Egyptian E-Learning University

University Inventory Management System

By:

Abdalla Mohamed Abdel Aleem

Afnan Ayman Ali

Esraa Mohamed Mustafa

Nourhan Shabaan Mohamed

Pola Nasser Mansour

Rana Adel Mahmoud

Shahd Atef Mahmoud

A project graduation document submitted in partial fulfillment of the requirement for the BSc. Degree in software engineering

(Information Technology).

Faculty of Information Technology. South Valley University
Qena.Egypt
2024

University inventory management project applied on

Egyptian E-Learning University

By:

ID	NAMES
20-01950	Abdalla Mohamed Abdel Aleem
20-01769	Afnan Ayman Ali
20-01966	Esraa Mohamed Mustafa
20-01605	Nourhan Shaban Mohamed
20-01351	Pola Nasser Habeel
20-00340	Rana Adel Mahmoud
20-00397	Shahd Atef Mahmoud

Under the Supervision of:

Dr. Mahmoud Mohamed Hussein

Assisting Supervisor:

Eng. Ali Abdullah Ali

Faculty of Information Technology. South Valley University
Qena.Egypt

2024

Acknowledgment

The success of any project depends largely on the encouragement and guidelines of many others. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.
The special thanks go to our helpful supervisor
Dr. / Mahmoud Mohamed Hussien

The supervision and support he gave truly helped in the progression of our graduation project. The cooperation is much appreciated.
Lastly, we thank Almighty, Our Parents and all the people who helped, supported and encouraged us to successfully finish the graduation project whether they were in the university or in the industry.

Table of content

Abstract

In a university setting, efficiently managing and ordering supplies is crucial to support the diverse needs of faculty and staff. This project proposes the development of a comprehensive inventory management system aimed at simplifying the process of ordering supplies from the university administration store for university employees. The system will enable users to effortlessly order supplies such as computer screens and projectors, while providing a streamlined approval workflow to ensure all orders are properly authorized. Additionally, the system will offer real-time tracking of order status, from placement to approval and shipment, and notify users at each stage.

Key features of the system include a user-friendly interface for easy navigation and ordering, an automated approval mechanism to manage order validations, and a robust inventory management component to monitor stock levels and trigger restocking alerts. The system will also generate detailed reports and analytics to help university administrators understand usage patterns and optimize inventory management.

By integrating advanced technologies and ensuring a responsive design, this inventory management system aims to enhance operational efficiency, reduce administrative burden, and improve the overall ordering experience for university employees. The implementation of this system will result in a more organized and transparent process, ultimately contributing to the effective functioning of the university's supply chain.

CHAPTER 1: Project Description

1.1 INRODUCTION

Efficient management of university supplies is essential for supporting faculty and staff. The current manual and inefficient ordering process for various supplies, including devices and other essential items, needs improvement. This project aims to develop an inventory management system to streamline the process of ordering all types of supplies from the university administration store.

The system will offer a user-friendly platform for employees to browse, select, and order supplies. It will feature an automated approval workflow, real-time tracking, and notifications to ensure proper authorization and keep users informed of order status. Additionally, robust inventory management capabilities will monitor stock levels, generate low-stock alerts, and provide detailed reports and analytics.

By integrating advanced technologies and best practices, this system will enhance operational efficiency, improve the user experience, and ensure a transparent and organized supply management process, ultimately supporting the university's academic and administrative functions

1.1.1 Key feature:

1- **User Interface:**

Intuitive platform for browsing and ordering supplies.

Search functionality and clear product categorization.

Responsive design for accessibility across devices

2- **Automated Approval Workflow:**

Multi-level approval system (e.g., department head, finance).

Automated notifications to approve upon new order submissions.

Status updates for employees on order approval progress.

3- **Real-time Tracking and Notifications:**

Tracking of order status from placement to approval and shipment.

Notifications via email or system messages for status changes.

Estimated delivery dates for ordered items.

4- **Inventory Management:**

Real-time monitoring of stock levels.

Automatic alerts for low-stock items.

Detailed logs of inventory movements.

5- **Reporting and Analytics:**

Reports on usage patterns and trends.

Analytics on approval times and order fulfillment efficiency.

Cost tracking for budgeting and financial planning.

6- **Security and Authentication:**

Secure login system for university employees.

