem – Cream & Onion	704113893102	130.0	10	100	A3	available	
3		Crunchem – Chaat Chaska	704113893103	150.0	10	100	A4	available	

## View Products

The screenshot shows the 'Product Management' section with the 'Add Product' sub-section active. The sidebar menu is identical to the previous screenshot. The main area has a form for adding a new product. Fields include Product Name, SKU, Price, Max Qty, Area, Description, and Status. There is also a 'Product Image' section with a file upload input and a preview area showing a stack of three cards with a red sun icon on top.

Product Name	SKU	Product Image
<input type="text"/>	<input type="text"/> Scan	<input type="file"/> Choose file...
Price	Min Qty	
Max Qty	Unit	
Area	Status	
Description		

## Manage Product

# Warehouse Management System

The screenshot shows the 'Stock Management' section of the WMS. At the top, there are tabs for 'Scanning' and 'Searching', with 'Scanning' selected. Below this, the product details are displayed: Product Name: Crunchex – Simply Salted and Product SKU: 704113893109. To the right, there is a field labeled 'Add Stock:' with a 'Submit' button. Below these fields is a table showing stock history:

Sr. No.	Date	+ Quantity	- Quantity	Total Quantity	Action
1	22/10/2024, 02:08:38	0	2	493	
2	22/10/2024, 02:06:33	0	5	495	
3	22/10/2024, 02:05:46	500	0	500	

## Manage Stock

The screenshot shows the 'Product Management' section. On the left, there is a sidebar with navigation links: Dashboard, Users, Product (selected), New Product, Product List, Manage Area, Stock, Clients, Orders, and Report. The main area displays a form for adding a new product. A camera viewfinder window titled 'scanner' is overlaid on the form, showing a close-up of a QR code being scanned. The form fields include Product Name, SKU, Price, Min Q, Max Qty, Unit, Area (set to A10 (Box: 100)), Status, and Description. At the bottom are 'Submit' and 'Reset' buttons.

## Scan QR Code

# Warehouse Management System

The screenshot shows the 'Client List' section of the WMS. The table has the following data:

No	First name	Phone	Email	City	Status	Address
1	Riyank	7041138931	lathiaryank2004@gmail.com	Pune	On	Mohitewadi, Maharashtra
2	Priyank Mangaroliya	7415820369	priyankmangaroliya@gmail.com	Surat	On	Mota Varachha
3	PM	8598742630	pm8849288@gmail.com	Surat	On	APSO Complex, Chhatrapati Shivaji Interna
4	Swanandini	6638892845	swanandinikamble@gmail.com	Surat	On	Nalasopara
5	Priyamsi savaliya	2574189630	priyasmisavaliya01@gmail.com	Pune	On	Kanhe
6	Harsh	7415820369	harshjethwa7091@gmail.com	Surat	On	nana varachha
7	Shubham	7041138931	shubham.lathiya24@pcu.edu.in	Surat	On	Mota Varachha
8	priyank	7884526786	priyank.mangaroliya24@pcu.edu.in	Pune	On	Kanhe

## View Clients

The screenshot shows the 'Order List' section of the WMS. The table has the following data:

No	Name	Phone	Order Date	Product	Qty	Total Amount	Payment Status	Payment Type	Status
1	Priyamsi savaliya	2574189630	2024-10-19	Crunchem – Pizza Masala	2	500.0	Paid	Cash	Delivered
2	Priyamsi savaliya	2574189630	2024-10-19	Crunchem – Pizza Masala	1	400.0	Paid	Cash	Delivered
3	PM	8598742630	2024-10-20	Masala Sev Murrura	5	2800.0	Paid	Cash	Delivered
4	Shubham	7041138931	2024-10-20	Crunchem – Chaat Chaska	5	1390.0	Paid	Cash	Delivered
5	Shubham	7041138931	2024-10-20	Masala Sev Murrura	10	4400.0	Paid	Cash	Delivered
6	priyank	7884526786	2024-10-20	Crunchem – Cream & Onion	4	1240.0	Paid	Cash	Delivered
7	Harsh	7415820369	2024-10-21	Crunchem – Cream & Onion	4	1600.0	Pending	Cash	Assigned
8	Riyank	7041138931	2024-10-21	CP – Masala Masti	1	3060.0	Pending	Cash	Assigned to Supplier
9	Priyank Mangaroliya	7415820369	2024-10-21	CP – Masala Masti	1	490.0	Pending	Cash	Assigned
10	PM	8598742630	2024-10-21	CP – Masala Masti	1	640.0	Pending	Cash	Assigned

## View Orders

# Warehouse Management System

The screenshot shows a web browser window titled "WMS - Settings" with the URL "127.0.0.1:5000/settings/settings". The left sidebar contains navigation links: Dashboard, Users, Product, Stock, Clients, Orders, and Report. The main content area is titled "Users Permissions" and "View All Users". It features a search bar and a table with columns "No", "Roles", and "Action". The table data is as follows:

No	Roles	Action
1	admin	
2	client	
3	employee	
4	manager	
5	supplier	

At the bottom, there is a "Show per page" dropdown set to 10, a page number indicator "1 - 5 of 5 items", and a "1" button.

