كلية الحاسبات والذكاء الإصطناعي
Faculty of Computers & Artificial Intelligence

جامعة حلوان
HELWAN UNIVERSITY

Software Engineering-2 Cover Sheet

Project Title: ……

Row Number (in PDF): ………….. Time: ………………….

#	Member Name (printed in Arabic)	Member Id (printed)	Handwritten signature
1	مايكل ممدوح سدراك	20210720	
2	كنزى عصام عبد الحليم	20210690	
3	احمد شفيق جلال	20210061	
4	يوسف مصطفى محمد سليم	20211112	
5	لارا محمد عبد الوهاب	20210699	
6	احمد سامى محمد	20210055	
7	احمد عادل احمد فؤاد	20210066	

Project Requirements(grades)

SRS	2	
SDD	2	
Validation	3	
OCL	4	
AOP	4	
Microservices	1	
Cloud	4	

______
20

Contents

Introduction

Effective Inventory Management: A Challenge for Businesses of All Sizes

In today's dynamic business landscape, efficient stock management is crucial for success. Businesses, regardless of their size or industry, face the constant challenge of maintaining optimal stock levels. Overstocking leads to wasted storage space, tied-up capital, and potential product obsolescence, while understocking can result in lost sales, frustrated customers, and reputational damage.

This project tackles this challenge by introducing a **Stock Management System** designed to streamline inventory control for businesses with multiple warehouses and a distributed workforce.

Introducing a User-Friendly Solution with Defined Roles

This system focuses on a two-tiered user approach: **Administrators** and **Supervisors**. **Administrators** possess the overarching authority to manage the system, create user accounts, oversee product information, and assign warehouses to supervisors. **Supervisors,** on the other hand, concentrate on their designated warehouses, monitoring and updating stock levels, and keeping the administrator informed of critical low-stock situations through automated reports.

Enhancing Collaboration and Streamlining Operations

By facilitating clear communication and role-based responsibilities, this Stock Management System promotes collaboration between administrators and supervisors. This paves the way for proactive inventory management, ensuring products are readily available while minimizing the risk of stockouts. Additionally, the system empowers supervisors to take ownership of their assigned warehouses, fostering accountability and a sense of responsibility for inventory control.

The Road Ahead: A System Poised for Growth

This project delves into the functionalities, data flow, and potential technology stack for this Stock Management System. We'll explore the benefits of such a system for businesses, considering its scalability and adaptability to accommodate future growth and evolving needs.

Functionality

The Stock Management System provides functionalities to address various inventory management needs for businesses with Admin and Supervisor roles.

Admin Functionalities:

- **User Management:**
    - Create new supervisor accounts with usernames and passwords.
    - Edit existing supervisor accounts (potentially including profile information or access levels).
    - Delete supervisor accounts when necessary.
- **Warehouse Management:**
    - Add new warehouses, specifying location details (optional).
    - Edit existing warehouse information (e.g., name, location).
    - Delete warehouses that are no longer in use.
- **Product Management:**
    - Add new products with descriptions and unique identifiers.
    - Edit existing product information (e.g., name, description).
    - Delete products that are no longer relevant.
    - Assign product quantities to specific warehouses, defining initial stock levels.
    - View product stock reports across all warehouses, providing insights into overall inventory levels.
- **Supervisor Management:**
    - Assign supervisors to manage specific warehouses, establishing clear lines of responsibility.
    - View a list of supervisors and their assigned warehouses for easy reference.

Supervisor Functionalities:

- **Stock Monitoring:**
    - View assigned warehouse details, including a list of products and their current stock quantities.
    - Update product stock quantities in the assigned warehouse as inventory fluctuates due to sales or deliveries.
- **Low-Stock Reporting:**
    - Generate automated reports for the Admin when product stock falls below a predefined threshold (configurable by the Admin). These reports should clearly identify products nearing depletion and their assigned warehouses.

Additional Considerations:

- **Security:** The system should implement user authentication and authorization to restrict access based on user roles (Admin vs. Supervisor). Only authorized users should be able to perform specific actions.
- **Data Validation:** Ensure data entered by users is valid (e.g., positive stock quantities, unique product identifiers).
- **Logging and Auditing:** Maintain logs of user actions (e.g., product additions, stock updates) for traceability and potential analysis.

Non-Functionality

Non-functional requirements encompass the broader characteristics of the system, focusing on how it behaves rather than its specific features. Here's an outline of some key non-functional requirements:

- **Performance:** The system should respond to user actions and data requests in a timely manner, especially for critical tasks like stock updates and low-stock reporting.

- **Scalability:** The system should be designed to accommodate future growth, potentially with an increasing number of products, warehouses, and users.

- **Availability:** The system should be highly available, minimizing downtime and ensuring continuous inventory management. Consider backup and recovery mechanisms for data security.

- **Usability:** The user interface should be intuitive and easy to use for both Admins and Supervisors, regardless of their technical expertise.

- **Security:** The system should prioritize data security by protecting user information, product details, and stock levels from unauthorized access. Implement strong encryption mechanisms and access controls.

- **Maintainability:** The code should be well-structured, documented, and modular for easier maintenance and future updates.

System Architecture Overview

The system architecture of the Stock Management System is designed to provide efficient inventory control for businesses with multiple warehouses and a distributed workforce. It incorporates various components that work together to ensure seamless stock management and streamline operations. The following is an overview of the system architecture:

- **Presentation Layer:** This layer is responsible for handling the user interface and user interactions. It includes the interfaces that users, such as administrators and supervisors, interact with to perform their respective tasks. The user interface should be intuitive, user-friendly, and accessible from different devices.
- **Application Layer:** The application layer serves as the core processing component of the system. It contains the business logic and functionality required for stock management. This layer handles user requests, processes data, and coordinates the interactions between different system components.
- **Database Layer:** The database layer is responsible for storing and managing the system's data. It includes the database management system (DBMS) and the underlying database that stores information related to products, warehouses, users, stock levels, and other relevant data. The database should be designed to support efficient data retrieval and storage, ensuring data integrity and security.
- **Integration Layer:** The integration layer enables communication and data exchange between the Stock Management System and external systems or services. It facilitates integration with other business systems, such as ERP (Enterprise Resource Planning) systems, to synchronize data and ensure consistency across different platforms.
- **Security Layer:** The security layer focuses on ensuring the confidentiality, integrity, and availability of the system and its data. It includes mechanisms for user authentication, authorization, and access control to restrict access to sensitive information. Encryption techniques and secure communication protocols should be implemented to protect data during transmission and storage.
- **Infrastructure Layer:** The infrastructure layer comprises the hardware and network infrastructure required to support the Stock Management System. This includes servers, storage devices, networking equipment, and other components necessary for hosting and deploying the system. The infrastructure should be scalable, reliable, and capable of meeting the system's performance and availability requirements.

Technology Stack:

Backend:

- Framework: Spring Boot
- Programming Language: Java
- Database: MySQL or MariaDB
- RESTful API: Spring Web or Spring MVC for building API endpoints
- Database Access: Spring Data JPA for seamless database interaction
- Authentication and Authorization: Spring Security for managing user authentication and authorization

Frontend:

- Framework: React
- Programming Languages: JavaScript, HTML, CSS
- State Management: Redux or React Context API for managing application state
- UI Component Library: Material-UI or Bootstrap for pre-built UI components and styling

Additional Technologies:

- API Documentation: Swagger or Springfox for generating API documentation
- Development Tools: IntelliJ IDEA or Eclipse (IDE), npm (Node Package Manager) for frontend dependencies, Git for version control
- Deployment: Docker for containerization, Jenkins or GitLab CI/CD for continuous integration and deployment

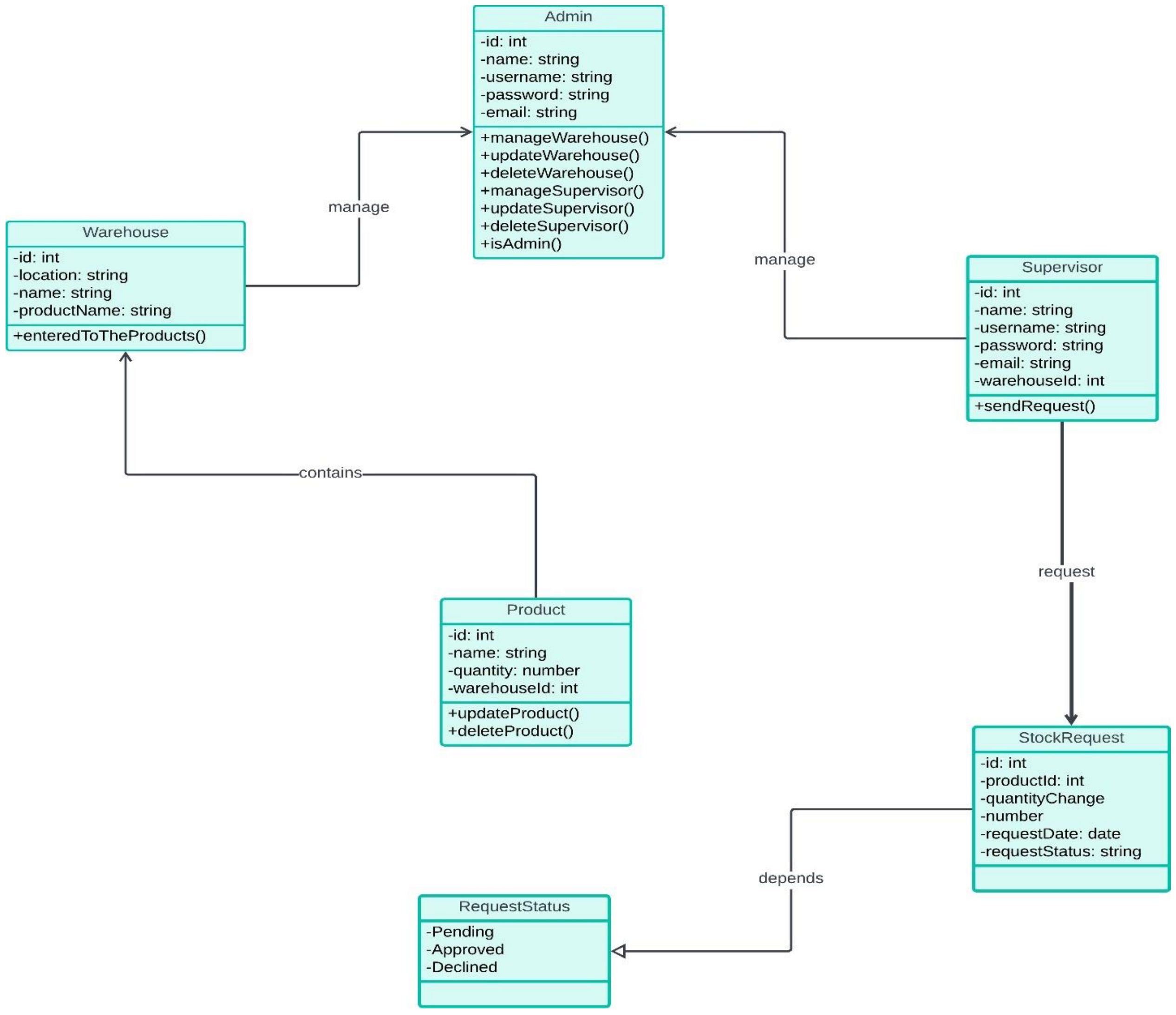
API Structure:

- Authentication Endpoints: /api/auth/login (POST), /api/auth/logout (POST)
- User Endpoints: /api/users (GET, POST, PUT, DELETE)
- Product Endpoints: /api/products (GET, POST, PUT, DELETE)
- Warehouse Endpoints: /api/warehouses (GET, POST, PUT, DELETE)
- Stock Level Endpoints: /api/stock-levels (GET, PUT)

Diagrams

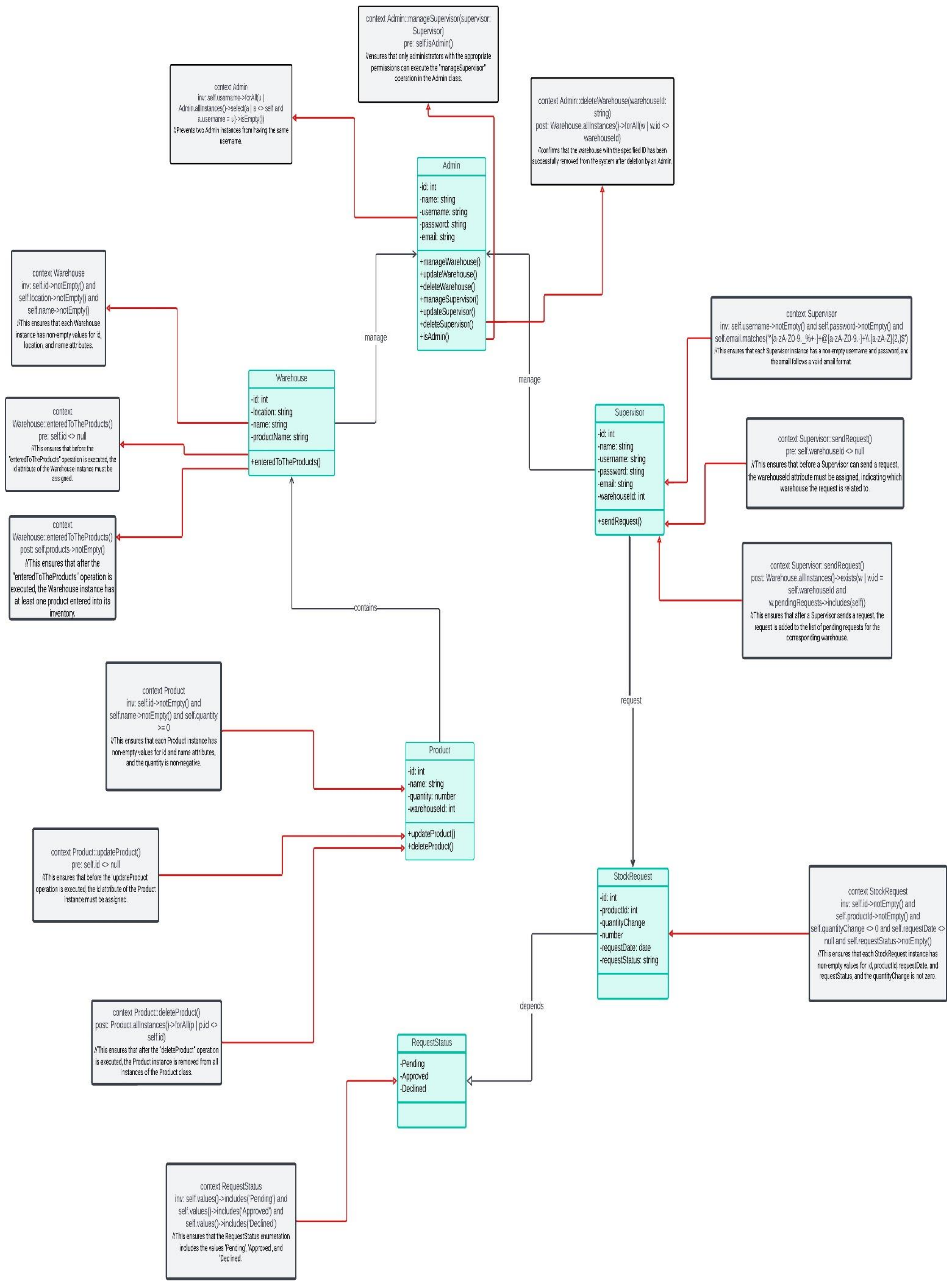
Class Diagram:

Class diagrams are like blueprints for object-oriented software. They show the building blocks (classes) and how they connect (relationships) to create the overall system. They help developers understand and design the software effectively.



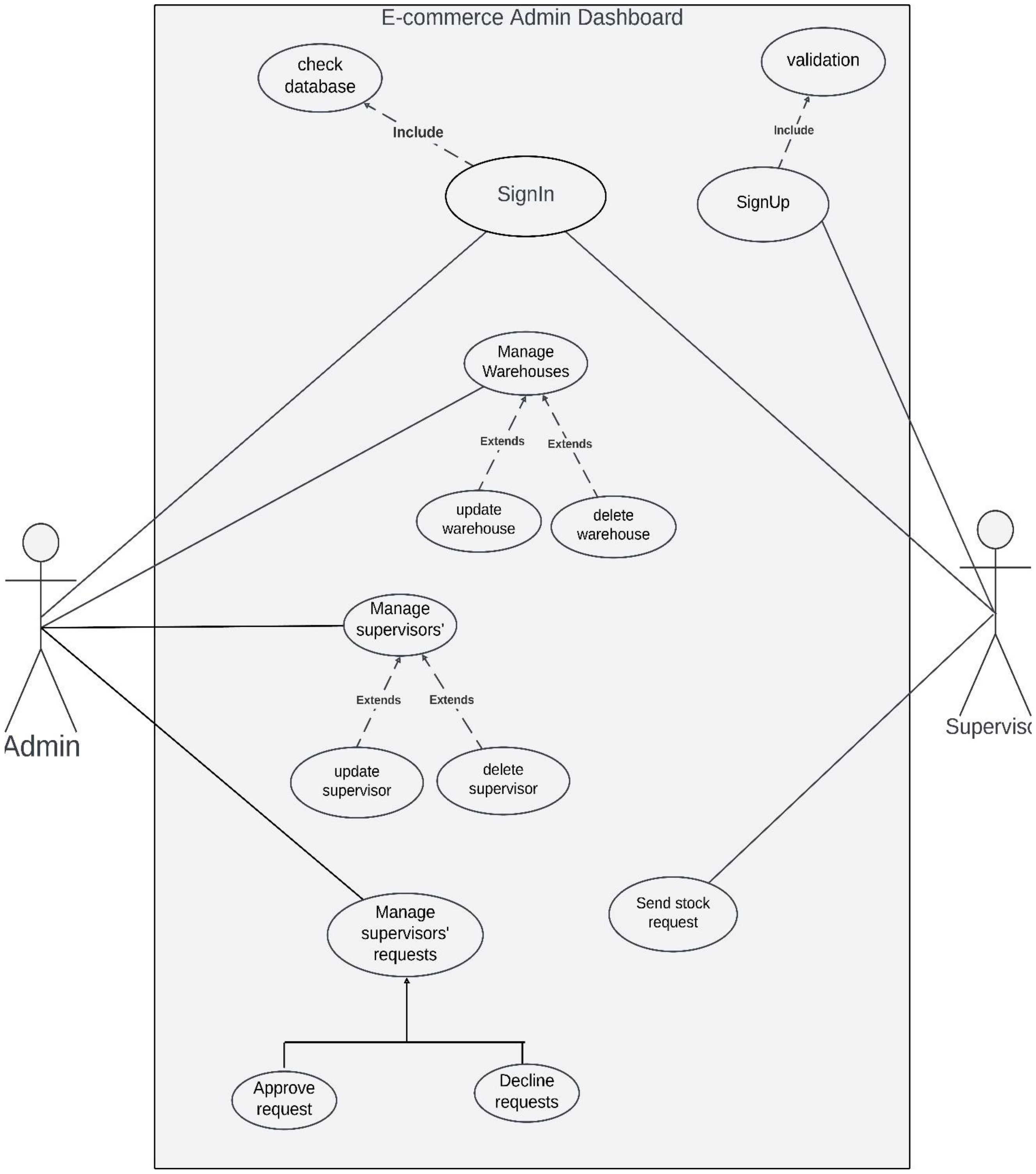
OCL Diagram:

a language for adding rules to UML models. It lets you define restrictions on data (like product stock must be non-negative) to ensure a well-designed system.


context Admin::manageSupervisor(supervisor: Supervisor)
pre: self.isAdmin()
//ensures that only administrators with the appropriate permissions can execute the "manageSupervisor" operation in the Admin class.
context Admin
inv: self.username->forAll(u | Admin.allInstances()->select(a | a <> self and a.username = u)->isEmpty())
//Prevents two Admin instances from having the same username.
context Admin::deleteWarehouse(warehouseId: string)
post: Warehouse.allInstances()->forAll(w | w.id <> warehouseId)
//confirms that the warehouse with the specified ID has been successfully removed from the system after deletion by an Admin.
Admin
-id: int
-name: string
-username: string
-password: string
-email: string
+manageWarehouse()
+updateWarehouse()
+deleteWarehouse()
+manageSupervisor()
+updateSupervisor()
+deleteSupervisor()
+isAdmin()
manage
manage
context Warehouse
inv: self.id->notEmpty() and self.location->notEmpty() and self.name->notEmpty()
//This ensures that each Warehouse instance has non-empty values for id, location, and name attributes.
Warehouse
-id: int
-location: string
-name: string
-productName: string
+enteredToTheProducts()
context Warehouse::enteredToTheProducts()
pre: self.id <> null
//This ensures that before the "enteredToTheProducts" operation is executed, the id attribute of the Warehouse instance must be assigned.
context Warehouse::enteredToTheProducts()
post: self.products->notEmpty()
//This ensures that after the "enteredToTheProducts" operation is executed, the Warehouse instance has at least one product entered into its inventory.
context Supervisor
inv: self.username->notEmpty() and self.password->notEmpty() and self.email.matches('^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$')
//This ensures that each Supervisor instance has a non-empty username and password, and the email follows a valid email format.
Supervisor
-id: int
-name: string
-username: string
-password: string
-email: string
-warehouseId: int
+sendRequest()
context Supervisor::sendRequest()
pre: self.warehouseId <> null
//This ensures that before a Supervisor can send a request, the warehouseId attribute must be assigned, indicating which warehouse the request is related to.
context Supervisor::sendRequest()
post: Warehouse.allInstances()->exists(w | w.id = self.warehouseId and w.pendingRequests->includes(self))
//This ensures that after a Supervisor sends a request, the request is added to the list of pending requests for the corresponding warehouse.
contains
request
context Product
inv: self.id->notEmpty() and self.name->notEmpty() and self.quantity >= 0
//This ensures that each Product instance has non-empty values for id and name attributes, and the quantity is non-negative.
Product
-id: int
-name: string
-quantity: number
-warehouseId: int
+updateProduct()
+deleteProduct()
context Product::updateProduct()
pre: self.id <> null
//This ensures that before the "updateProduct" operation is executed, the id attribute of the Product instance must be assigned.
StockRequest
-id: int
-productId: int
-quantityChange
-number
-requestDate: date
-requestStatus: string
context StockRequest
inv: self.id->notEmpty() and self.productId->notEmpty() and self.quantityChange <> 0 and self.requestDate <> null and self.requestStatus->notEmpty()
//This ensures that each StockRequest instance has non-empty values for id, productId, requestDate, and requestStatus, and the quantityChange is not zero.
context Product::deleteProduct()
post: Product.allInstances()->forAll(p | p.id <> self.id)
//This ensures that after the "deleteProduct" operation is executed, the Product instance is removed from all instances of the Product class.
depends
RequestStatus
-Pending
-Approved
-Declined
context RequestStatus
inv: self.values()->includes('Pending') and self.values()->includes('Approved') and self.values()->includes('Declined')
//This ensures that the RequestStatus enumeration includes the values 'Pending', 'Approved', and 'Declined'.