Role-based access control for different user levels.

Encryption of sensitive data to ensure confidentiality.

7- **Integration with University Systems:**

Seamless integration with existing university databases and systems.

Compatibility with university protocol

ls and standards.

8- **Scalability and Maintenance:**

Scalable architecture to accommodate future growth.

Ongoing maintenance and updates based on user feedback.

Regular monitoring and optimization for system performance.

1.2 PROBLEM DEFINITION

The current process of ordering supplies from the university administration store is inefficient and prone to errors, leading to delays, miscommunications, and suboptimal inventory management. University employees face challenges such as manual ordering processes, lack of real-time visibility into order status, cumbersome approval procedures, inventory mismanagement, and communication challenges regarding order updates. These issues hinder the university's ability to efficiently procure and manage supplies, resulting in increased administrative burden, wasted time and resources, and suboptimal utilization of funds. Therefore, there is a pressing need for a comprehensive inventory management system to streamline the ordering process, improve visibility and transparency, automate approvals, and optimize inventory management practices.

1.3 PROJECT OBJECTIVES

1- **Streamline Ordering Process:**

Develop a user-friendly platform that simplifies the process of ordering supplies from the university administration store.

2- **Automate Approval Workflow:**

Implement an automated approval system to expedite the authorization process and minimize delays in order processing.

3- **Enhance Visibility and Tracking:**

Provide real-time tracking of order status and notifications to keep users informed throughout the procurement process.

4- **Optimize Inventory Management:**

Develop robust inventory management capabilities to monitor stock levels, trigger restocking alerts, and prevent overstocking or stock outs.

5- **Improve Reporting and Analytics:**

Generate detailed reports and analytics to help university administrators understand usage patterns, optimize inventory levels, and make informed decisions.

6- **Ensure Security and Compliance:**

Implement stringent security measures to protect sensitive data and ensure compliance with data protection regulations and university policies.

7- **Enhance User Experience:**

Design a user-friendly interface and intuitive navigation to enhance the overall user experience and increase user satisfaction.

8- **Integrate with Existing Systems:**

Seamlessly integrate with existing university databases and systems to ensure compatibility and facilitate data exchange.

9- **Promote Efficiency and Cost Savings:**

Enhance operational efficiency, reduce administrative burden, and optimize resource allocation to achieve cost savings and maximize the value of university resources.

10- **Support Organizational Goals:**

Contribute to the achievement of organizational goals by improving supply chain management, supporting academic and administrative functions, and enhancing overall operational effectiveness.

1.4 OPTIMIZING INVENTORY MANAGEMENT IN A UNIVERSITY SETTING

1. **Introduction:**
   - Brief overview of the challenges faced by universities in managing their inventories.
2. **Statement of the Problem:**
   - Lack of real-time visibility into inventory levels and usage patterns.
3. **Objectives**
   - Develop a centralized Inventory Management System (IMS) for the university.
4. **Scope:**
   - Inclusion of all types of inventory items used across different university departments.
   - Accessibility for authorized personnel from various departments.
5. **Significance of the Study:**

- Cost savings through optimized inventory levels and reduced wastage.
- Improved resource allocation and planning.
- Enhanced productivity by ensuring the availability of necessary resources.