## Manage Role Permission

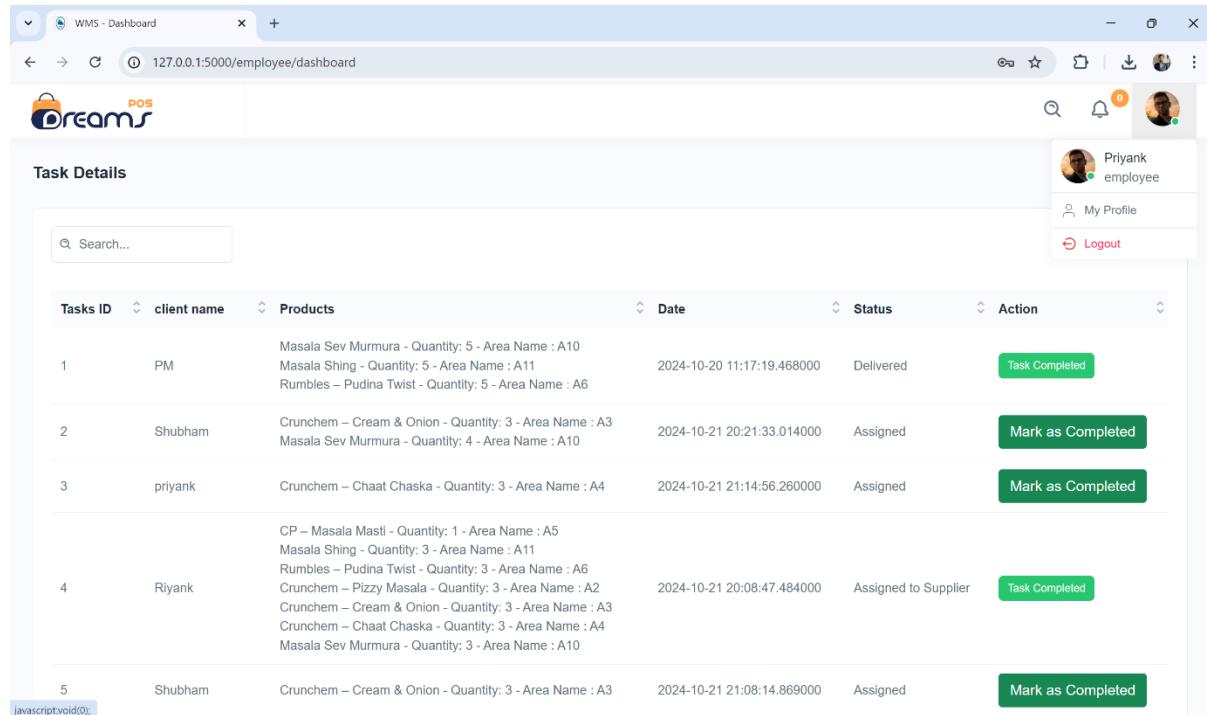
The screenshot shows a web browser window titled "WMS - Manage Permissions" with the URL "127.0.0.1:5000/settings/roles/admin". The left sidebar contains navigation links: Dashboard, Users, Product, Stock, Clients, Orders, and Report. The main content area is titled "Edit Permission" and "Manage Edit Permissions". It features a search bar and a table with columns "No", "Roles", and "Action". The table data is as follows:

No	Roles	Action
1	areas	<input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete
2	stocks	<input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete
3	users	<input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete
4	products	<input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete
5	admin_dashboard	<input checked="" type="checkbox"/> View <input checked="" type="checkbox"/> Create <input checked="" type="checkbox"/> Edit <input checked="" type="checkbox"/> Delete
6	client_dashboard	<input type="checkbox"/> View <input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
7	employee_dashboard	<input type="checkbox"/> View <input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
8	manager_dashboard	<input type="checkbox"/> View <input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete
9	supplier_dashboard	<input type="checkbox"/> View <input type="checkbox"/> Create <input type="checkbox"/> Edit <input type="checkbox"/> Delete

## Manage Permission

## Warehouse Management System

### 3) Employee:

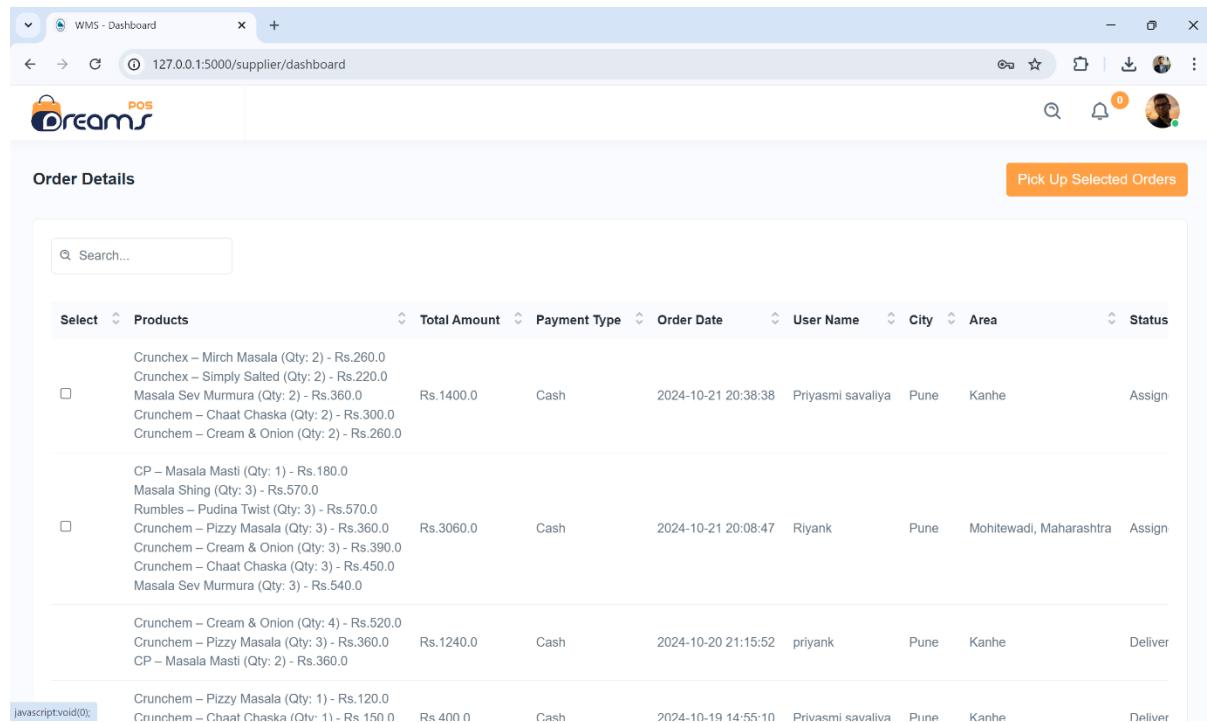


The screenshot shows the Employee Dashboard with the title "Task Details". The interface includes a search bar, user profile information (Priyank, employee), and a logout button. A table lists tasks assigned to employees (PM, Shubham, priyank, Riyank) with details like product names, quantities, areas, dates, and statuses (Delivered, Assigned). Action buttons like "Task Completed" and "Mark as Completed" are present.

Tasks ID	client name	Products	Date	Status	Action
1	PM	Masala Sev Murmura - Quantity: 5 - Area Name : A10 Masala Shing - Quantity: 5 - Area Name : A11 Rumbles - Pudina Twist - Quantity: 5 - Area Name : A6	2024-10-20 11:17:19.468000	Delivered	<button>Task Completed</button>
2	Shubham	Crunchem - Cream & Onion - Quantity: 3 - Area Name : A3 Masala Sev Murmura - Quantity: 4 - Area Name : A10	2024-10-21 20:21:33.014000	Assigned	<button>Mark as Completed</button>
3	priyank	Crunchem - Chaat Chaska - Quantity: 3 - Area Name : A4	2024-10-21 21:14:56.260000	Assigned	<button>Mark as Completed</button>
4	Riyank	CP – Masala Masti - Quantity: 1 - Area Name : A5 Masala Shing - Quantity: 3 - Area Name : A11 Rumbles - Pudina Twist - Quantity: 3 - Area Name : A6 Crunchem - Pizzi Masala - Quantity: 3 - Area Name : A2 Crunchem - Cream & Onion - Quantity: 3 - Area Name : A3 Crunchem - Chaat Chaska - Quantity: 3 - Area Name : A4 Masala Sev Murmura - Quantity: 3 - Area Name : A10	2024-10-21 20:08:47.484000	Assigned to Supplier	<button>Task Completed</button>
5	Shubham	Crunchem - Cream & Onion - Quantity: 3 - Area Name : A3	2024-10-21 21:08:14.869000	Assigned	<button>Mark as Completed</button>

### Employee Dashboard

### 4) Supplier:



The screenshot shows the Supplier Dashboard with the title "Order Details". The interface includes a search bar and a "Pick Up Selected Orders" button. A table lists orders with columns for Select, Products, Total Amount, Payment Type, Order Date, User Name, City, Area, and Status. Each row shows a list of items ordered by the supplier.

