EX NO:1	INVENTORY MANAGEMENT SYSTEM
DATE:	

AIM:

To prepare PROBLEM STATEMENT for any inventory management systems.

ALGORITHM:

1. Initialize System and Load Inventory Database:
   - Load inventory data, including item details, stock levels, and reorder thresholds.
2. User Login:
   - Allow users to log in to their account using credentials.
3. Inventory Search:
   - Users search for items by name, category, or SKU.
   - The system displays real-time stock status and location details.
4. Inventory Check and Update:
   - Users can check stock levels, and, if required, update quantities manually (with permissions).
   - The system records all transactions and updates stock levels in real-time.
5. Automated Reordering:
   - The system monitors stock levels and initiates a reorder if stock falls below a defined threshold.
   - Notifications for reorder are sent to authorized personnel for approval.
6. Online Ordering Process:
   - Authorized users can place orders for restocking online.
   - The system validates order details and updates stock upon arrival.
7. Notifications and Alerts:
   - Send automated notifications for low stock, successful reorder, and stock arrivals.
8. Stock In/Out Transactions:
   - Record transactions each time stock is issued or received.
   - Update inventory levels accordingly and maintain transaction logs.
9. Reporting:
   - Generate reports on inventory usage, reorder history, stock levels, and system performance.

INPUT:

1. User Details:
   - Username and password for login.
   - Contact information for notifications.
2. Item Information:
   - Item name, SKU, category, reorder threshold, and availability status.
3. Transaction Details:
   - Item issue date, restocking date, quantity adjustments, and reorder requests.
4. Reorder Requests:
   - User-initiated or automated requests to reorder low-stock items.
5. Notifications and Alerts:
   - Inputs for automated notifications, such as low-stock alerts and reorder confirmations.

Problem:

Current inventory management systems lack automation and online functionality, making it difficult for users to track stock levels, place restocking orders, and access inventory information efficiently. Manual processes result in delays, inaccuracies, and a subpar user experience, including stockouts and order errors.

Background:

The inventory team at [Organization Name] is facing issues with outdated processes that are neither user-friendly nor efficient. Staff members are frustrated by manual tracking and ordering, limited online capabilities, and lack of real-time stock information. These inefficiencies impact productivity and user satisfaction.

Relevance:

An efficient inventory management system is essential for smooth operations. Providing real-time stock updates, automated reorder options, and timely notifications will enhance the user experience, save time, and reduce errors, making resources more accessible and manageable..

Objectives:

The primary objective of this project is to develop a modern parking management system that enhances efficiency, transparency, and user satisfaction. The specific objectives include:

1. Automate stock tracking, issuing, and restocking processes.
2. Enable online ordering for efficient inventory management.
3. Provide real-time updates on stock levels.
4. Send automated alerts for low stock and reorder confirmations.
5. Improve overall inventory management and user satisfaction.

Result:

EX NO:2	**WRITE THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT**
DATE:	

AIM:

To do requirement analysis and develop Software Requirement Specification Sheet(SRS) for inventory management system.

ALGORITHM for Smart Inventory Management System:

1. Initialize System and Load Inventory Database:

- Load inventory data, including item details, stock levels, and reorder thresholds.

2. User Login:

- Allow users to log in to their account using credentials.

3. Inventory Search:

- Users search for items by name, category, or SKU.

- The system displays real-time stock status and location details.

4. Inventory Check and Update:

- Users can check stock levels, and, if required, update quantities manually (with permissions).

- The system records all transactions and updates stock levels in real-time.

5. Automated Reordering:

- The system monitors stock levels and initiates a reorder if stock falls below a defined threshold.

- Notifications for reorder are sent to authorized personnel for approval.

6. Online Ordering Process:

- Authorized users can place orders for restocking online.

- The system validates order details and updates stock upon arrival.

7. Notifications and Alerts:

- Send automated notifications for low stock, successful reorder, and stock arrivals.

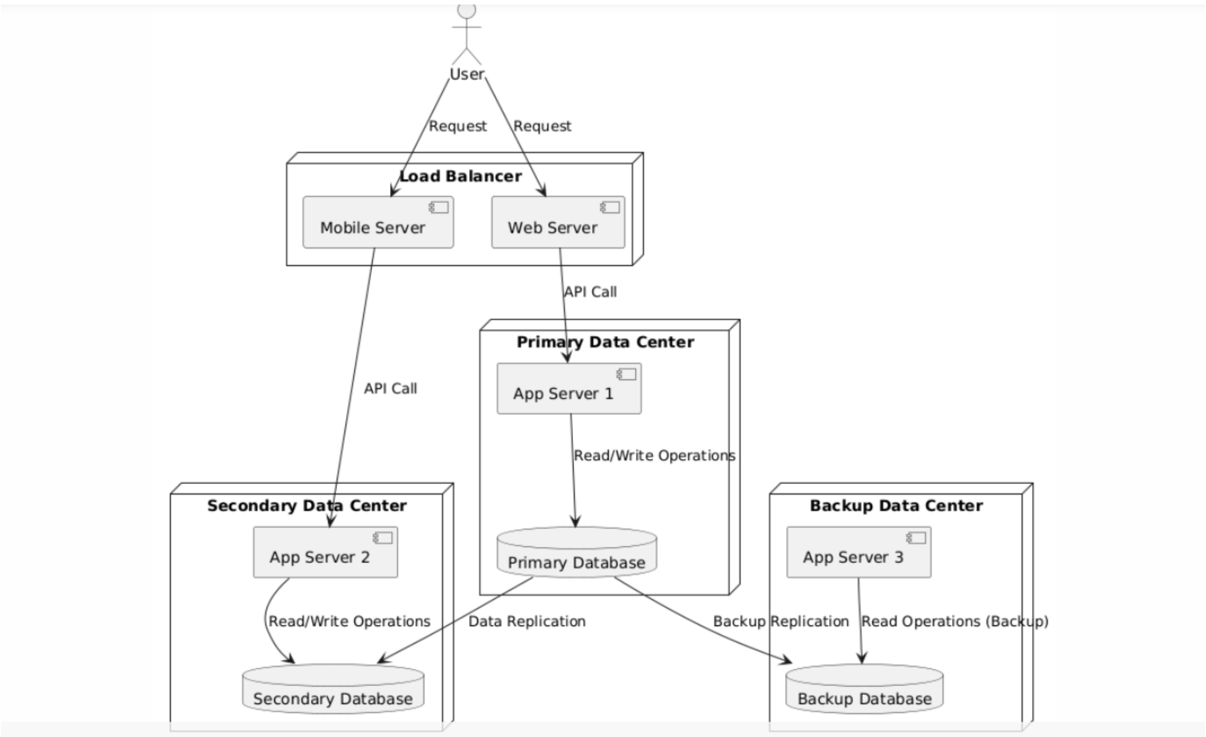
8. Stock In/Out Transactions:

- Record transactions each time stock is issued or received.

- Update inventory levels accordingly and maintain transaction logs.

9. Reporting:

- Generate reports on inventory usage, reorder history, stock levels, and system performance.