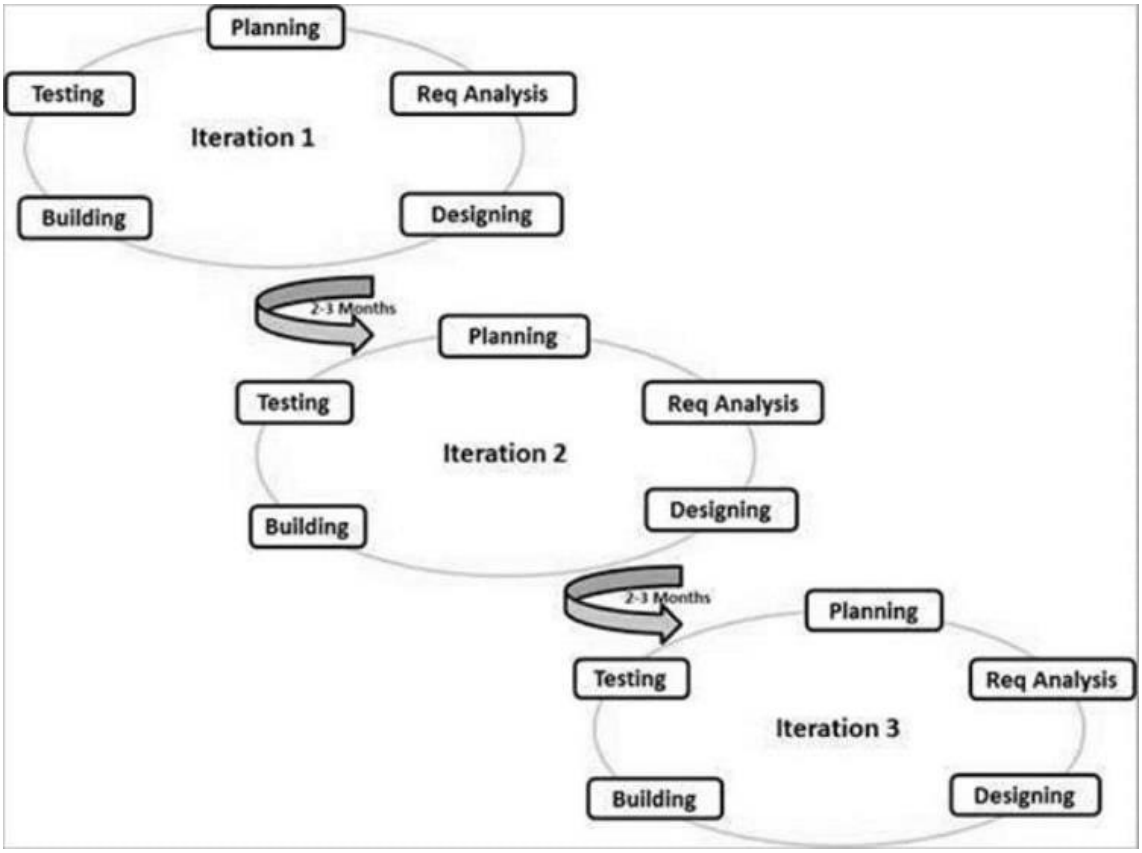
6. **Expected Outcomes:**
   - Improved accuracy and timeliness in inventory tracking.
   - Reduction in excess inventory and associated costs.
   - Enhanced collaboration among departments.
   - Streamlined procurement processes.
7. **Constraints and Limitations:**
   - Budget constraints for system development and implementation.
   - Potential resistance to change from existing manual processes.
   - Technical constraints based on the existing IT infrastructure of the university.
8. **Conclusion:**
   - Summary of the identified problems, proposed solutions, and potential benefits.

1.5 PROJECT DEVELOPMENT METHODOLOGY

1.5.1 Agile Model

Agile is an iterative and incremental approach to software development that prioritizes flexibility, collaboration, and customer satisfaction. It involves breaking down the development process into small, manageable iterations called sprints, typically lasting two to four weeks. Agile emphasizes continuous feedback, adaptability to changing requirements, and close collaboration between development teams and stakeholders.

- **General Overview of the Agile Model**



1.5.2 Why We Use Agile Model?

1- **Flexibility and Adaptability:**
   Agile allows for changes to be accommodated easily, making it well-suited for projects with evolving requirements.
2- **Continuous Delivery of Value:**
   Regular iterations result in frequent deliveries of working software, providing continuous value to stakeholders.
3- **Customer Involvement:**
   Regular collaboration and feedback sessions keep customers and stakeholders involved throughout the development process, ensuring the final product meets their expectations.
4- **Early Identification of Issues:**
   Issues and challenges are identified early in the development process, allowing for timely resolution.
5- **Improved Product Quality:**
   Continuous attention to technical excellence and regular testing contributes to a higher-quality end product.
6- **Motivated and Empowered Teams:**
   Agile promotes self-organizing and cross-functional teams, fostering a sense of ownership and motivation among team members.

1.6 PROBLEM SOLUTION:

From this, we express the importance of having inventory system to reduce the stages of request orientation so that when the employee indicates his needs, the request is directed directly to all superiors from stores to purchases to the direct manager at the same time the request occurs, and in the event of the availability of the request and approval, it is provided.