Select	Products	Total Amount	Payment Type	Order Date	User Name	City	Area	Status
<input type="checkbox"/>	Crunchex – Mirch Masala (Qty: 2) - Rs.260.0 Crunchex – Simply Salted (Qty: 2) - Rs.220.0 Masala Sev Murmura (Qty: 2) - Rs.360.0 Crunchem – Chaat Chaska (Qty: 2) - Rs.300.0 Crunchem – Cream & Onion (Qty: 2) - Rs.260.0	Rs.1400.0	Cash	2024-10-21 20:38:38	Priyansmi savalya	Pune	Kanhe	Assign
<input type="checkbox"/>	CP – Masala Masti (Qty: 1) - Rs.180.0 Masala Shing (Qty: 3) - Rs.570.0 Rumbles – Pudina Twist (Qty: 3) - Rs.520.0 Crunchem – Pizzi Masala (Qty: 3) - Rs.360.0 Crunchem – Cream & Onion (Qty: 3) - Rs.390.0 Crunchem – Chaat Chaska (Qty: 3) - Rs.450.0 Masala Sev Murmura (Qty: 3) - Rs.540.0	Rs.3060.0	Cash	2024-10-21 20:08:47	Riyank	Pune	Mohitwadi, Maharashtra	Assign
	Crunchem – Cream & Onion (Qty: 4) - Rs.520.0 Crunchem – Pizzi Masala (Qty: 3) - Rs.360.0 CP – Masala Masti (Qty: 2) - Rs.360.0	Rs.1240.0	Cash	2024-10-20 21:15:52	priyank	Pune	Kanhe	Deliver
	Crunchem – Pizzi Masala (Qty: 1) - Rs.120.0 Crunchem – Chaat Chaska (Qty: 1) - Rs.150.0	Rs.400.0	Cash	2024-10-19 14:55:10	Priyansmi savalya	Pune	Kanhe	Deliver

### Supplier Dashboard

# Warehouse Management System

The screenshot shows the WMS Supplier Dashboard. At the top, there is a modal window titled "Enter OTP for Order" with a text input field for "OTP" and a "Submit" button. Below the modal is a table of "Order Details". The table has columns for "Order ID", "Total Amount", "Payment Type", "Order Date", "Status", and "Actions". There are 10 rows of order data, each detailing a specific item purchase with its quantity, price, and delivery status (e.g., Delivered, Out for Delivery). The last row shows a total amount of Rs. 1600.0.