User
Request
Request
Load Balancer
Mobile Server
Web Server
API Call
API Call
Primary Data Center
App Server 1
Read/Write Operations
Primary Database
Secondary Data Center
App Server 2
Read/Write Operations
Secondary Database
Data Replication
Backup Replication
Backup Data Center
App Server 3
Read Operations (Backup)
Backup Database

INPUTS FOR THE SMART INVENTORY MANAGEMENT SYSTEM:

1. User Details:
   - Username and password for login.
   - Contact information for notifications.

2. Item Information:
   - Item name, SKU, category, reorder threshold, and availability status.

3. Transaction Details:
   - Item issue date, restocking date, quantity adjustments, and reorder requests.

4. Reorder Requests:
   - User-initiated or automated requests to reorder low-stock items.

5. Notifications and Alerts:
   - Inputs for automated notifications, such as low-stock alerts and reorder confirmations.

These components work together to create a streamlined, user-friendly inventory management system.

Stakeholder Problem Statement: Smart Inventory Management System

Problem:

Current inventory management systems lack automation and online functionality, making it difficult for users to track stock levels, place restocking orders, and access inventory information efficiently. Manual processes result in delays, inaccuracies, and a subpar user experience, including stockouts and order errors.

Background:

The inventory team at [Organization Name] is facing issues with outdated processes that are neither user-friendly nor efficient. Staff members are frustrated by manual tracking and ordering, limited online capabilities, and lack of real-time stock information. These inefficiencies impact productivity and user satisfaction.

Relevance:

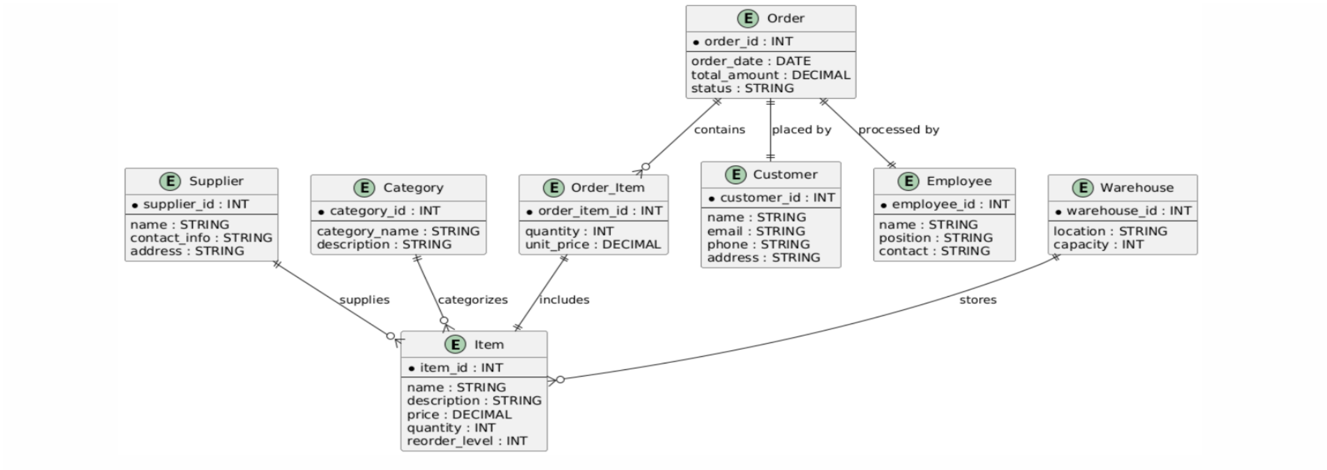
An efficient inventory management system is essential for smooth operations. Providing real-time stock updates, automated reorder options, and timely notifications will enhance the user experience, save time, and reduce errors, making resources more accessible and manageable.

Objectives:

1. Automate stock tracking, issuing, and restocking processes.
2. Enable online ordering for efficient inventory management.
3. Provide real-time updates on stock levels.
4. Send automated alerts for low stock and reorder confirmations.
5. Improve overall inventory management and user satisfaction.

Result:

The Smart Inventory Management System will optimize operations, boost user satisfaction, and improve resource management by providing real-time availability, automated reorder functions, and efficient tracking, resulting in a better experience for staff, users, and management.


E Order
• order_id : INT
order_date : DATE
total_amount : DECIMAL
status : STRING
contains
placed by
processed by
E Supplier
• supplier_id : INT
name : STRING
contact_info : STRING
address : STRING
E Category
• category_id : INT
category_name : STRING
description : STRING
E Order_Item
• order_item_id : INT
quantity : INT
unit_price : DECIMAL
E Customer
• customer_id : INT
name : STRING
email : STRING
phone : STRING
address : STRING
E Employee
• employee_id : INT
name : STRING
position : STRING
contact : STRING
E Warehouse
• warehouse_id : INT
location : STRING
capacity : INT
supplies
categorizes
includes
stores
E Item
• item_id : INT
name : STRING
description : STRING
price : DECIMAL
quantity : INT
reorder_level : INT

EX NO:3	**DRAW THE ENTITY RELATIONSHIP DIAGRAM**
DATE:	

AIM:

To Draw the Entity Relationship Diagram for inventory management system.

ALGORITHM:

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relation Types

Step 4: Mapping of Binary 1:N Relationship Types.

Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

INPUT:

Entities

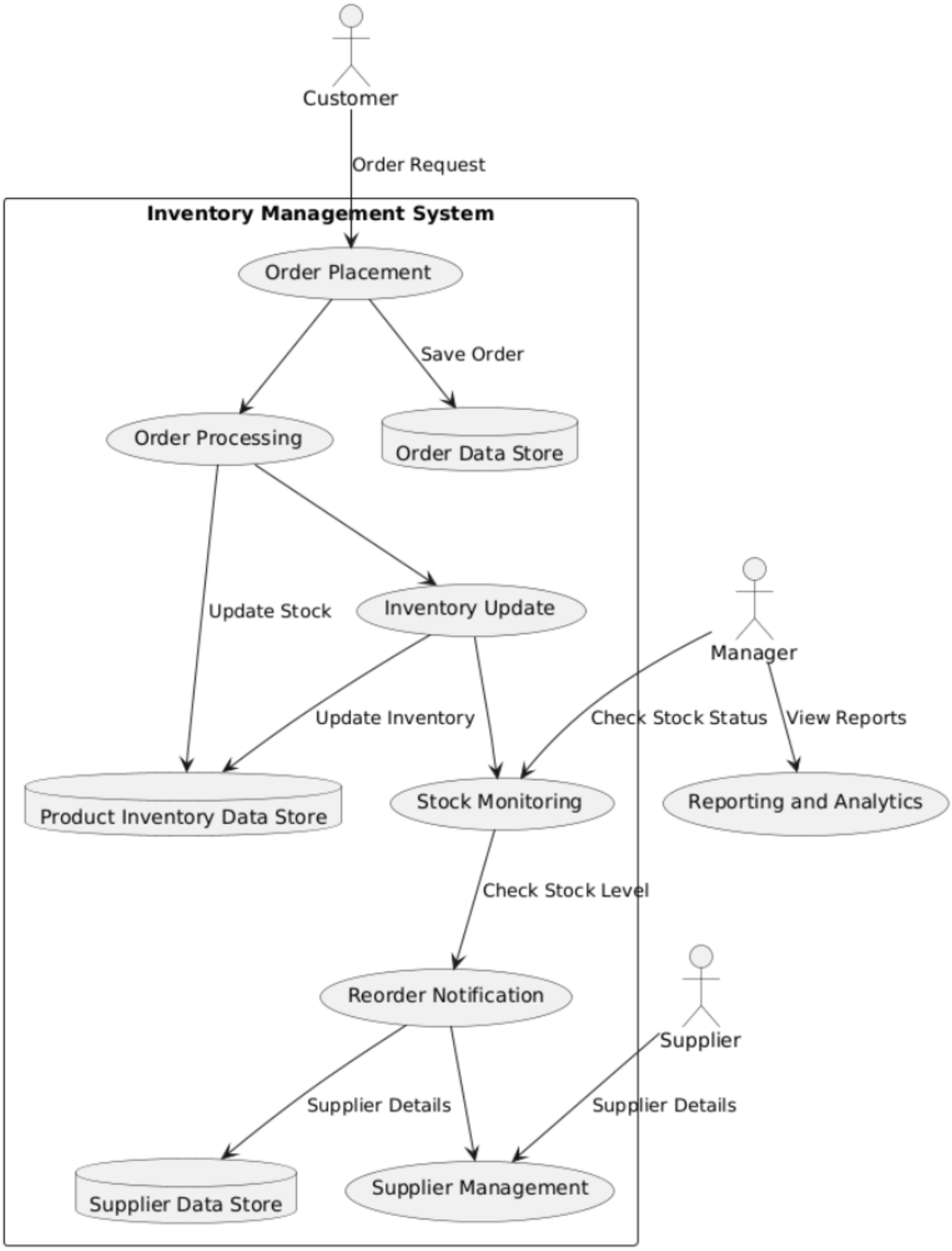
Entity Relationship Matrix

Primary Keys

Attributes

Mapping of Attributes with Entities

Result:


Customer
Order Request
Inventory Management System
Order Placement
Save Order
Order Processing
Order Data Store
Update Stock
Inventory Update
Manager
Update Inventory
Check Stock Status
View Reports
Product Inventory Data Store
Stock Monitoring
Reporting and Analytics
Check Stock Level
Reorder Notification
Supplier
Supplier Details
Supplier Details
Supplier Data Store
Supplier Management

EX NO:4	**DRAW THE DATA FLOW DIAGRAMS AT LEVEL 0 AND LEVEL 1**
DATE:	

AIM:

To Draw the Data Flow Diagram for inventory management system and List the Modules in the Application.

ALGORITHM:

1. Open the Visual Paradigm to draw DFD (Ex.Lucidchart)
2. Select a data flow diagram template
3. Name the data flow diagram
4. Add an external entity that starts the process
5. Add a Process to the DFD
6. Add a data store to the diagram
7. Continue to add items to the DFD
8. Add data flow to the DFD
9. Name the data flow
10. Customize the DFD with colours and fonts
11. Add a title and share your data flow diagram

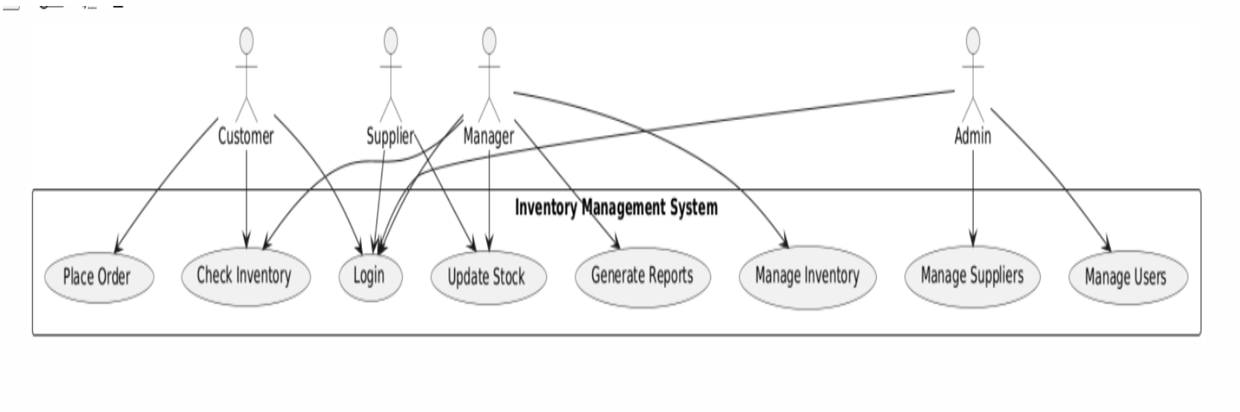
INPUT:

Processes

Datastores

External Entities

Result:


Customer
Supplier
Manager
Admin
Inventory Management System
Place Order
Check Inventory
Login
Update Stock
Generate Reports
Manage Inventory
Manage Suppliers
Manage Users

EX NO:5	**DRAW USE CASE DIAGRAM**
DATE:	

AIM:

To Draw the Use Case Diagram for inventory management system

ALGORITHM:

Step 1: Identify Actors

Step 2: Identify Use Cases

Step 3: Connect Actors and Use Cases

Step 4: Add System Boundary

Step 5: Define Relationships

Step 6: Review and Refine

Step 7: Validate

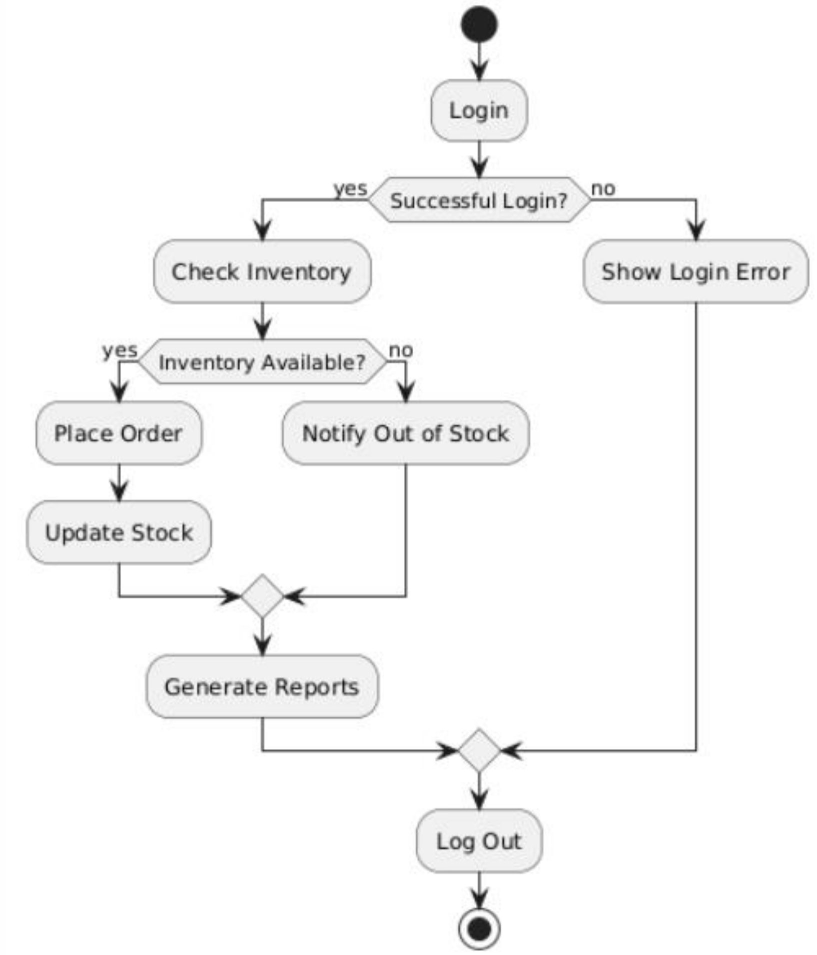
INPUTS:

Actors

Use Cases

Relations

Result:


Login
Successful Login?
yes
no
Check Inventory
Show Login Error
Inventory Available?
yes
no
Place Order
Notify Out of Stock
Update Stock
Generate Reports
Log Out

EX NO:6	**DRAW ACTIVITY DIAGRAM OF ALL USE CASES.**
DATE:	

AIM:

To Draw the activity Diagram for inventory management system

ALGORITHM:

Step 1: Identify the Initial State and Final States

Step 2: Identify the Intermediate Activities Needed

Step 3: Identify the Conditions or Constraints

Step 4: Draw the Diagram with Appropriate Notations

INPUTS:

Activities

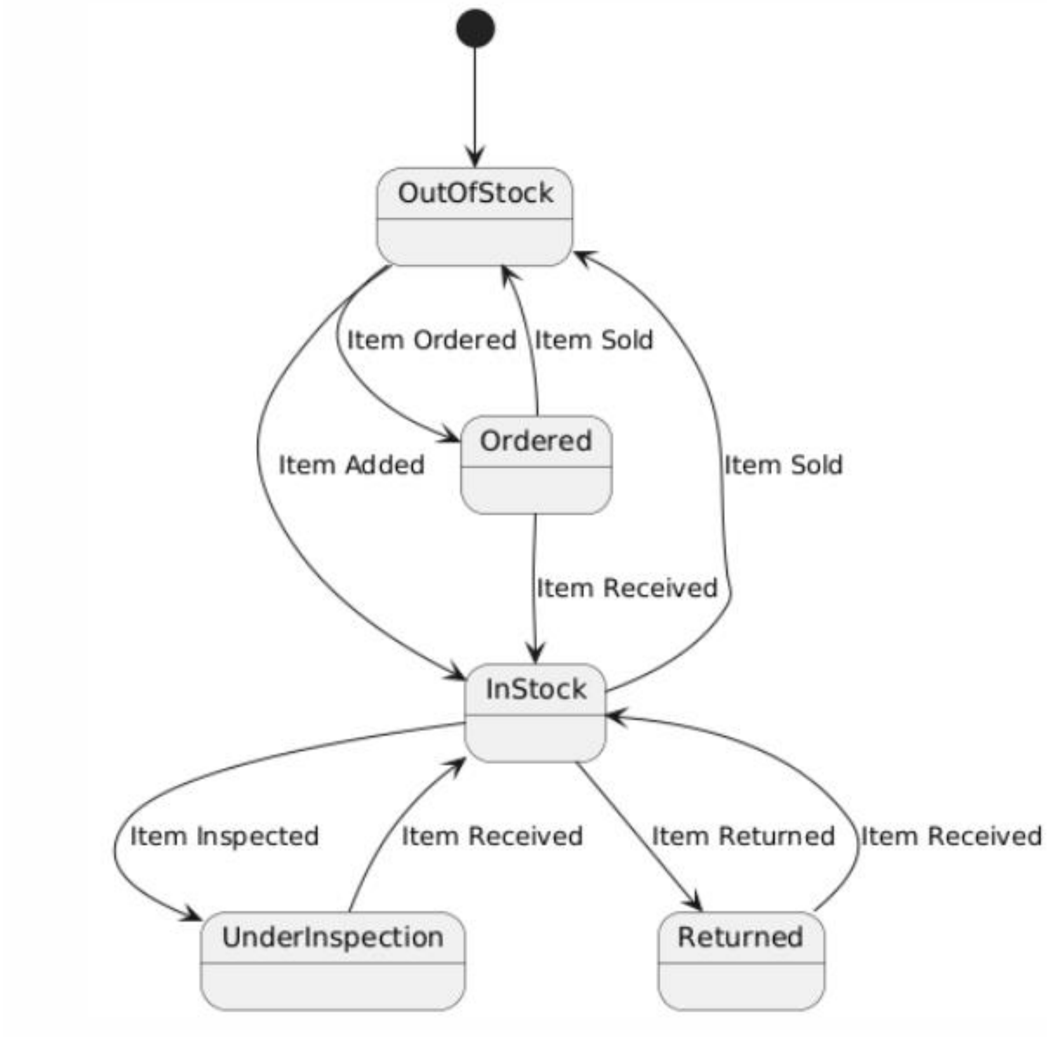
Decision Points

Guards

Parallel Activities

Conditions

Result:


OutOfStock
Item Ordered
Item Sold
Ordered
Item Added
Item Sold
Item Received
InStock
Item Inspected
Item Received
Item Returned
Item Received
UnderInspection
Returned

EX NO:7	**DRAW STATE CHART DIAGRAM OF ALL USE CASES.**
DATE:	

AIM:

To Draw the State Chart Diagram for inventory management system

ALGORITHM:

STEP-1: Identify the important objects to be analysed.

STEP-2: Identify the states.

STEP-3: Identify the events.

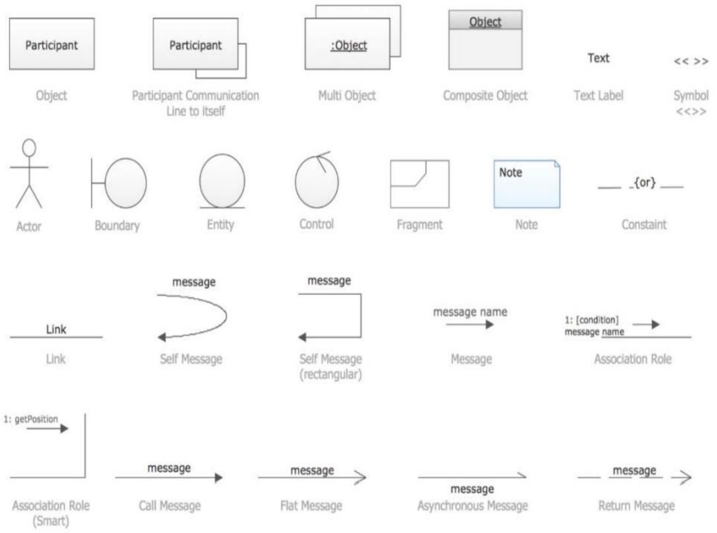
INPUTS:

Objects

States

Events

Result:


Participant
Object
Participant
Participant Communication Line to itself
:Object
Multi Object
Object
Composite Object
Text
Text Label
<< >>
Symbol <<>>
Actor
Boundary
Entity
Control
Fragment
Note
Note
{or}
Constaint
Link
Link
message
Self Message
message
Self Message (rectangular)
message name
Message
1: [condition] message name
Association Role
1: getPosition
Association Role (Smart)
message
Call Message
message
Flat Message
message
Asynchronous Message
message
Return Message

EX NO:8	**DRAW SEQUENCE DIAGRAM OF ALL USE CASES.**
DATE:	

AIM: To Draw the Sequence Diagram for inventory management system

ALGORITHM:

1. Identify the Scenario

2. List the Participants

3. Define Lifelines

4. Arrange Lifelines

5. Add Activation Bars

6. Draw Messages

7. Include Return Messages

8. Indicate Timing and Order

9. Include Conditions and Loops

10. Consider Parallel Execution

11. Review and Refine

12. Add Annotations and Comments

13. Document Assumptions and Constraints

14. Use a Tool to create a neat sequence diagram

INPUTS:

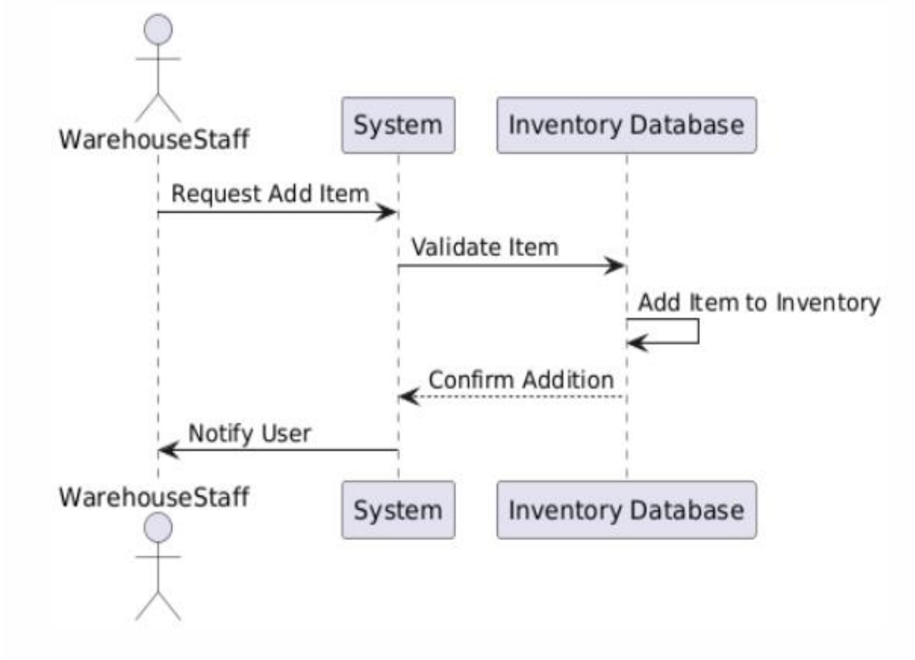
Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:


WarehouseStaff
System
Inventory Database
Request Add Item
Validate Item
Add Item to Inventory
Confirm Addition
Notify User
WarehouseStaff
System
Inventory Database

EX NO:9	**DRAW COLLABORATION DIAGRAM OF ALL USE CASES**
DATE:	

AIM:

To Draw the Collaboration Diagram for inventory management system

ALGORITHM:

Step 1: Identify Objects/Participants

Step 2: Define Interactions

Step 3: Add Messages

Step 4: Consider Relationships

Step 5: Document the collaboration diagram along with any relevant

explanations or annotations.

INPUTS:

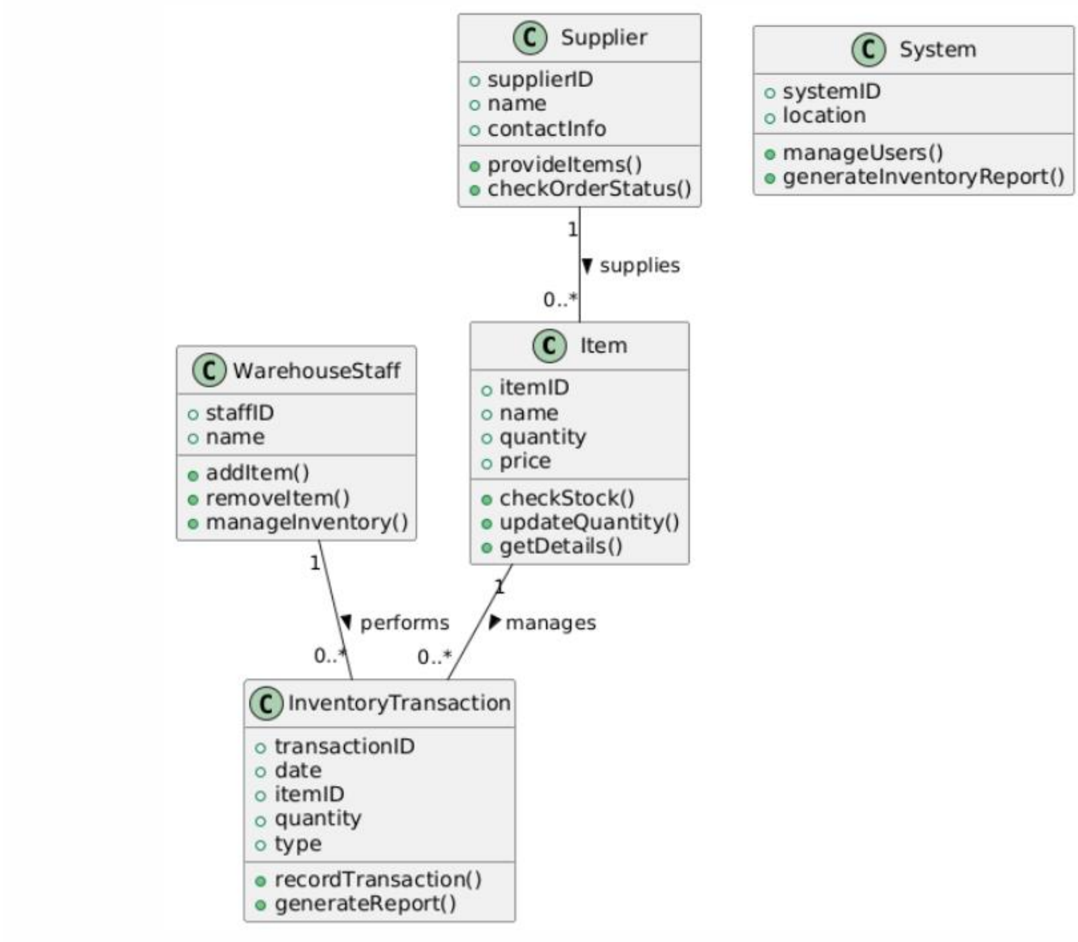
Objects taking part in the interaction.