However, in the event of a lack of response or approval from any party for a period of 3 days, the request is directed to Purchases.

CHAPTER 2: System Requirements & Diagrams

2.1 INTRODUCATION TO SYSTEM ANALYSIS

System analysis is a crucial process in the field of information technology and software engineering that involves studying, understanding, and evaluating a system to identify its components, functions, and interactions. It aims to improve system efficiency, effectiveness, and overall performance by analyzing its current state, identifying areas for enhancement, and proposing solutions to meet user needs and organizational objectives.

2.2 SYSTEM REQUIREMENTS

2.2.1 Part 1: User (Employee) Functionality

1- Requesting Supplies

Browse Products: Users can view a catalog of available supplies.

Place Orders: Users can request any supplies they need by placing an order through the system.

Track Orders: Users can follow up on their orders to check whether they have been approved or shipped.

2- Order Management

Order Status: Users can view the status of their orders in real-time.

Notifications: Users receive notifications about the approval, shipment, or rejection of their orders

2.2.2 Part 2: Responsible Person (Admin) Functionality

1- User Management

A) Add Users:

Admins can add new users, granting them the ability to place orders.

B) Modify Users:

Admins can update user information and permissions.

C) Delete Users:

Admins can remove users from the system.

2- Product Management

A) Add Products:

Admins can add new products to the inventory catalog.

B) Modify Products:

Admins can update product details such as description, quantity, and availability.

C) Delete Products:

Admins can remove products from the catalog.

3- Order Management

A) Approve Orders:

Admins review and approve or reject orders placed by users.

B) Track Shipment:

Admins confirm the shipment of approved orders and update the system accordingly.

Detailed Analysis

C) Current Workflow:

Document the existing workflow for ordering and approving supplies.

D) Desired Workflow:

Define the desired workflow with the new system in place, highlighting improvements.

Problem Identification

E) Manual Processes:

Identify issues with the current manual processes, such as delays, errors, and lack of transparency.

F) Approval Bottlenecks:

Pinpoint bottlenecks in the approval process that lead to delays.

G) Inventory Management:

Identify problems with inventory tracking, such as stockouts and overstocking.

Feasibility Study

H) Technical Feasibility:

Assess the technical requirements and resources needed for the project.

I) Economic Feasibility:

Evaluate the cost-benefit analysis to ensure the project is economically viable.

J) Operational Feasibility:

Ensure that the new system can be integrated smoothly into the university's existing operations.

2.3 REQUIREMENT SPECIFICATION

2.3.1 Functional Requirements:

- For Users (University Employees):

1. **Placing Orders:**
    - Users should be able to browse products and add them to a shopping cart.
    - Users should be able to place orders and receive confirmation.
    - Users should be able to view past orders and reorder items.
2. **Tracking Order Status:**
    - Users should be able to track the status of their orders in real-time.
    - Notifications should be sent to users regarding order status updates (e.g., processing, shipped, delivered).
3. **Account Management:**
    - Users should be able to create and manage their accounts.
    - Users should be able to update their personal information and preferences.

- For Admins:

1. **User Management:**
    - Admins should be able to add, update, and remove user accounts.
    - Admins should be able to manage user roles and permissions.
2. **Product Management:**
    - Admins should be able to add, update, and remove products from the catalog.
    - Admins should be able to categorize products and manage inventory levels.
3. **Order Management:**
    - Admins should be able to view and manage all orders.
    - Admins should be able to update order statuses and handle returns or cancellations.

2.3.2 Non-Functional Requirements:

1. **Performance Requirements:**
    - The system should support concurrent users without significant performance degradation.
    - The system should have a response time of less than 2 seconds for most user interactions.

2. **Security Measures:**
    - The system should use SSL encryption for data transmission.
    - User data should be stored securely, with appropriate encryption for sensitive information.
    - Role-based access control should be implemented to ensure proper authorization.
3. **Usability Standards:**
    - The system should have an intuitive and user-friendly interface.
    - The system should be accessible and meet WCAG (Web Content Accessibility Guidelines) standards.
    - Comprehensive user documentation and help resources should be provided.

2.3.3 Stakeholder Analysis

2.3.3.1 User Needs

- **Ease of Use:**

  The system should provide an intuitive interface that allows users to easily browse products, place orders, and track their status.