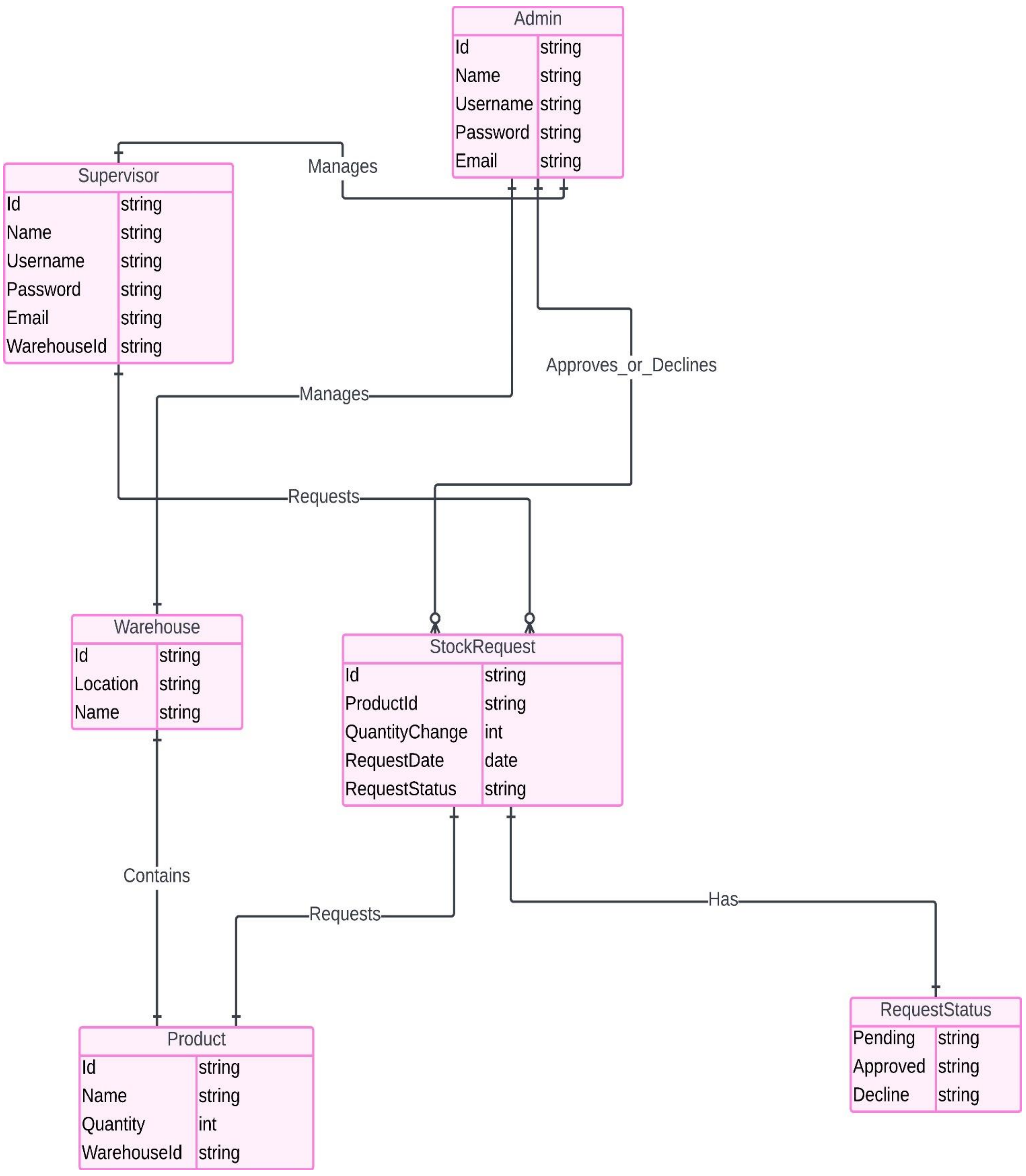
Use case Diagram:

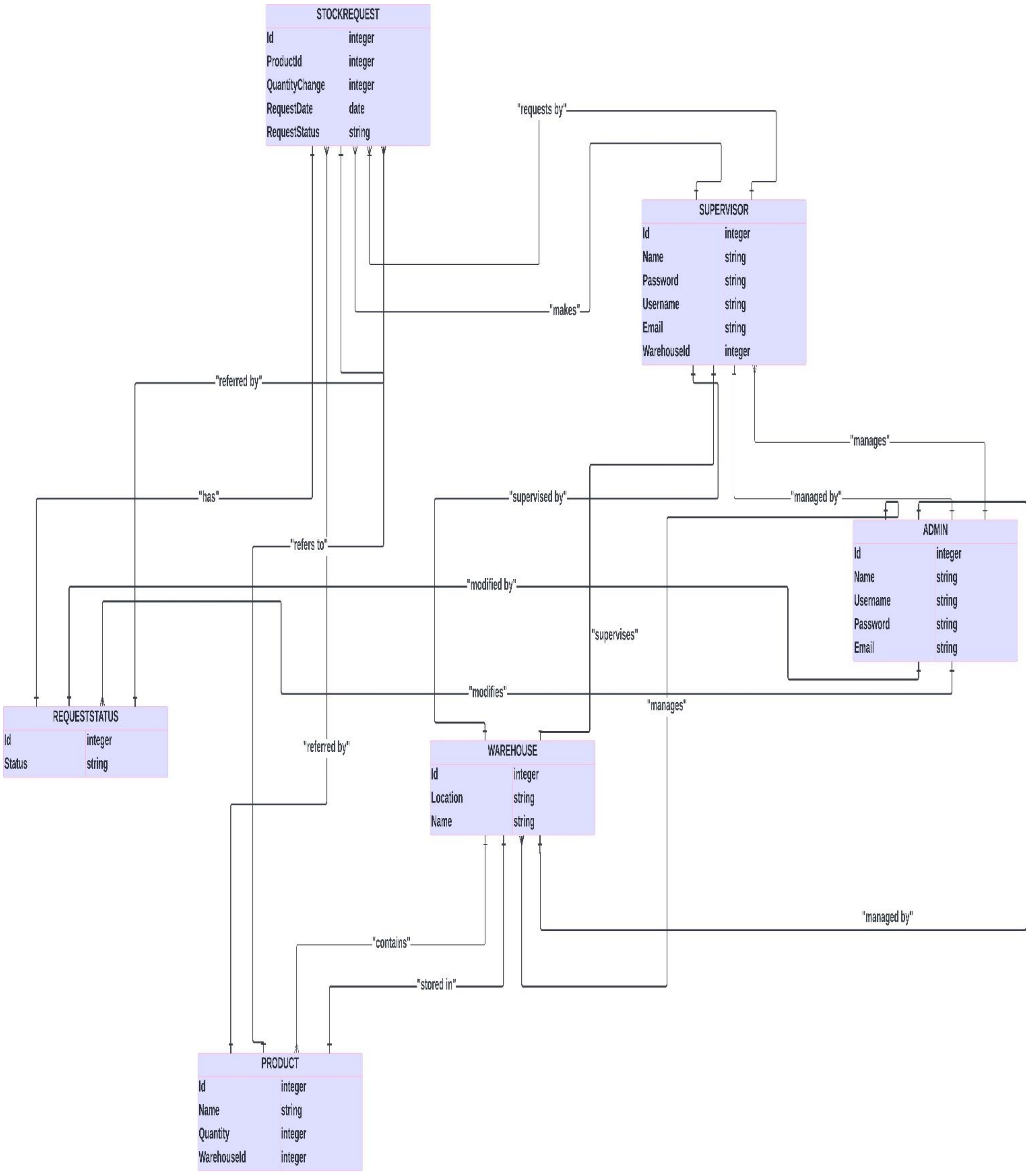
Use cases are short stories explaining how users interact with a system to achieve goals. They help ensure the system meets user needs.



ERD Diagram:

Blueprints for data! They show the main things (entities) in a system and how they connect (relationships) to store information effectively.