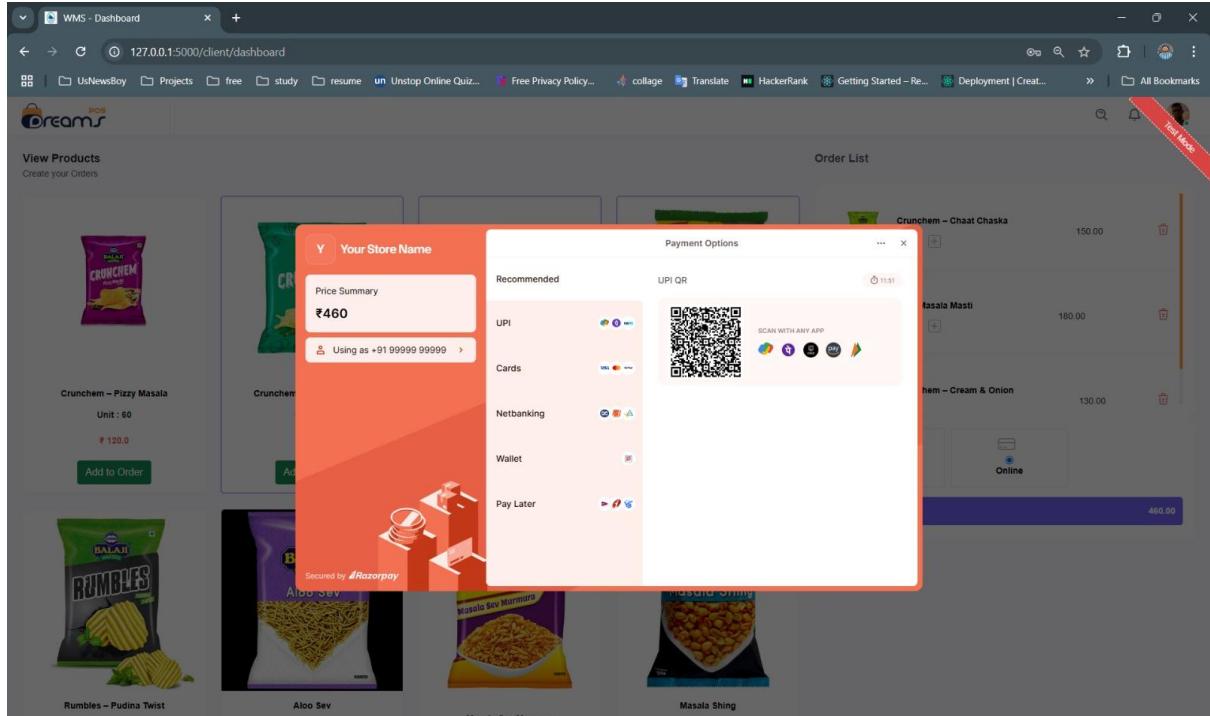
## Verify OTP

### 5) Client:

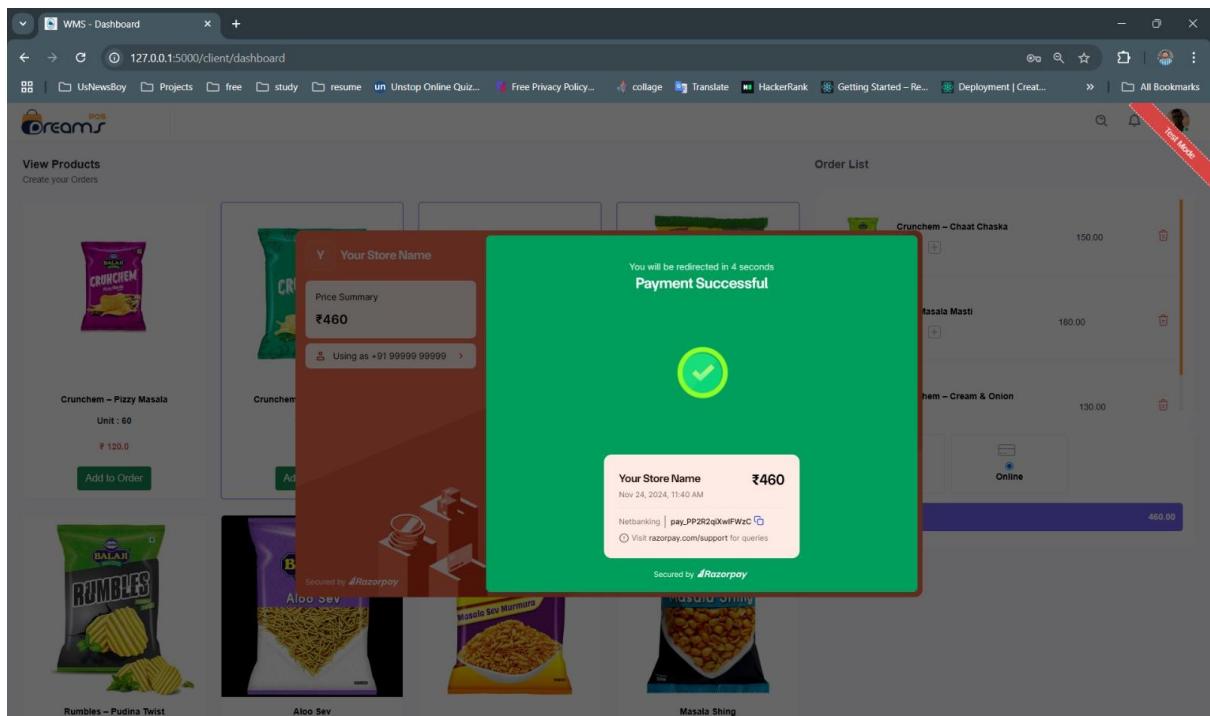
The screenshot shows the WMS Client Dashboard. On the left, there is a "View Products" section with a heading "Create your Orders". It displays five product cards: "Crunchem – Pizzy Masala" (Unit: 60, ₹ 120.0), "Crunchem – Cream & Onion" (Unit: 60, ₹ 130.0), "Crunchem – Chaat Chaska" (Unit: 40, ₹ 150.0), "CP – Masala Masti" (Unit: 30, ₹ 180.0), and "Aloo Sev" (Unit: 30, ₹ 150.0). Each card has an "Add to Order" button. On the right, there is an "Order List" section with a heading "Order List". It shows a list of items with quantities and unit prices: "CP – Masala Masti" (3 units, ₹ 180.0), "Crunchem – Chaat Chaska" (3 units, ₹ 150.0), and "Aloo Sev" (3 units, ₹ 150.0). Below the order list are buttons for "Cash" and "Online". At the bottom right is a "Checkout" button with a total amount of ₹ 1120.0.

## Client Dashboard

# Warehouse Management System



## Payment



## Payment Success

# Warehouse Management System

The screenshot shows the 'Order History List' section of the WMS. At the top, there's a search bar labeled 'Search...'. On the right, a sidebar displays a user profile for 'Shubham' (client), options to 'My Profile', 'Order History', and 'Logout'. The main area is a table with columns: S. No, Products, Order Date, Total Amount, Payment Type, Order Status, Payment Status, and View Status. The table lists six orders:

S. No	Products	Order Date	Total Amount	Payment Type	Order Status	Payment Status	View Status
1	Crunchem – Chaat Chaska (Qty: 5) - Rs.750.0 Crunchem – Cream & Onion (Qty: 2) - Rs.260.0 Rumbles – Pudina Twist (Qty: 2) - Rs.380.0	2024-10-20 18:33:16.482000	1390.0	Cash	Received	Paid	
2	Masala Sev Murmura (Qty: 10) - Rs.1800.0 Rumbles – Pudina Twist (Qty: 8) - Rs.1520.0 Crunchem – Pizzi Masala (Qty: 9) - Rs.1080.0	2024-10-20 18:59:54.079000	4400.0	Cash	Received	Paid	
3	Crunchem – Cream & Onion (Qty: 3) - Rs.390.0 Masala Sev Murmura (Qty: 4) - Rs.720.0	2024-10-21 20:21:32.907000	1110.0	Cash	Pending	Pending	
4	Crunchem – Pizzi Masala (Qty: 5) - Rs.600.0 Masala Shing (Qty: 1) - Rs.190.0 Masala Sev Murmura (Qty: 1) - Rs.180.0 Rumbles – Pudina Twist (Qty: 2) - Rs.380.0	2024-10-21 20:21:53.944000	1350.0	Cash	Pending	Pending	
5	Crunchem – Cream & Onion (Qty: 4) - Rs.520.0	2024-10-21 21:07:27.197000	520.0	Cash	Pending	Pending	
6	Crunchem – Cream & Onion (Qty: 3) - Rs.390.0	2024-10-21 21:08:14.788000	1170.0	Cash	Pending	Pending	