- **Efficiency:**

  The order placement and tracking process should be streamlined to save time and effort for university employees.
- **Support:**

  Users should have access to support resources and customer service for any issues they encounter.

2.3.3.2 Admin Needs:

- **Effective Management:**

  The system should allow admins to efficiently manage users, products, and orders.
- **Control and Oversight:**

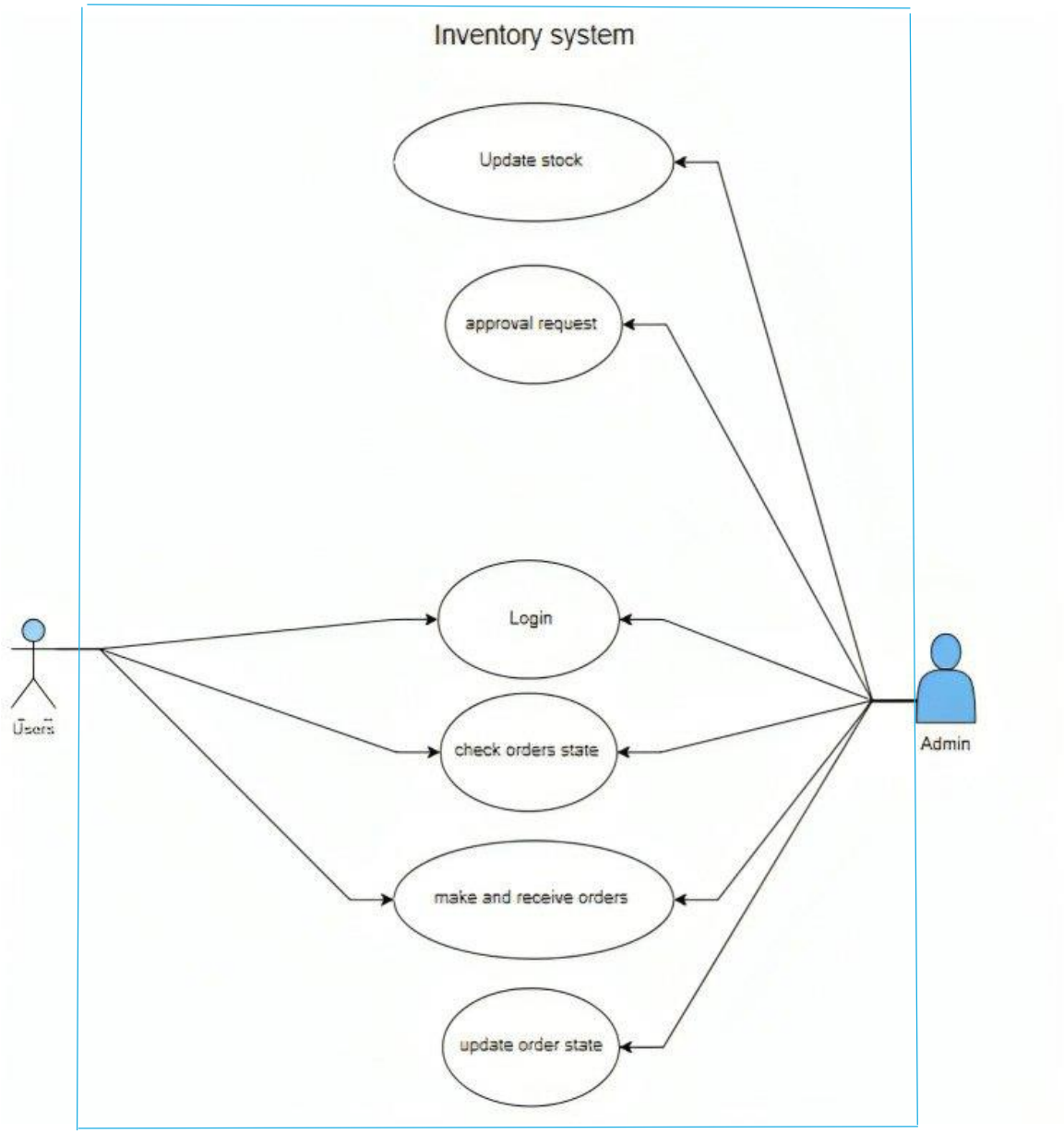
  Admins should have the tools necessary to monitor system performance, user activity, and inventory levels.
- **Scalability:**

  The system should be scalable to handle growth in the number of users, products, and orders.
- **Security:**

  Admins need robust security features to protect user data and ensure compliance with data protection regulations.