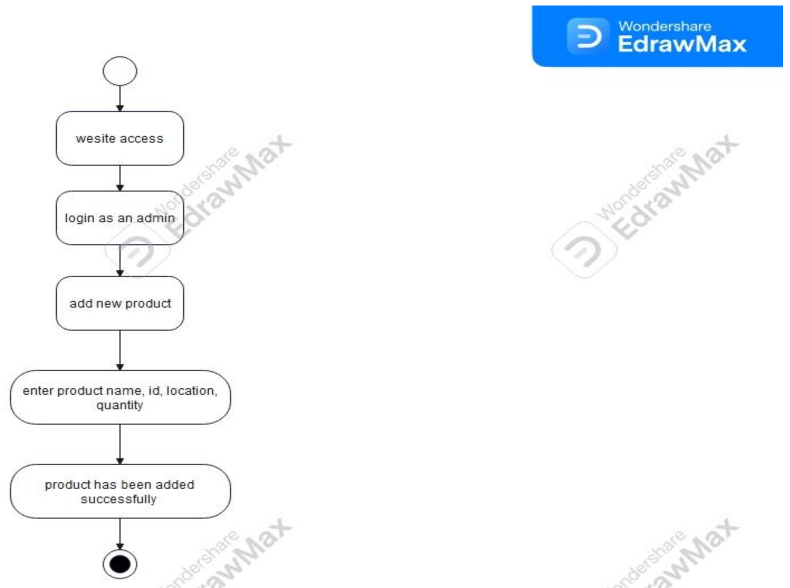



STOCKREQUEST
Id integer
ProductId integer
QuantityChange integer
RequestDate date
RequestStatus string
"requests by"
SUPERVISOR
Id integer
Name string
Password string
Username string
Email string
WarehouseId integer
"makes"
"referred by"
"manages"
"has"
"supervised by"
"managed by"
ADMIN
Id integer
Name string
Username string
Password string
Email string
"refers to"
"modified by"
"supervises"
"modifies"
"manages"
REQUESTSTATUS
Id integer
Status string
"referred by"
WAREHOUSE
Id integer
Location string
Name string
"managed by"
"contains"
"stored in"
PRODUCT
Id integer
Name string
Quantity integer
WarehouseId integer

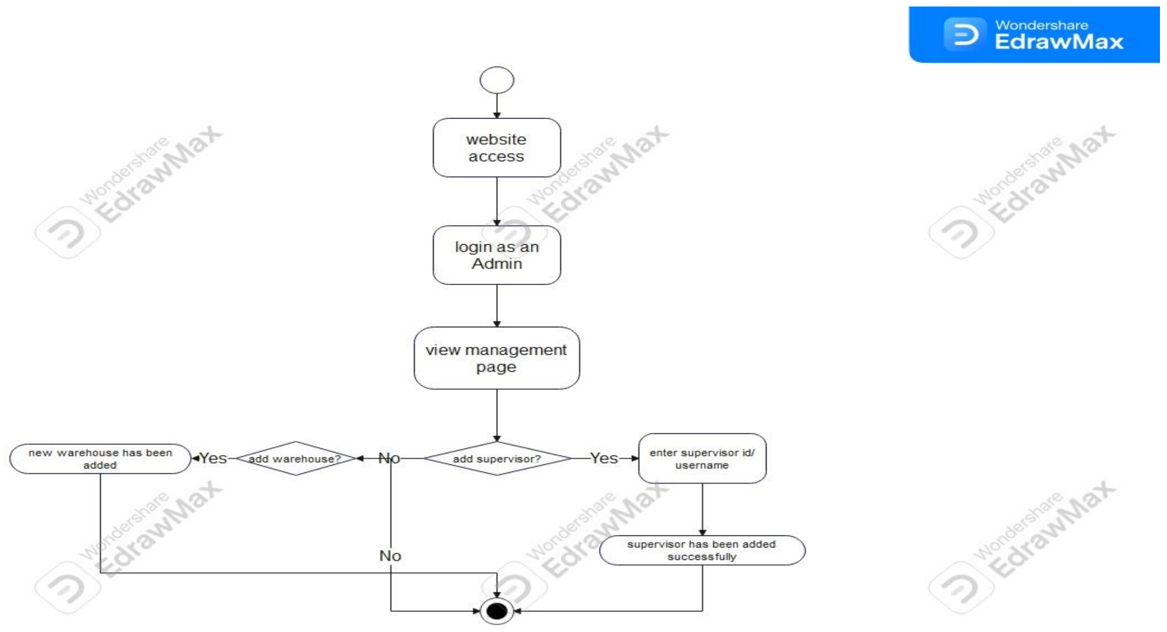
Activity Diagrams:

Flowcharts on steroids! They visually depict the flow of actions within a system, including decisions, steps, and potential alternative paths. They help understand the overall process and how different parts interact.

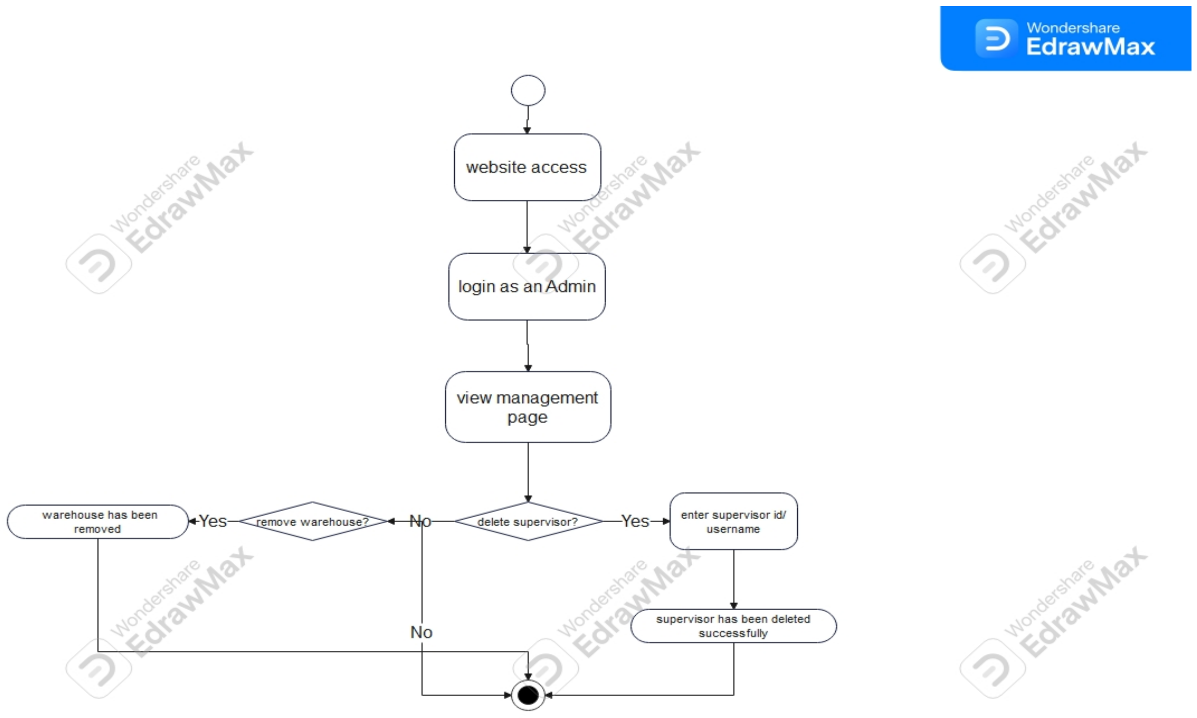
1-Add product diagram:


wesite access
login as an admin
add new product
enter product name, id, location, quantity
product has been added successfully

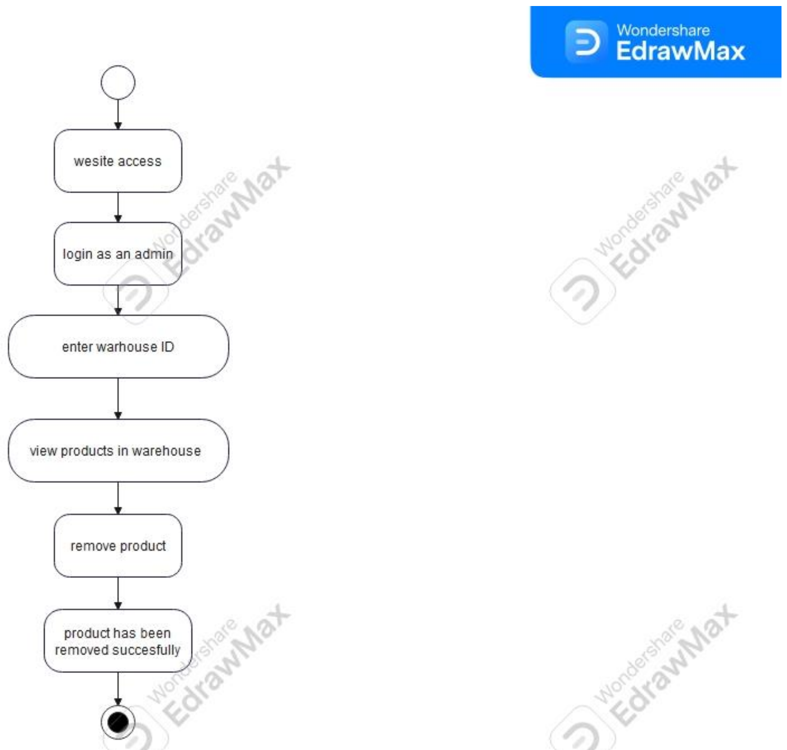
2-Add warehouse & supervisor diagram:


website access
login as an Admin
view management page
add supervisor?
Yes
enter supervisor id/ username
supervisor has been added successfully
No
add warehouse?
Yes
new warehouse has been added
No

3-Delete warehouse & supervisor diagram:

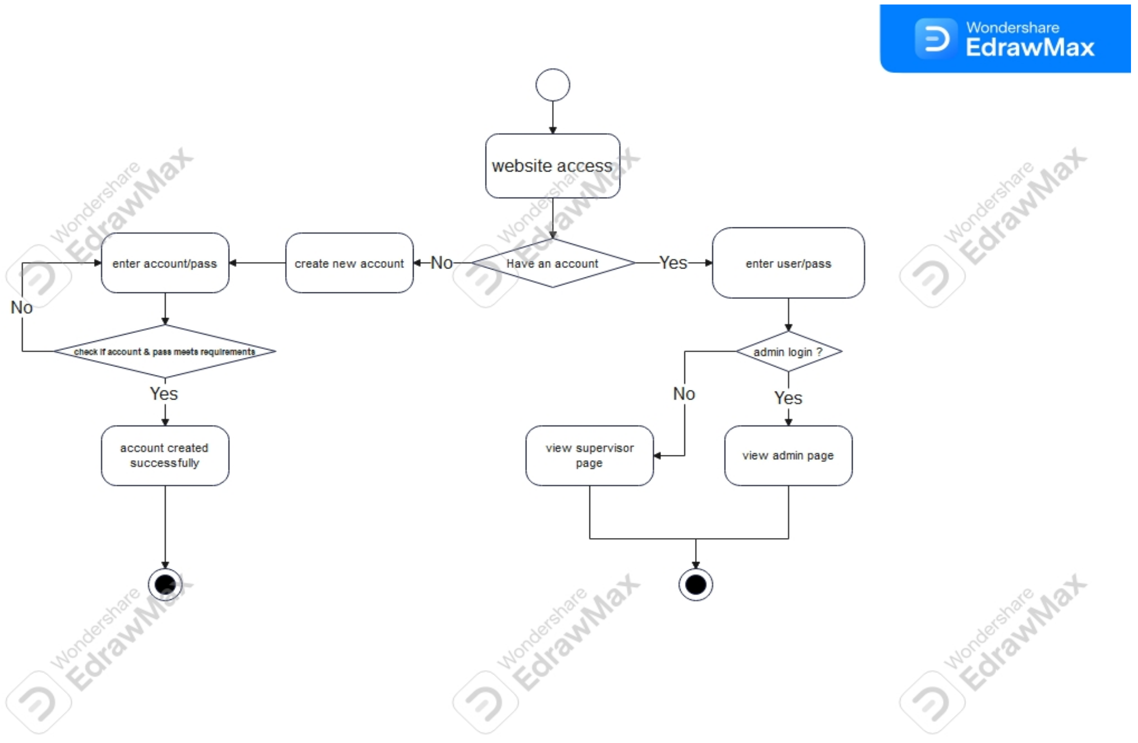


4-Delete product diagram:

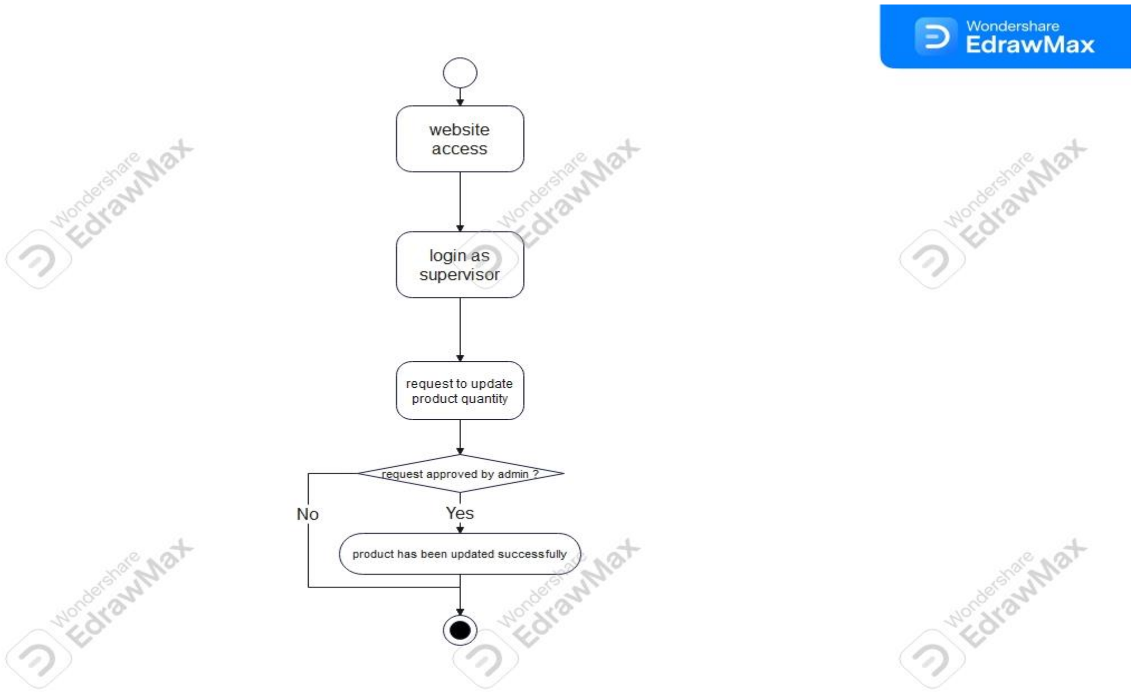


wesite access
login as an admin
enter warhouse ID
view products in warehouse
remove product
product has been removed succesfully

5-Login & Signup diagram:

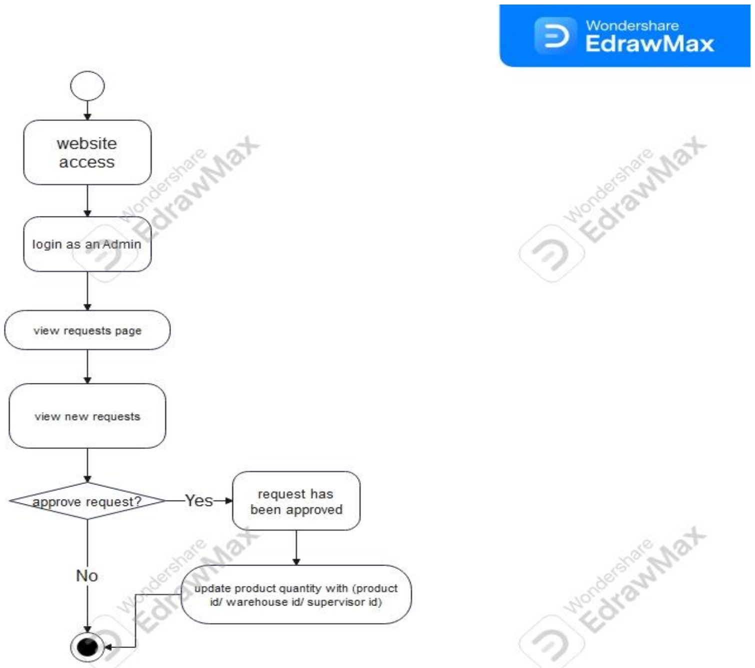


6-Update quantity Supervisor diagram:

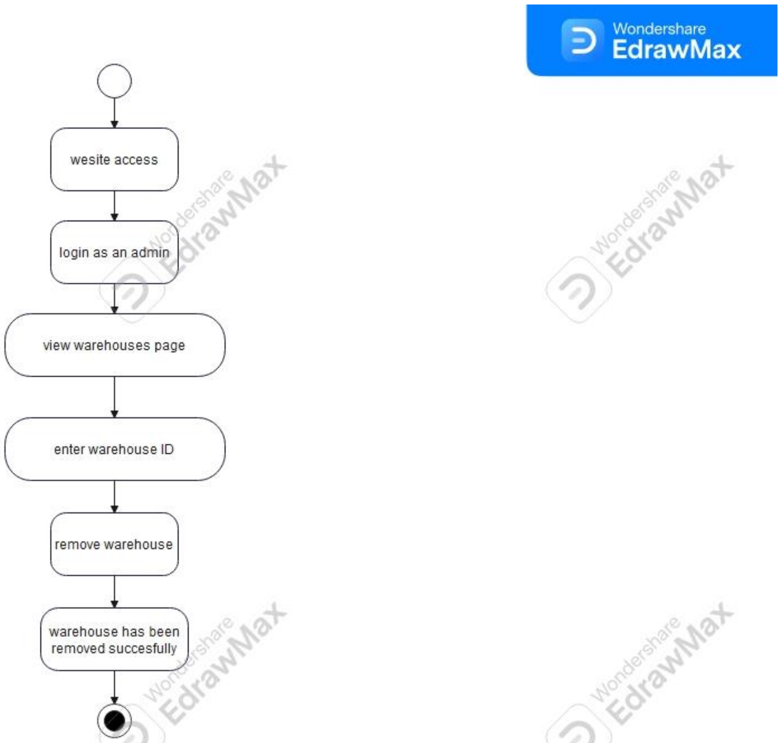


website access
login as supervisor
request to update product quantity
request approved by admin ?
No
Yes
product has been updated successfully

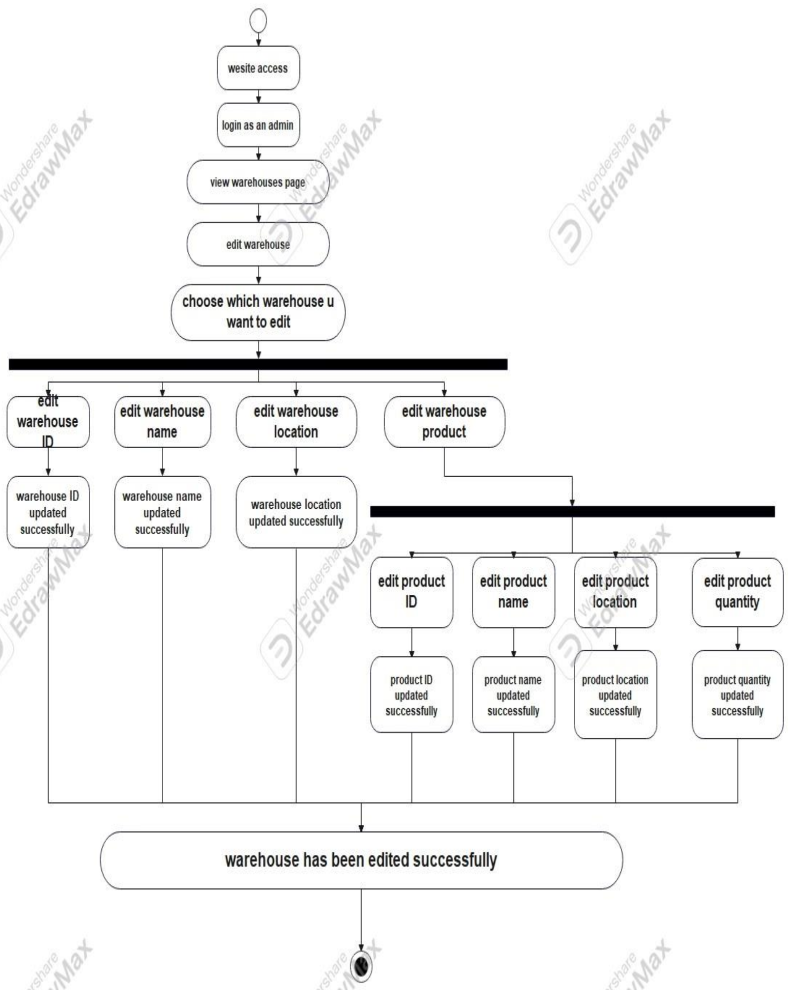
7-Update quantity Admin diagram:



8-Remove warehouse diagram:



9- Edit warehouse & product diagram:

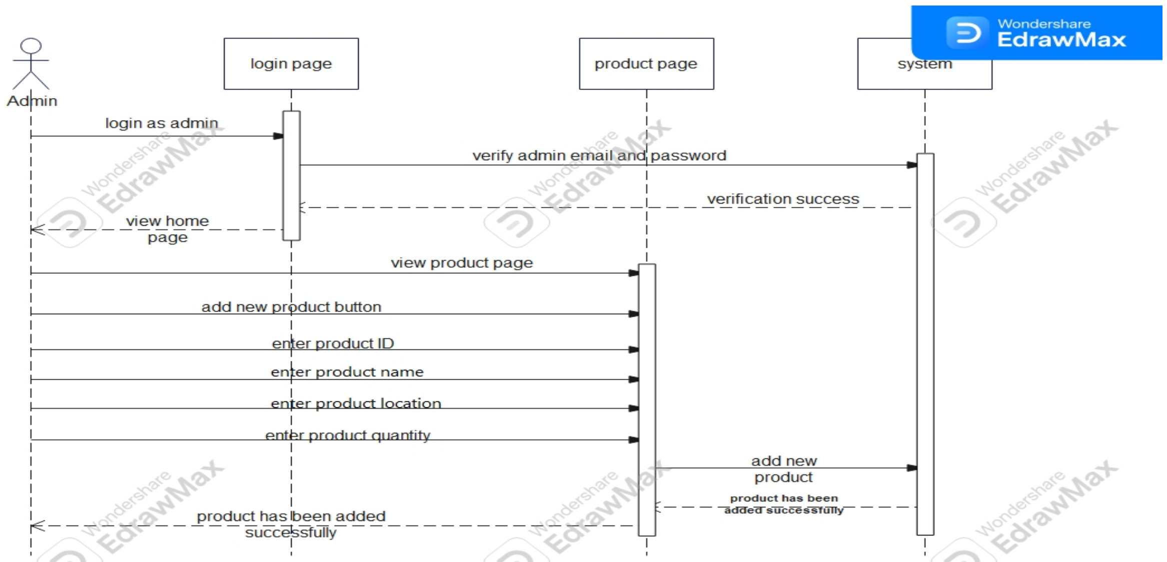


wesite access
login as an admin
view warehouses page
edit warehouse
choose which warehouse u want to edit
edit warehouse ID
edit warehouse name
edit warehouse location
edit warehouse product
warehouse ID updated successfully
warehouse name updated successfully
warehouse location updated successfully
edit product ID
edit product name
edit product location
edit product quantity
product ID updated successfully
product name updated successfully
product location updated successfully
product quantity updated successfully
warehouse has been edited successfully