## View Order History

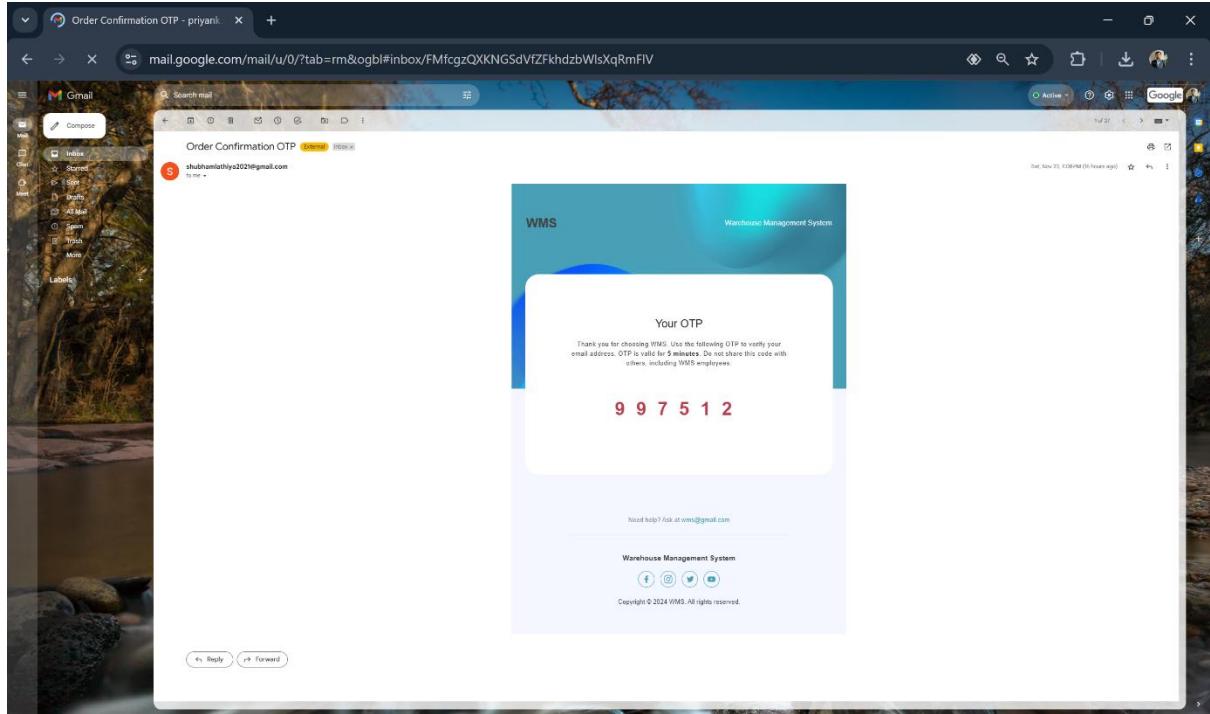
The screenshot shows the 'Timeline' section of the WMS. It displays a vertical timeline of order status updates with corresponding icons:

- Submitted (green checkmark icon)
- Assigned (orange circle icon)
- Shipment Ready (black checkmark icon)
- Assigned to Supplier (red circle icon)
- Out for Delivery (blue circle icon)
- Delivered (green checkmark icon)

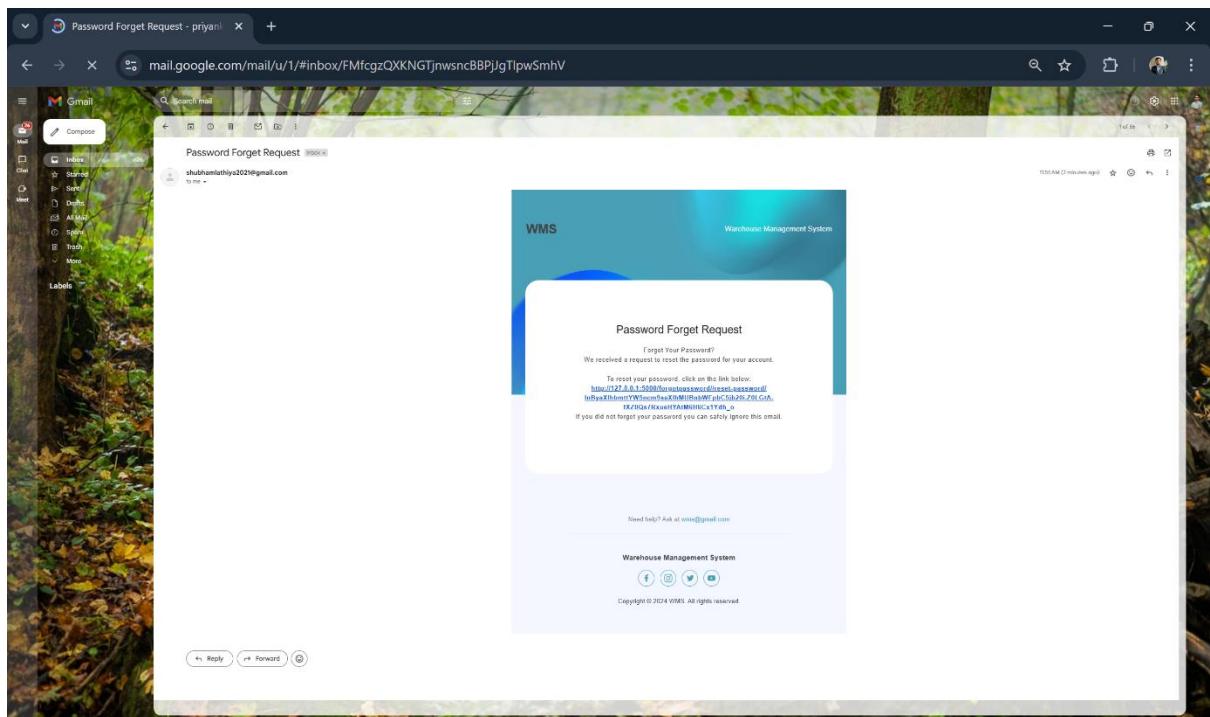
Each status update includes a timestamp: Status updated at: 2024-10-20 18:33:16, 2024-10-20 18:35:12, 2024-10-20 18:35:12, 2024-10-20 18:36:09, 2024-10-20 18:36:09, and Delivered at: 2024-10-20 18:36:57.

## View Timeline

# Warehouse Management System



## OTP Mail



## Reset Password Mail

## 6) Testing

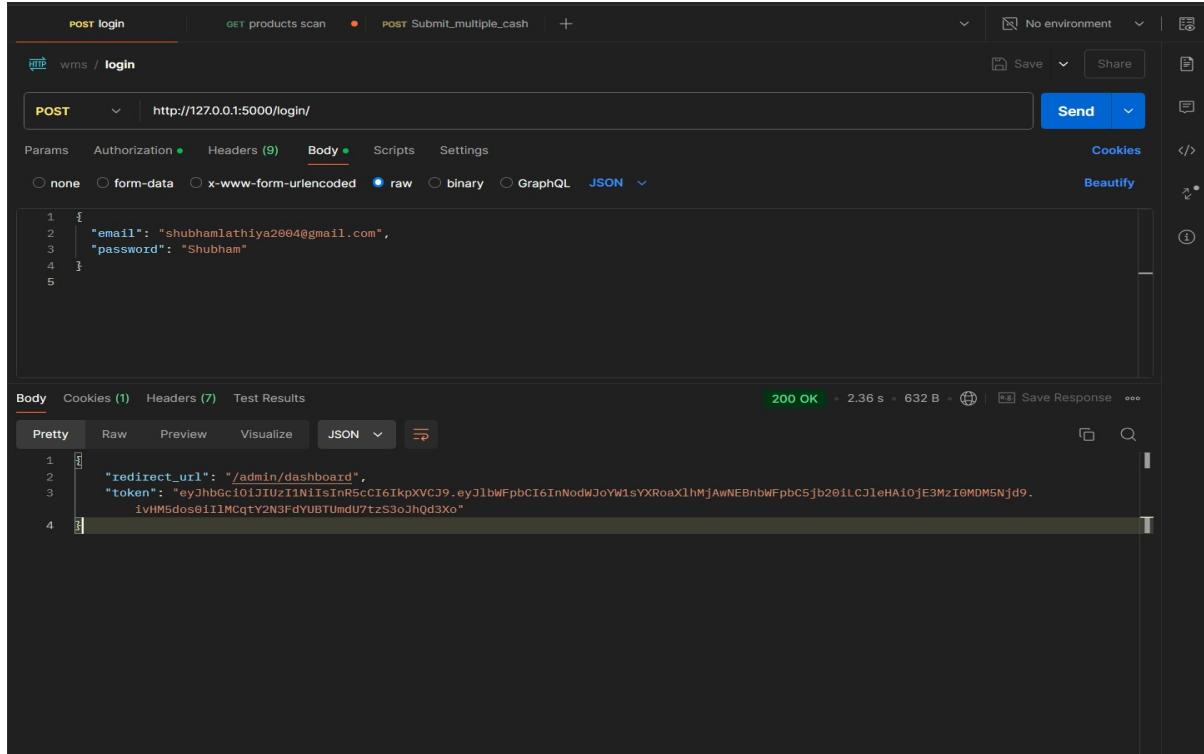
### 6.1) Test Cases and Testing Methodology