Message flows among the objects.

The sequence in which the messages are flowing.

Object organization.

Result:


Supplier
supplierID
name
contactInfo
provideItems()
checkOrderStatus()
System
systemID
location
manageUsers()
generateInventoryReport()
1
supplies
0..*
WarehouseStaff
staffID
name
addItem()
removeItem()
manageInventory()
Item
itemID
name
quantity
price
checkStock()
updateQuantity()
getDetails()
1
performs
0..*
1
manages
0..*
InventoryTransaction
transactionID
date
itemID
quantity
type
recordTransaction()
generateReport()

EX NO:10	**ASSIGN OBJECTS IN SEQUENCE DIAGRAM TO CLASSES AND MAKE CLASS DIAGRAM.**
DATE:	

AIM:

To Draw the Class Diagram for inventory management system

ALGORITHM:

1. Identify Classes
2. List Attributes and Methods
3. Identify Relationships
4. Create Class Boxes
5. Add Attributes and Methods
6. Draw Relationships
7. Label Relationships
8. Review and Refine
9. Use Tools for Digital Drawing

INPUTS:

1. Class Name
2. Attributes
3. Methods
4. Visibility Notation

RESULT:

EX NO:11	MINI PROJECT FOR INVENTORY MANAGEMENT SYSTEM
DATE:	

Aim

To develop a user-friendly Inventory Management System (IMS) using Streamlit to manage stock levels, track sales, add/remove products, and generate reports. The goal is to create a simple interface that helps businesses maintain inventory accuracy, minimize errors, and ensure efficient stock management.

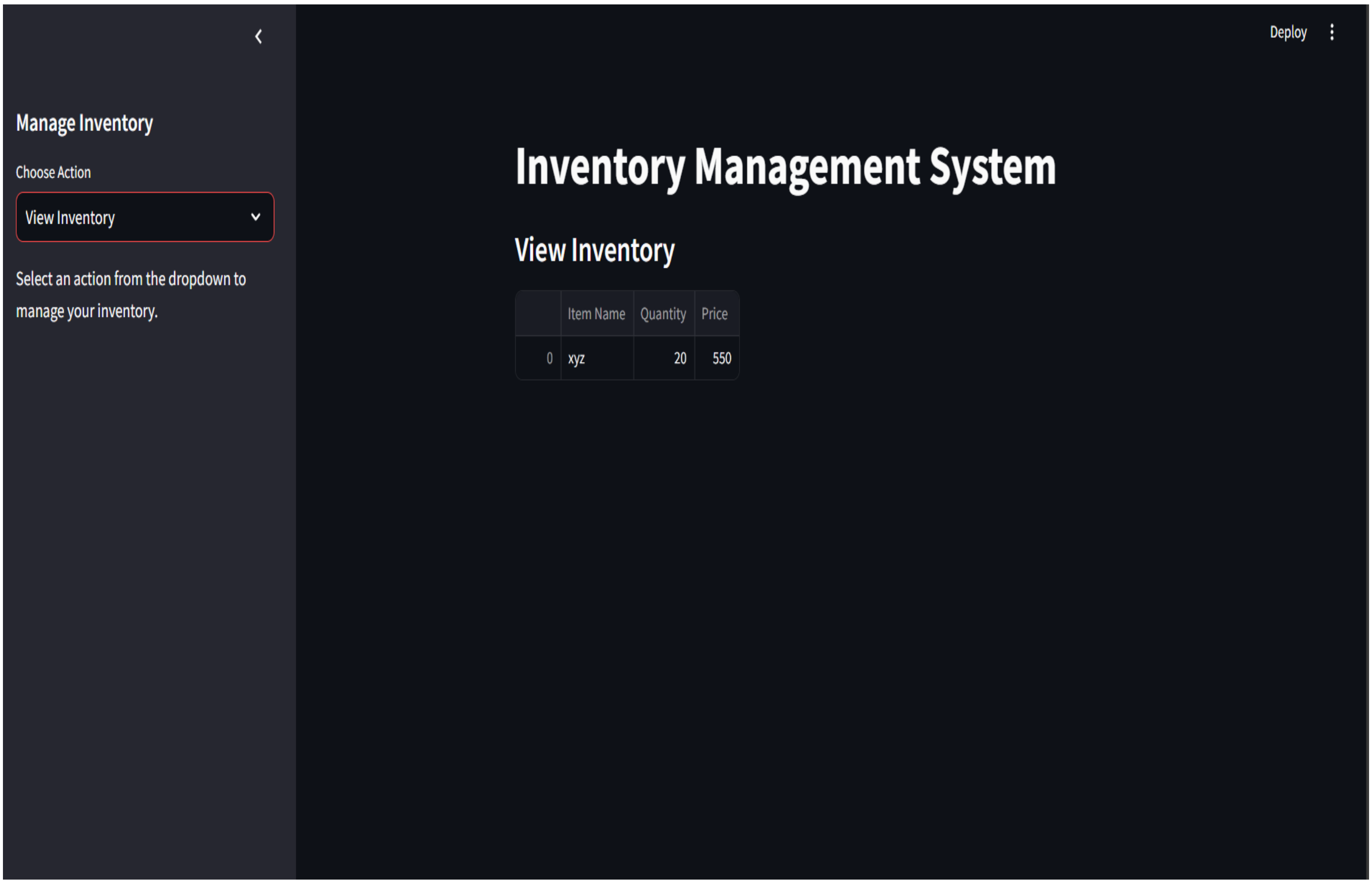
Algorithm

Here's a high-level algorithm for the IMS:

1. **Initialize**:
   - Import necessary libraries.
   - Set up Streamlit interface.
   - Initialize data structures (like a Pandas DataFrame) to store inventory data.
2. **Load/Save Inventory Data**:
   - If there's existing data, load it (from a file like CSV).
   - Create input fields for adding new items (Product Name, Quantity, Price).
   - Create buttons for adding and removing products.
3. **CRUD Operations**:
   - **Create**: Add new products to inventory.
   - **Read**: Display the inventory list.
   - **Update**: Modify product details (like changing quantities or prices).
   - **Delete**: Remove products from inventory.
4. **Search & Filtering**:
   - Implement search functionality to find items.
   - Filter based on specific criteria (like category, stock level).
5. **Sales & Restock**:
   - Create a form to handle sales (deduct items from inventory).
   - Implement restock functionality to increase stock levels.
6. **Reporting**:
   - Generate and display inventory status reports.
   - Include visual elements (charts/graphs) for data insights.
7. **Save Changes**:
   - Save updated inventory data to a file.
   - Ensure persistence across sessions.