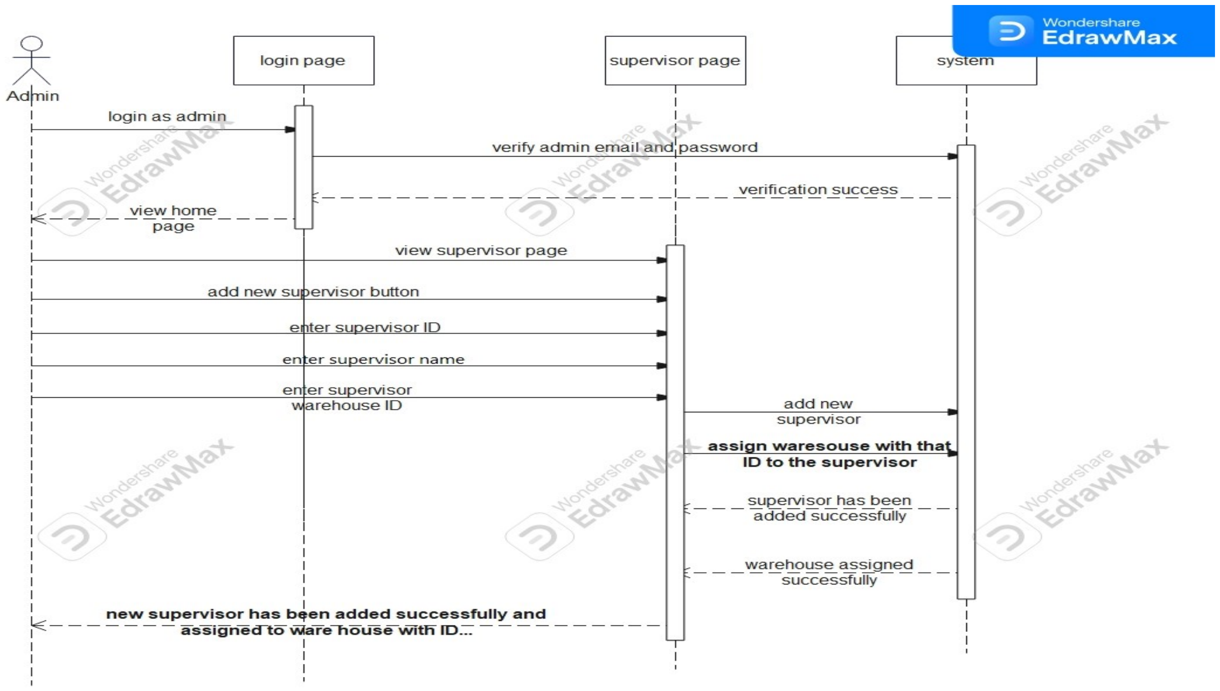
Sequence Diagrams:

Conversations in action! They show how objects in a system communicate with each other in a specific sequence, like a conversation, to achieve a task. They visualize the messages exchanged between objects and the order in which they occur.

1- Add product diagram:

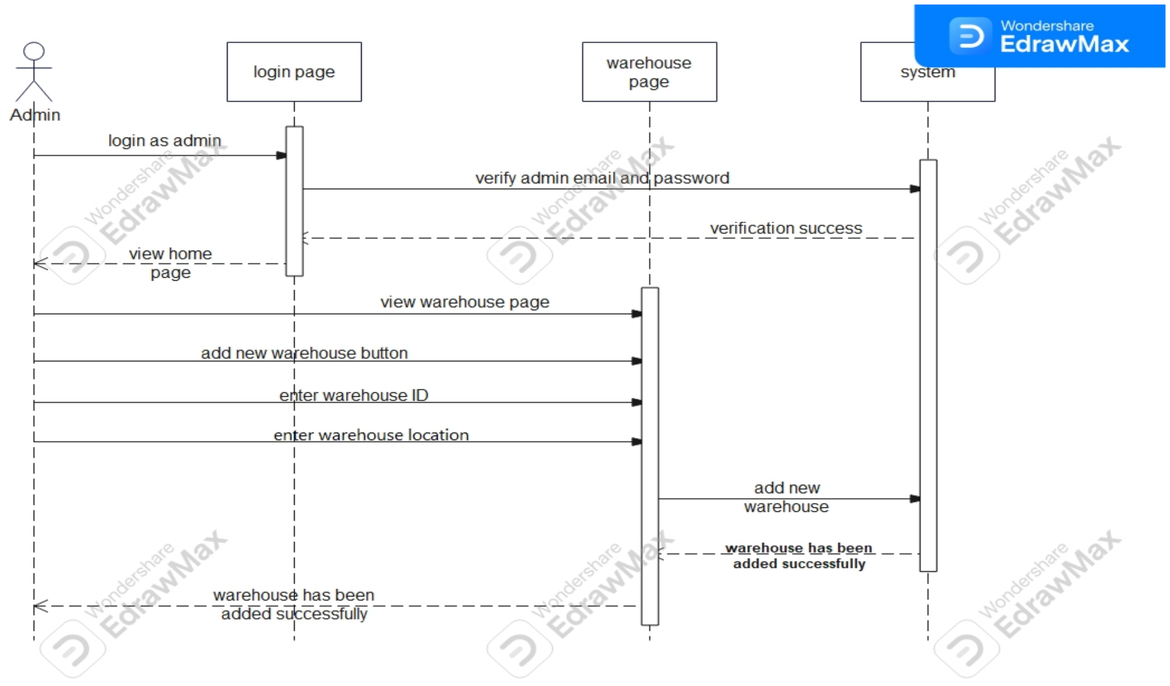


2- Add supervisor diagram:

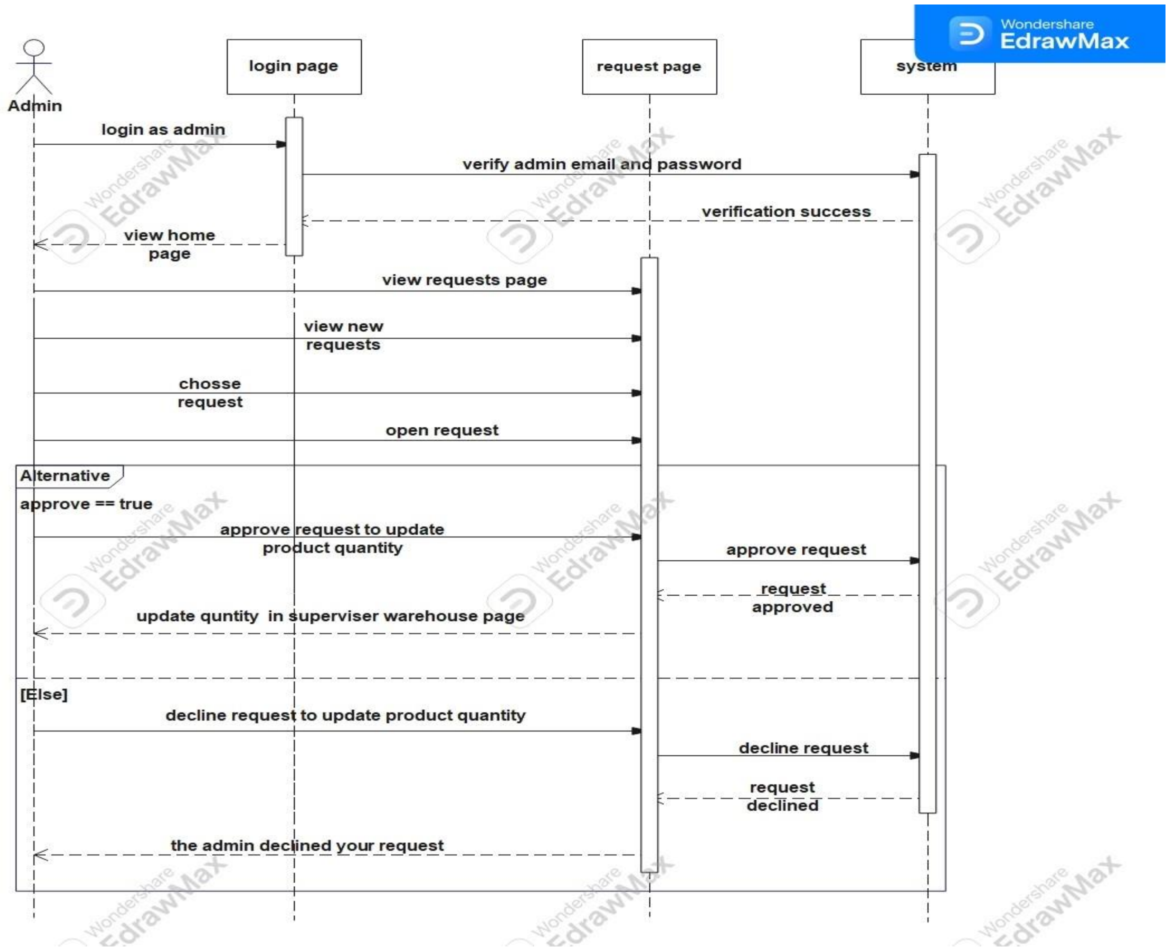


Admin
login page
supervisor page
system
login as admin
verify admin email and password
verification success
view home page
view supervisor page
add new supervisor button
enter supervisor ID
enter supervisor name
enter supervisor warehouse ID
add new supervisor
assign waresouse with that ID to the supervisor
supervisor has been added successfully
warehouse assigned successfully
new supervisor has been added successfully and assigned to ware house with ID...