- 1) **Unit Testing:** Each module, such as inventory management, order processing, and user authentication, is tested independently to ensure they function correctly.
- 2) **Integration Testing:** Focuses on the interaction between modules like order placement by clients, updates by suppliers, and dashboard updates for managers.
- 3) **Functional Testing:** Ensures that the system meets all functional requirements, including user role-specific functionalities.
- 4) **Performance Testing:** Tests the system's responsiveness, speed, and scalability under varying loads, including high data volumes and simultaneous user interactions.
- 5) **Security Testing:** Verifies encryption, role-based access control, and protection against vulnerabilities like data breaches.
- 6) **Usability Testing:** Assesses the system's user interface and ease of use for all roles, ensuring it is intuitive and accessible.
- 7) **Regression Testing:** Re-tests existing functionality after updates or bug fixes to ensure no new issues are introduced.
- 8) **Acceptance Testing:** Final validation by end-users to ensure the system meets business needs and is ready for deployment.

## 6.2) Results of Testing

### 1) Automatic Testing:

Login:



```

POST /login HTTP/1.1
Host: http://127.0.0.1:5000/login/
Content-Type: application/json

{
  "email": "shubhamlathiya2004@gmail.com",
  "password": "Shubham"
}
  
```

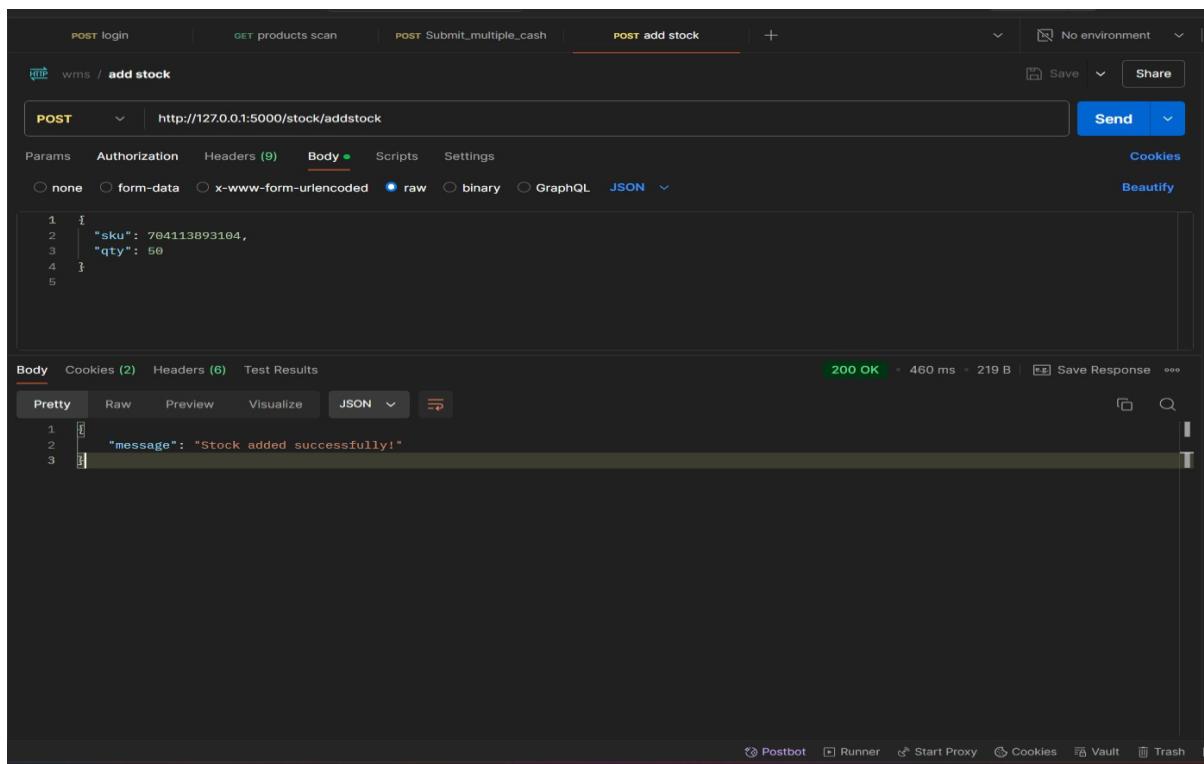
Body

```

1 {
2   "email": "shubhamlathiya2004@gmail.com",
3   "password": "Shubham"
4 }
  
```

200 OK

Add Stock:



```

POST /stock/addstock HTTP/1.1
Host: http://127.0.0.1:5000/stock/addstock
Content-Type: application/json

{
  "sku": "704113893104",
  "qty": 56
}
  