Program:

```
import streamlit as st

import pandas as pd


# Initialize or load inventory data

inventory_file = 'inventory.csv'


@st.cache(allow_output_mutation=True)
```



Manage Inventory

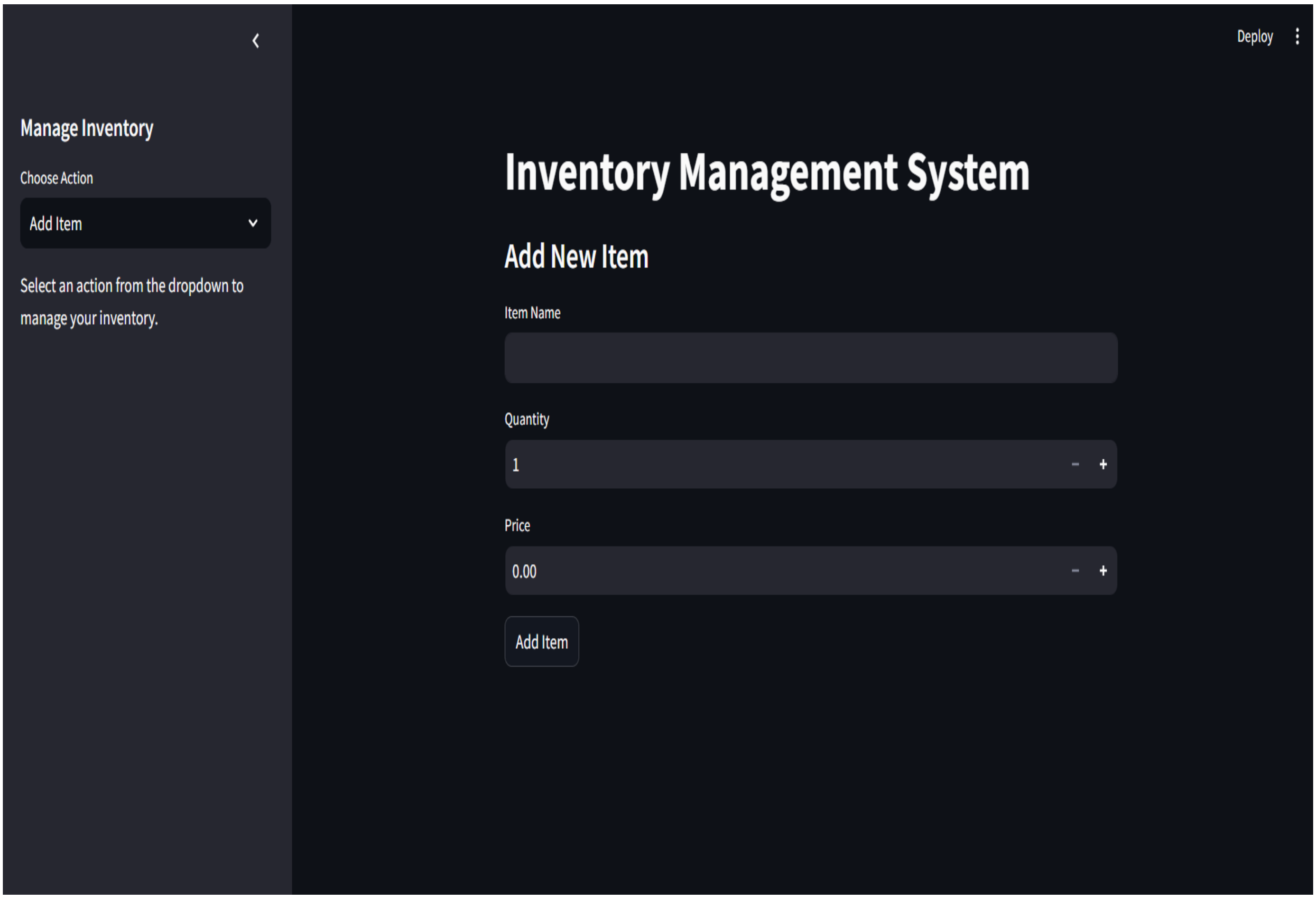
Choose Action: View Inventory

Select an action from the dropdown to manage your inventory.

Inventory Management System

View Inventory

	Item Name	Quantity	Price
0	xyz	20	550



Manage Inventory

Choose Action: Add Item

Select an action from the dropdown to manage your inventory.

Inventory Management System

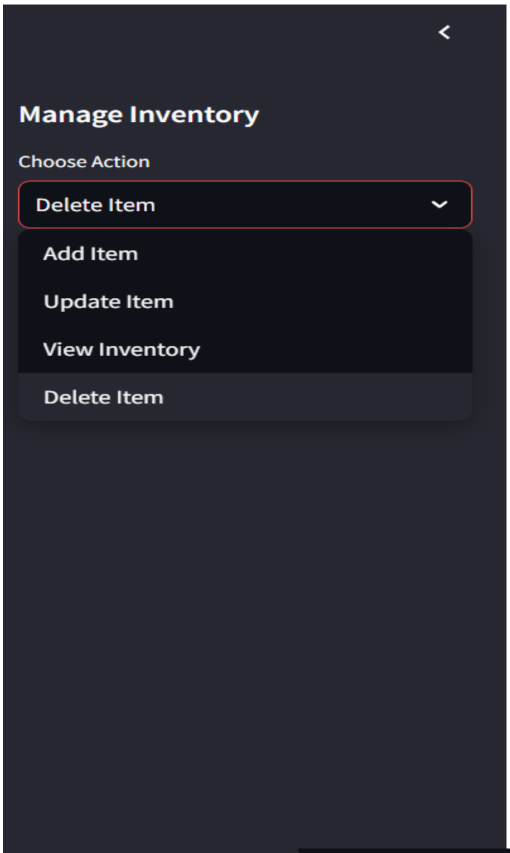
Add New Item

Item Name:

Quantity: 1

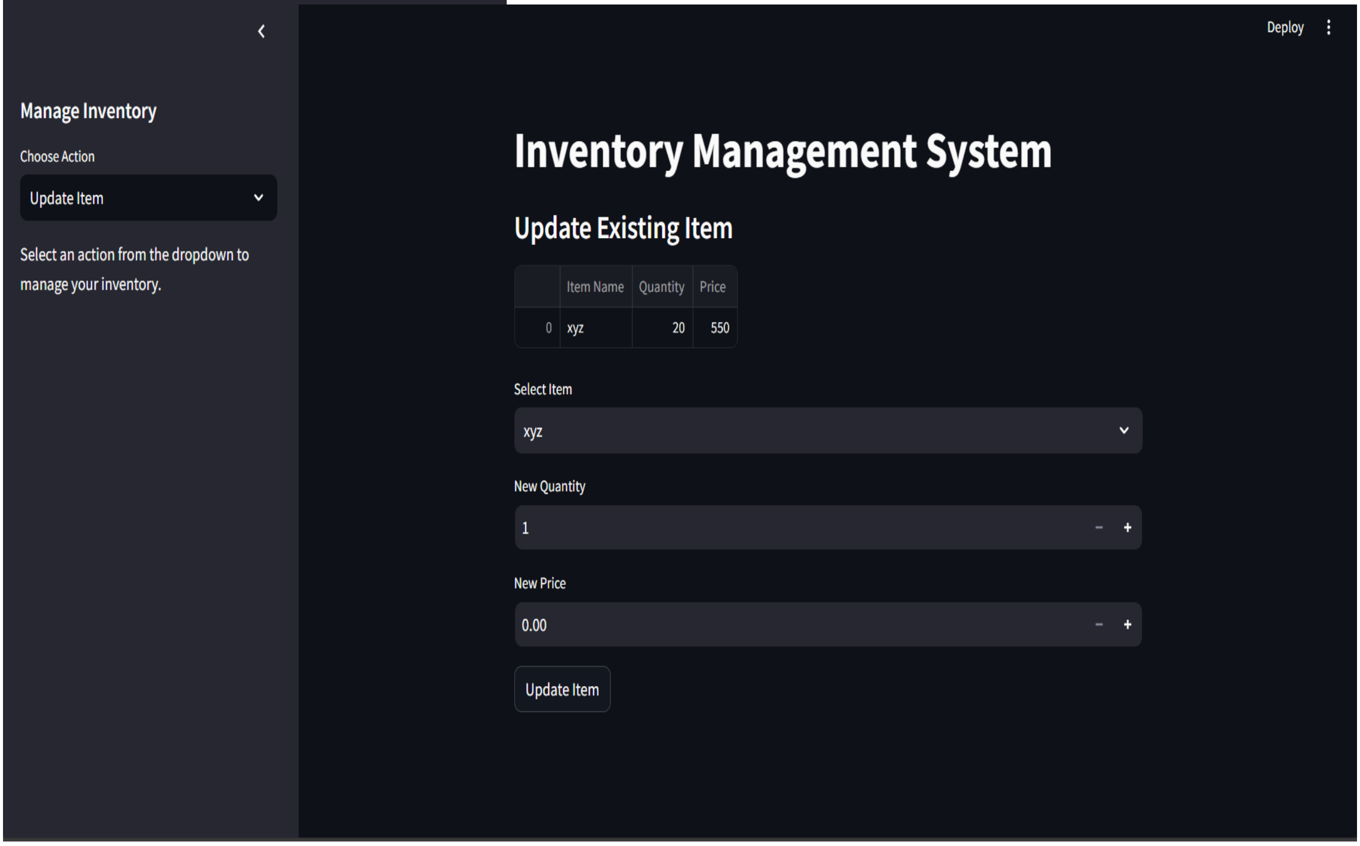
Price: 0.00

```
def load_data():
    try:
        return pd.read_csv(inventory_file)
    except FileNotFoundError:
        return pd.DataFrame(columns=['Product ID', 'Product Name', 'Quantity', 'Price'])

def save_data(df):
    df.to_csv(inventory_file, index=False)



# Load initial data
inventory = load_data()

# Streamlit app title
st.title("Inventory Management System")

# Sidebar for inventory options
st.sidebar.title("Inventory Options")
option = st.sidebar.selectbox("Choose an action", ["View Inventory", "Add Product", "Update Product",
"Delete Product", "Sales", "Restock", "Generate Report"])
# View Inventory
if option == "View Inventory":
    st.subheader("Current Inventory")
    st.dataframe(inventory)
# Add Product
elif option == "Add Product":
    st.subheader("Add New Product")
    product_id = st.text_input("Product ID")
    product_name = st.text_input("Product Name")
    quantity = st.number_input("Quantity", min_value=0, step=1)
    price = st.number_input("Price", min_value=0.0, format="%.2f")

    if st.button("Add Product"):
```



Manage Inventory

Choose Action: Delete Item



Manage Inventory

Choose Action: Update Item

Select an action from the dropdown to manage your inventory.

Inventory Management System

Update Existing Item

	Item Name	Quantity	Price
0	xyz	20	550

Select Item: xyz

New Quantity: 1

New Price: 0.00

```
        if product_id and product_name:
            new_product = pd.DataFrame([[product_id, product_name, quantity, price]], columns=['Product ID', 'Product Name', 'Quantity', 'Price'])
            inventory = pd.concat([inventory, new_product], ignore_index=True)
            save_data(inventory)
            st.success(f"Product '{product_name}' added to inventory!")
        else:
            st.error("Please fill all fields.")

# Update Product
elif option == "Update Product":
    st.subheader("Update Existing Product")
    product_id = st.text_input("Enter Product ID to Update")
    product = inventory[inventory['Product ID'] == product_id]

    if not product.empty:
        new_quantity = st.number_input("New Quantity", min_value=0, step=1)
        new_price = st.number_input("New Price", min_value=0.0, format="%.2f")
        if st.button("Update Product"):
            inventory.loc[inventory['Product ID'] == product_id, 'Quantity'] = new_quantity
            inventory.loc[inventory['Product ID'] == product_id, 'Price'] = new_price
            save_data(inventory)
            st.success("Product updated successfully!")
    else:
        st.warning("Product not found.")

# Delete Product
elif option == "Delete Product":
    st.subheader("Delete Product")
    product_id = st.text_input("Enter Product ID to Delete")

    if st.button("Delete Product"):
        if product_id in inventory['Product ID'].values:
            inventory = inventory[inventory['Product ID'] != product_id]
```

```
            save_data(inventory)
            st.success("Product deleted successfully!")
        else:
            st.warning("Product not found.")

# Sales
elif option == "Sales":
    st.subheader("Product Sales")
    product_id = st.text_input("Enter Product ID to Sell")
    quantity_to_sell = st.number_input("Quantity to Sell", min_value=1, step=1)

    if st.button("Sell Product"):
        product = inventory[inventory['Product ID'] == product_id]
        if not product.empty:
            current_quantity = product.iloc[0]['Quantity']
            if current_quantity >= quantity_to_sell:
                inventory.loc[inventory['Product ID'] == product_id, 'Quantity'] = current_quantity - quantity_to_sell
                save_data(inventory)
                st.success(f"Sold {quantity_to_sell} units of '{product.iloc[0]['Product Name']}'")
            else:
                st.error("Not enough stock available.")
        else:
            st.warning("Product not found.")

# Restock
elif option == "Restock":
    st.subheader("Restock Product")
    product_id = st.text_input("Enter Product ID to Restock")
    quantity_to_add = st.number_input("Quantity to Add", min_value=1, step=1)

    if st.button("Restock Product"):
        if product_id in inventory['Product ID'].values:
            inventory.loc[inventory['Product ID'] == product_id, 'Quantity'] += quantity_to_add
```

```
            save_data(inventory)
            st.success("Product restocked successfully!")
        else:
            st.warning("Product not found.")

# Generate Report
elif option == "Generate Report":
    st.subheader("Inventory Report")
    st.write("Total Products in Inventory:", len(inventory))
    st.write("Total Stock Quantity:", inventory['Quantity'].sum())
    st.write("Total Inventory Value: $", (inventory['Quantity'] * inventory['Price']).sum())
    st.bar_chart(inventory.set_index('Product Name')['Quantity'])
```

Conclusion

The Inventory Management System built with Streamlit allows users to perform essential inventory operations such as viewing, adding, updating, deleting, selling, restocking, and reporting. It is a flexible and easy-to-use tool for small to medium-sized businesses looking to maintain control over their stock efficiently. This system can be expanded by adding features like barcode scanning, multi-user roles, and detailed analytics to cater to larger organizations.