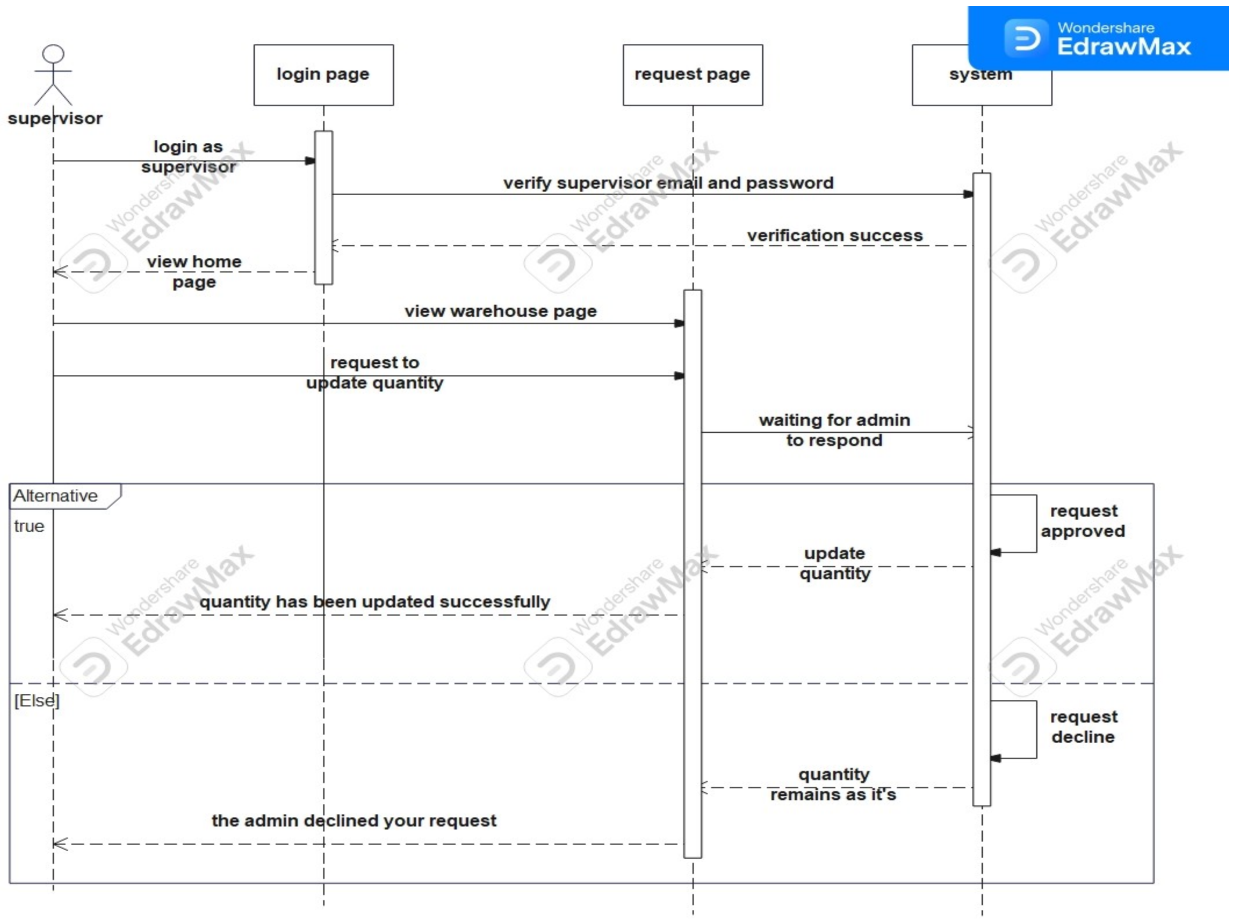
3- Add warehouse diagram:



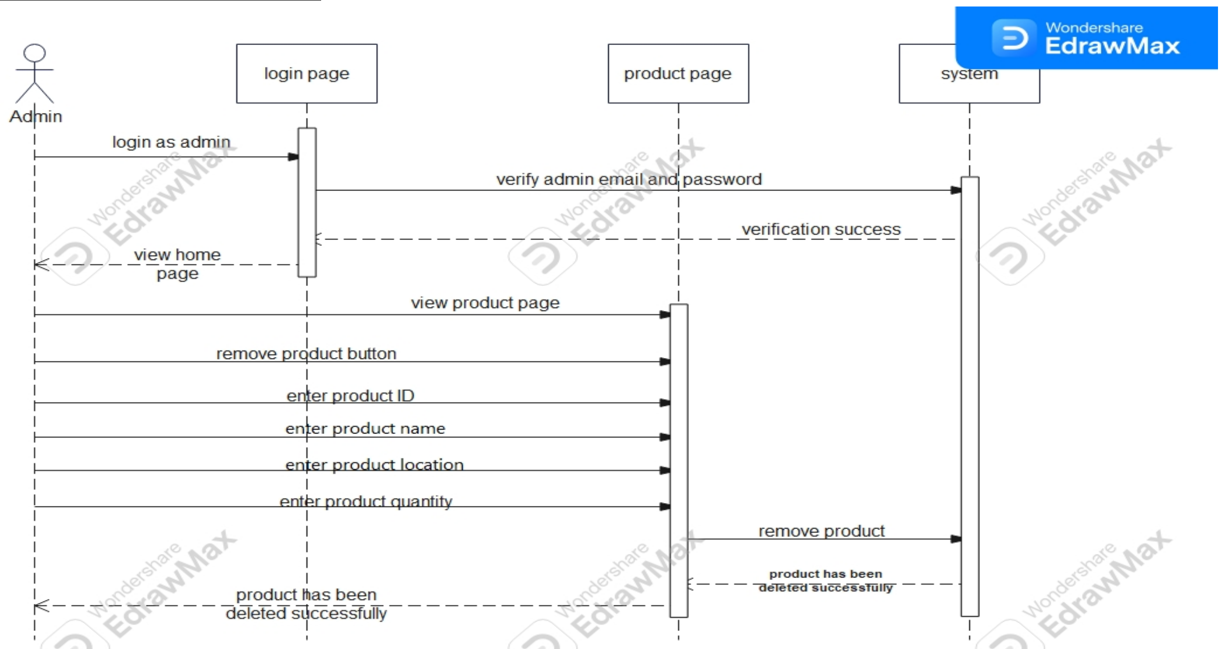
4- Update quantity Admin diagram:



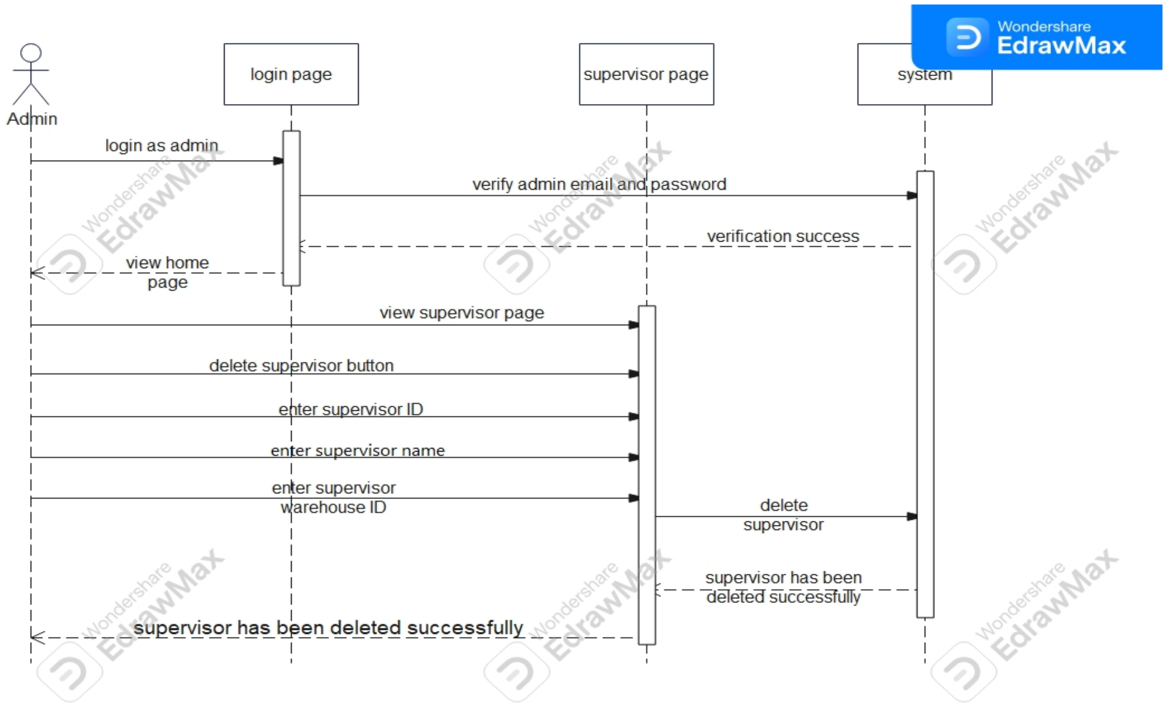
5- Update quantity supervisor diagram:



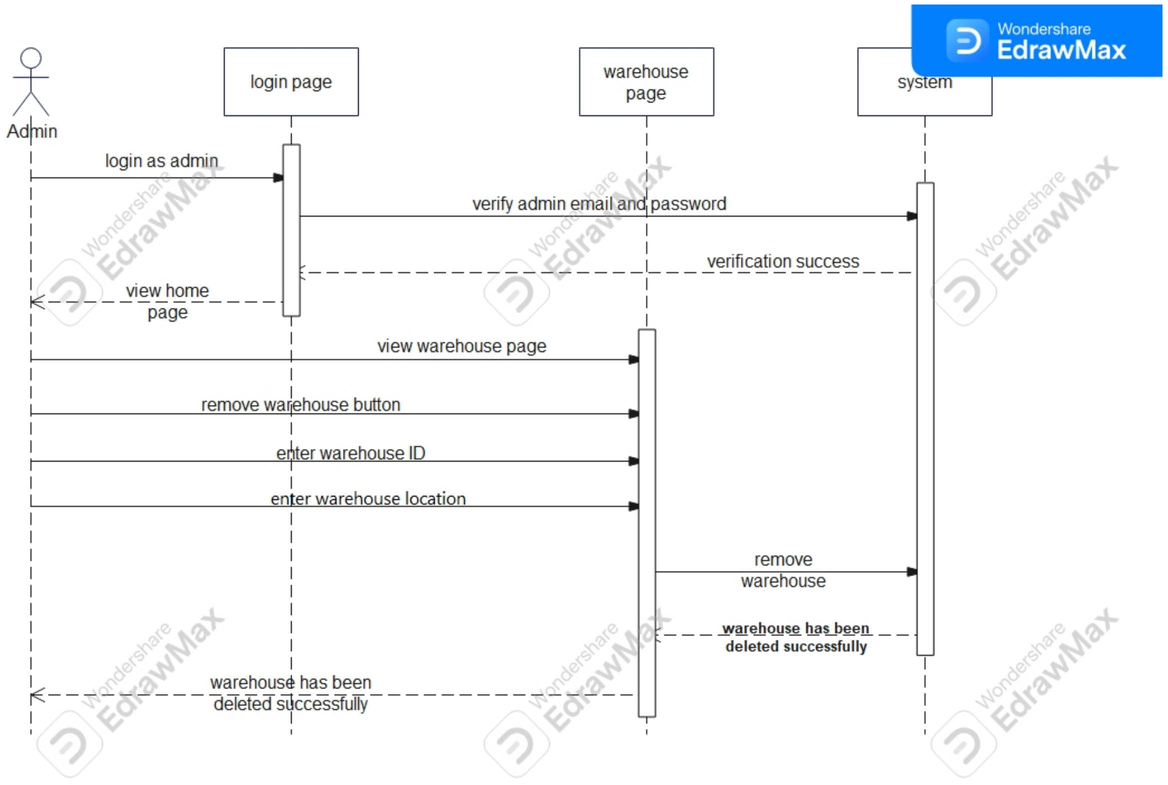
6- Delete product diagram:



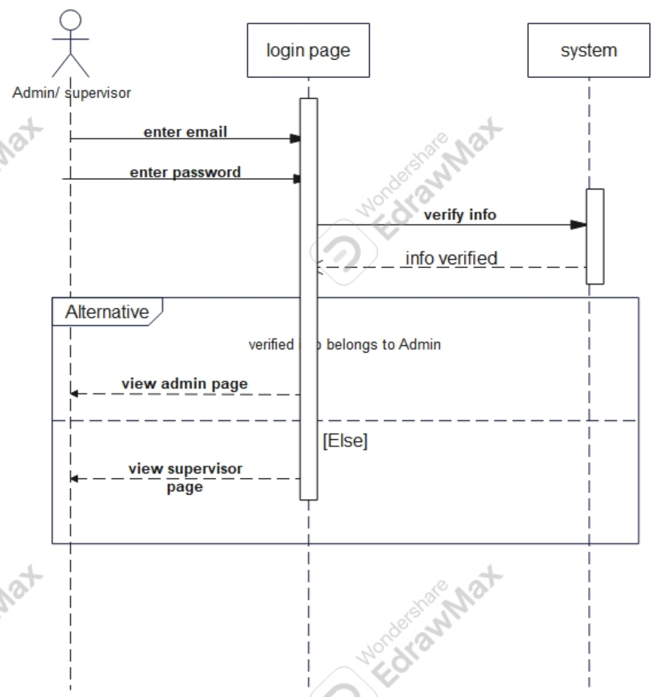
7- Delete supervisor diagram:



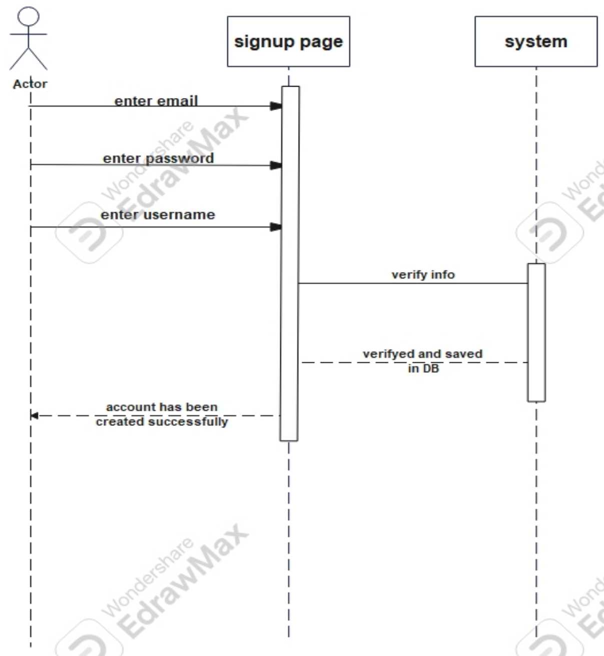
8- Delete warehouse diagram:



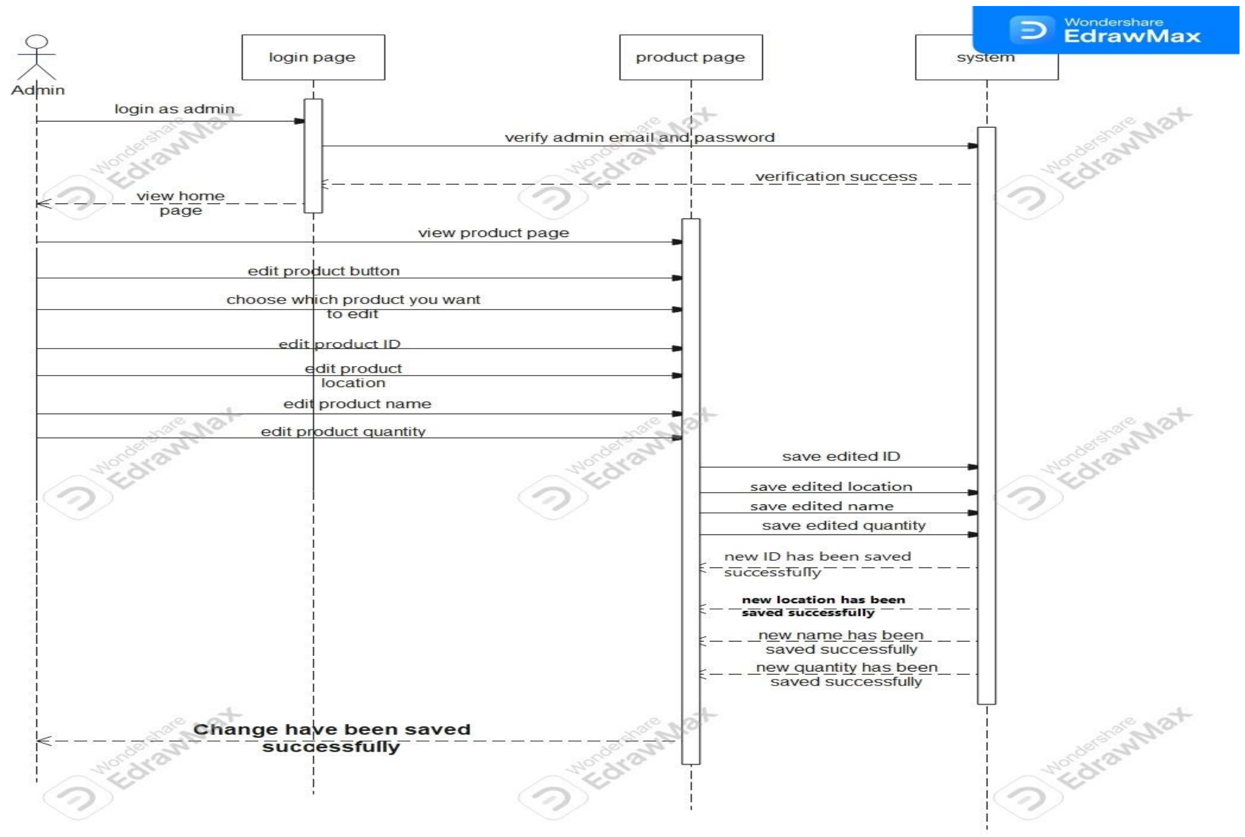
9- Login diagram:



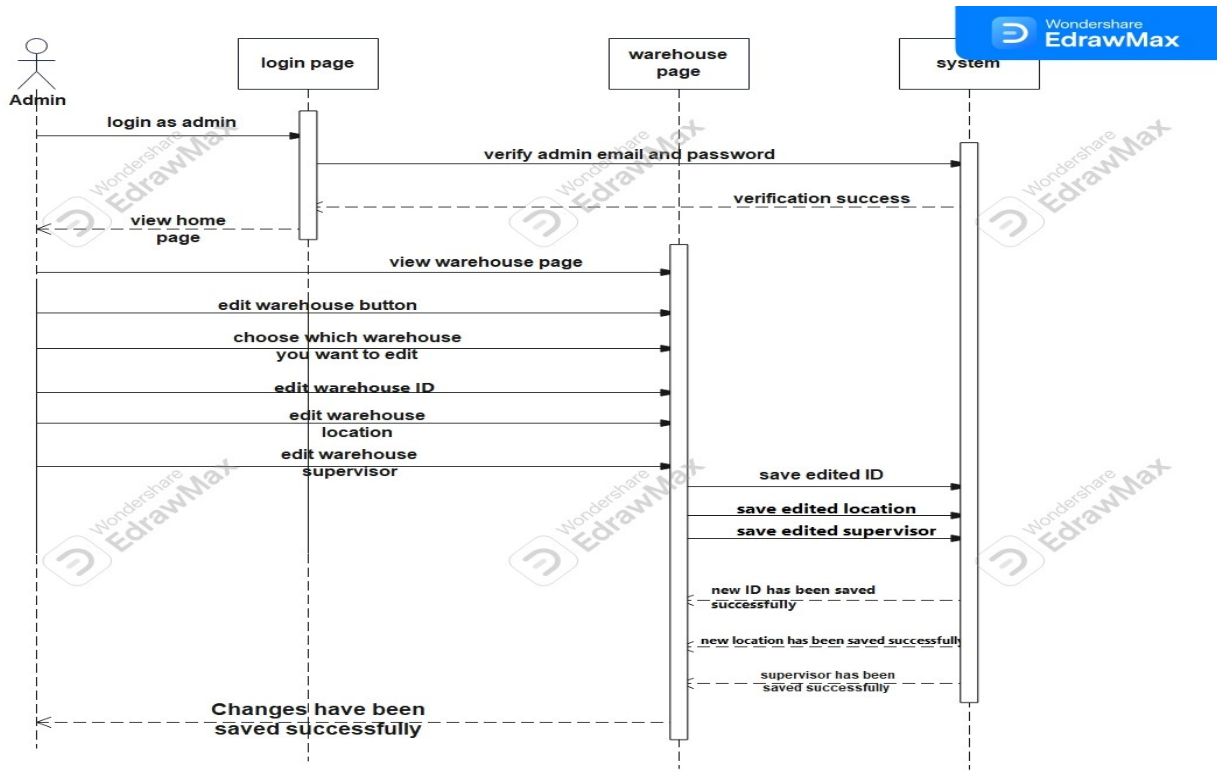
10- Signup diagram:



11- Edit product diagram:



12- Edit warehouse diagram:


Admin
login page
warehouse page
system
login as admin
verify admin email and password
verification success
view home page
view warehouse page
edit warehouse button
choose which warehouse you want to edit
edit warehouse ID
edit warehouse location
edit warehouse supervisor
save edited ID
save edited location
save edited supervisor
new ID has been saved successfully
new location has been saved successfully
supervisor has been saved successfully
Changes have been saved successfully