```

Body

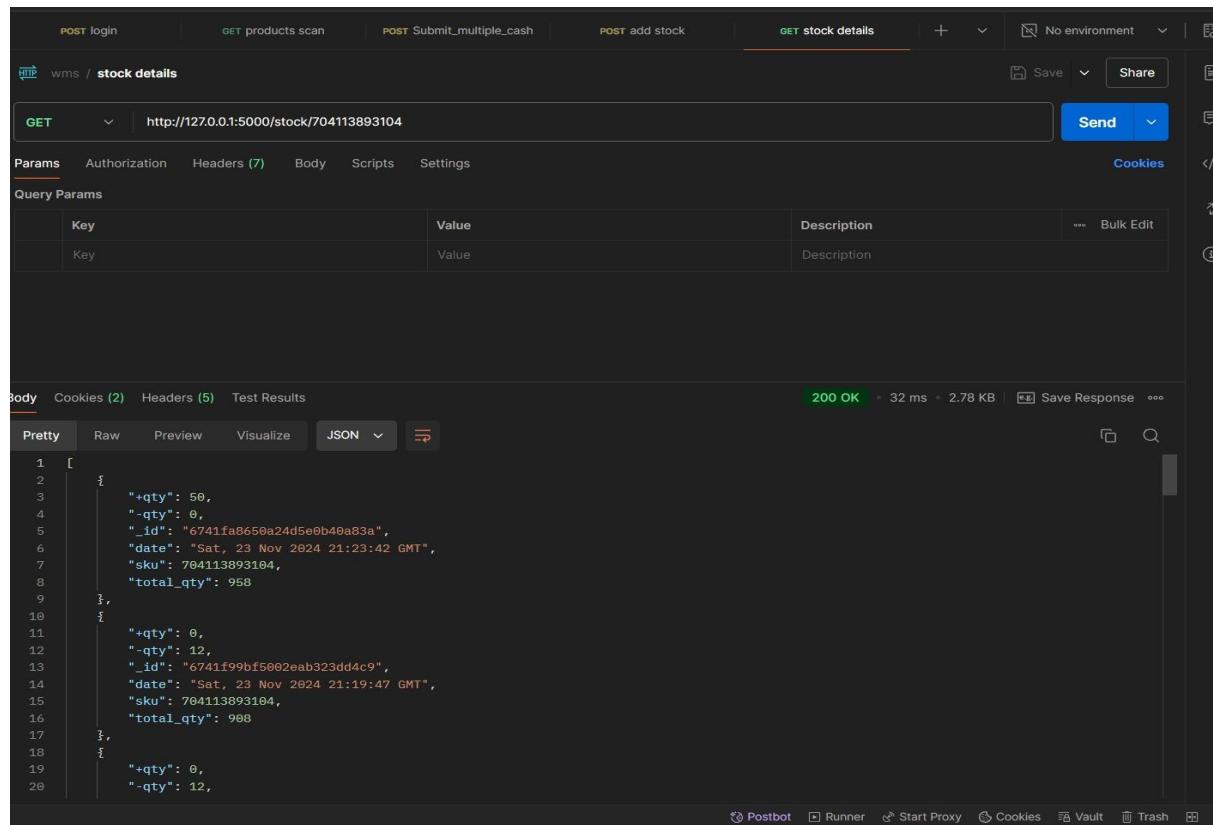
```

1 {
2   "sku": "704113893104",
3   "qty": 56
4 }
  
```

200 OK

# Warehouse Management System

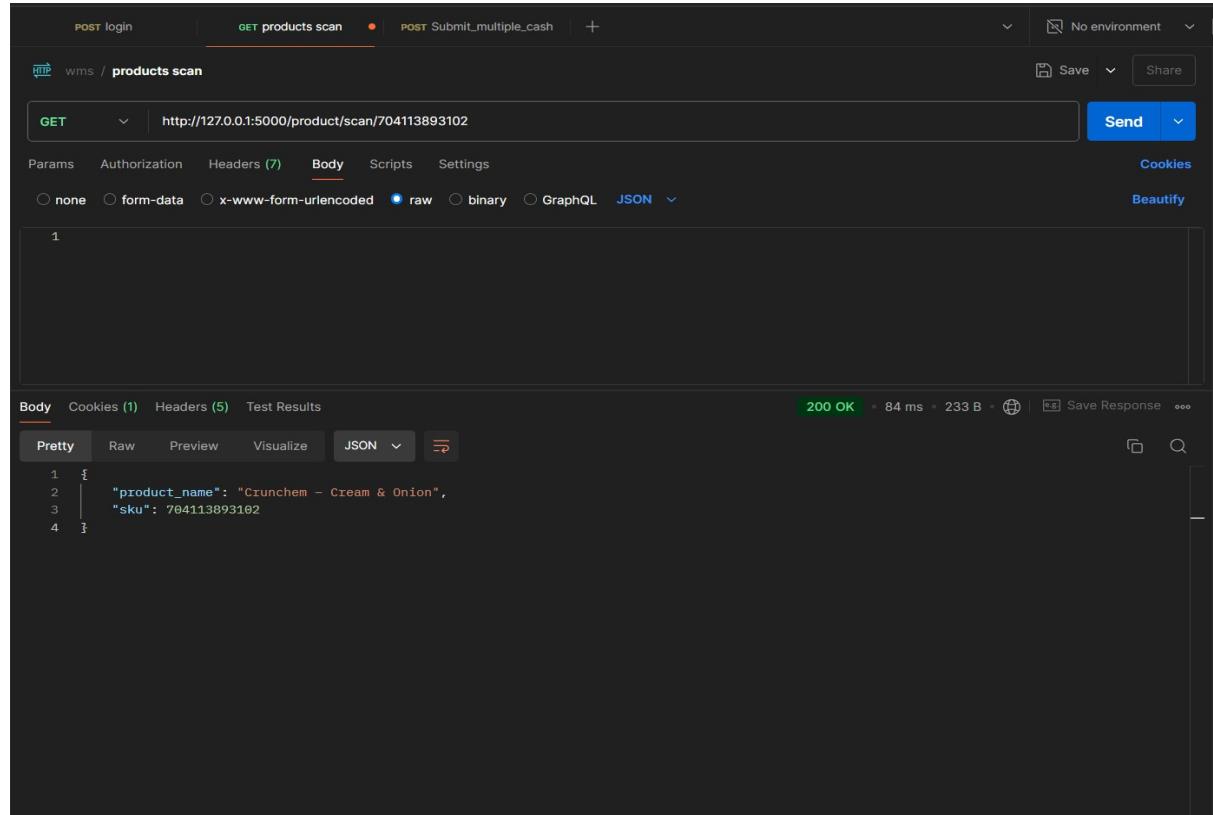
## Stock View:



The screenshot shows the Postman interface with a request to "GET stock details". The URL is `http://127.0.0.1:5000/stock/704113893104`. The response status is 200 OK, with a response body containing a JSON array of stock items.

```
1 [  
2   {  
3     "+qty": 50,  
4     "-qty": 0,  
5     "_id": "6741fa8650a24d5e0b40a83a",  
6     "date": "Sat, 23 Nov 2024 21:23:42 GMT",  
7     "sku": "704113893104",  
8     "total_qty": 958  
9   },  
10  {  
11    "+qty": 0,  
12    "-qty": 12,  
13    "_id": "6741f9bfe5002eab323dd4c9",  
14    "date": "Sat, 23 Nov 2024 21:19:47 GMT",  
15    "sku": "704113893104",  
16    "total_qty": 908  
17  },  
18  {  
19    "+qty": 0,  
20    "-qty": 12,
```

## Scan QR:



The screenshot shows the Postman interface with a request to "GET products scan". The URL is `http://127.0.0.1:5000/product/scan/704113893102`. The response status is 200 OK, with a response body containing a JSON object.

```
1 {  
2   "product_name": "Crunchem - Cream & Onion",  
3   "sku": "704113893102"  
4 }
```

**2) Manual Testing:**

Test Case ID	Test Case Name	Input Data	Expected Output	Actual Output	Test Status
TC_01	Retrieve stock by valid SKU	SKU: 1001 (valid SKU)	Stock details for SKU: 1001 with fields total_qty, +qty, -qty, and date. Example: { "sku": 1001, "total_qty": 100, "+qty": 50, "-qty": 0, "date": "2023-11-23" }	Stock details for SKU: 1001 returned as expected.	Pass
TC_02	Retrieve stock for non-existent SKU	SKU: 9999 (non-existent SKU)	Error message indicating "No stock found" for SKU 9999. Example: { "error": "No stock entries found" }.	Error message No stock entries found returned correctly.	Pass
TC_03	Test unauthorized access (no JWT token)	SKU: 1001, No JWT token	Error message indicating unauthorized access. Example: { "error": "Authentication required" }.	Error message Authentication required returned.	Pass
TC_04	Retrieve stock for SKU with multiple stock entries	SKU: 1001 (Multiple entries)	Multiple stock entries for SKU 1001, each showing total_qty, +qty, -qty, and date. Example: [ {	Multiple stock entries returned as expected.	Pass

			<pre>"sku": 1001, "total_qty": 100, "+qty": 50, "-qty": 0, "date": "2023-11-23" }, { "sku": 1001, "total_qty": 200, "+qty": 100, "-qty": 0, "date": "2023-11-24" } ]</pre>		
TC_05	Test stock retrieval with invalid SKU format	SKU: 'ABC' (invalid format)	Error message indicating invalid SKU format. Example: { "error": "Invalid SKU format" }.	Error message Invalid SKU format returned correctly.	Pass
TC_06	Test stock retrieval with missing SKU parameter	SKU: (empty)	Error message indicating SKU is required. Example: { "error": "SKU is required" }.	Error message SKU is required returned correctly.	Pass

### 6.3) Bug Fixes/Issues Resolved

#### 1) User Interface:

**Issue:** Some UI elements were not responsive on smaller screen sizes.

**Fix:** Updated CSS and JavaScript for better responsiveness and compatibility.

#### 2) Order Processing

**Issue:** Order status was not synchronizing correctly between the dashboard and client view.

**Fix:** Optimized database queries and improved data synchronization logic.

## 7) Results and Discussion

### 7.1) Outcomes of the Project

- **Admin:** Manage users, settings, integrations, and the whole system with real-time visibility, improved decision-making, and enhanced security.
- **Manager:** The Warehouse Manager ensures efficient warehouse performance by monitoring inventory, order processing, and workflows with real-time data access.
- **Employee:** Automates daily tasks like inventory updates, picking, and packing. Makes their tasks easier and more efficient by scanning barcodes and generating picking lists.
- **Client:** The system allows clients to place orders and track them in real time. They get order tracking at every step, order confirmation, picking, packing, and shipping.
- **Supplier:** The system enables suppliers to manage their orders efficiently by providing real-time order status updates, delivery tracking, and payment processing.

### 7.2) Comparison with existing solutions

- The proposed Warehouse Management System (WMS) offers several advantages over existing solutions. Unlike legacy systems, it provides real-time inventory tracking, scalability, and user-friendly interfaces. Cloud-based WMS solutions are more flexible but often come with high costs and security concerns, which this system addresses through cost-effective, open-source technologies and robust security features like AES-256 encryption.
- ERP-integrated WMS solutions offer unified operations but may be complex to implement, whereas our system focuses on simplicity and ease of use for quick adoption. Additionally, this WMS is more customizable and adaptable compared to many industry-specific solutions, offering a flexible platform without high implementation costs.

### 7.3) Challenges Faced

- During the development of the Warehouse Management System (WMS), several challenges were encountered. One major challenge was ensuring seamless real-time inventory tracking and accurate order processing, which required robust backend architecture and efficient database management. Integrating security features like role-based access control posed additional complexity to ensure data protection and prevent unauthorized access.
- The development also faced challenges in creating an intuitive user interface that could be easily adopted by warehouse staff with minimal training. Furthermore, scalability and ensuring the system could handle increased data loads as the warehouse grows required careful planning. Addressing these challenges required a balance between performance, security, and user experience to create a system that is both functional and efficient.

## 8) Conclusion and Future Scope

### 8.1) Summary of the work

- This project focuses on designing and developing a Warehouse Management System (WMS) to streamline warehouse operations by addressing common challenges like inaccurate inventory tracking, inefficient space utilization, and slow order processing.
- The WMS employs an Agile development methodology and a technology stack comprising Python (Flask) for backend, MongoDB for scalable storage, and a responsive frontend built with HTML, CSS, and JavaScript. Core functionalities include real-time inventory tracking, automated order management, role-based user access, and analytics reporting. Key user roles—admin, manager, employee, supplier, and client—have distinct functionalities, such as managing stock, processing orders, tracking deliveries, and viewing reports. Non-functional requirements like security, email notifications, and 99.9% availability are prioritized.

### 8.2) Limitations

- **No Multi-Warehouse Support:** Features like multi-warehouse management are excluded.
- **No Returns Processing:** The system does not include return or replacement functionality.
- **Limited Integrations:** Advanced integrations with external systems, such as ERP platforms, are not supported.

### 8.3) Future Enhancements

- **Multi-Warehouse Management:** Extend functionality to support multiple warehouses.
- **Returns Processing:** Add features for handling product returns and replacements.

- **Advanced Integrations:** Enable integration with ERP systems and third-party logistics providers.
- **AI and IoT Integration:** Incorporate AI for demand forecasting and IoT for real-time inventory tracking.
- **Mobile App:** Develop a mobile application for on-the-go access and warehouse task management.
- **Customizable Dashboards:** Allow users to personalize dashboard views and reports.

## References

- 1) <https://ieeexplore.ieee.org/document/278253>
- 2) [https://dspmuranchi.ac.in/pdf/Blog/srs\\_template-ieee.pdf](https://dspmuranchi.ac.in/pdf/Blog/srs_template-ieee.pdf)
- 3) <https://www.zoho.com/in/inventory/>
- 4) <https://www.kladana.com/product-tour/inventory-management/>