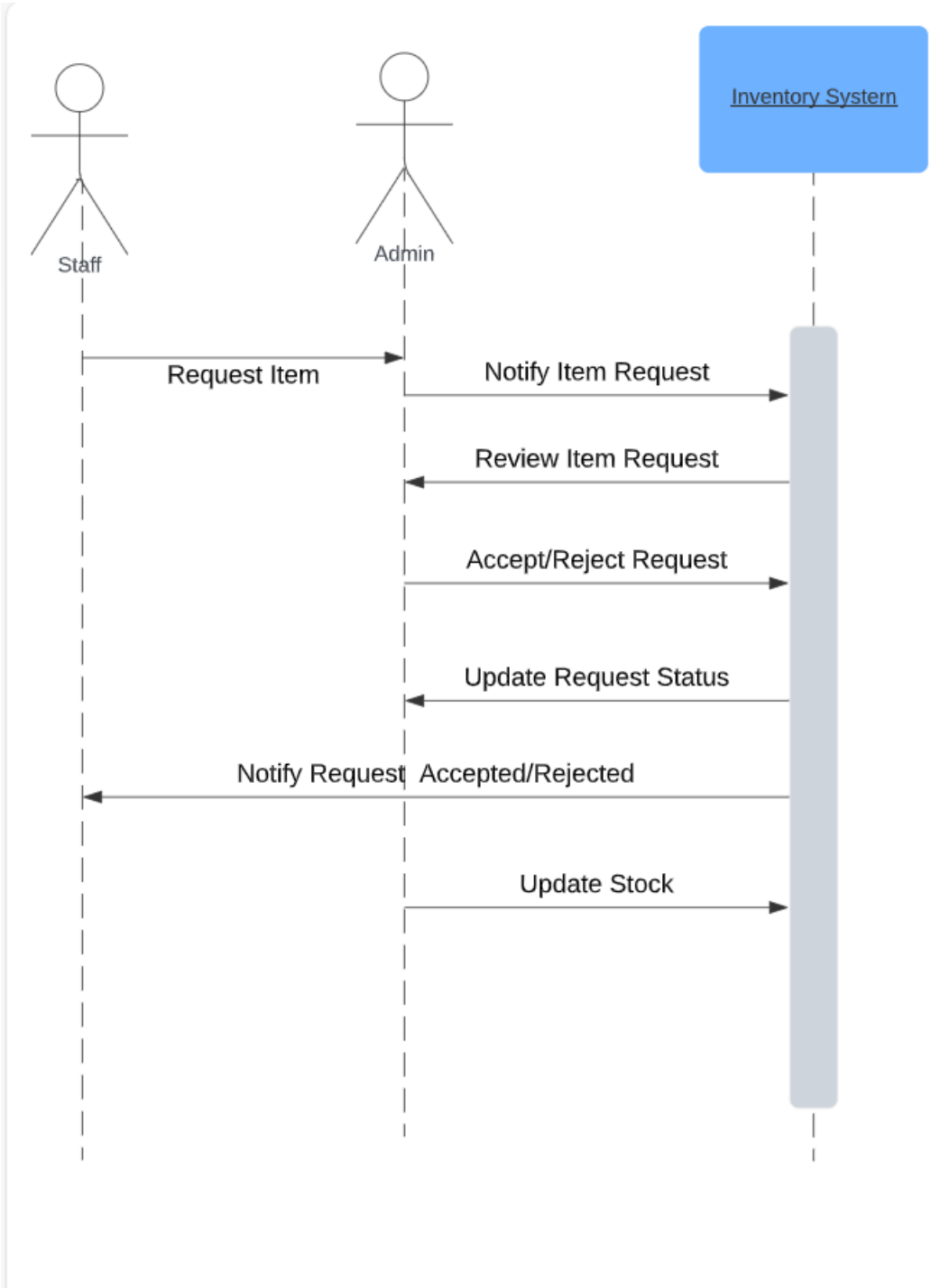
2.4 DIAGRAMS:

2.4.1 Use Case Diagram



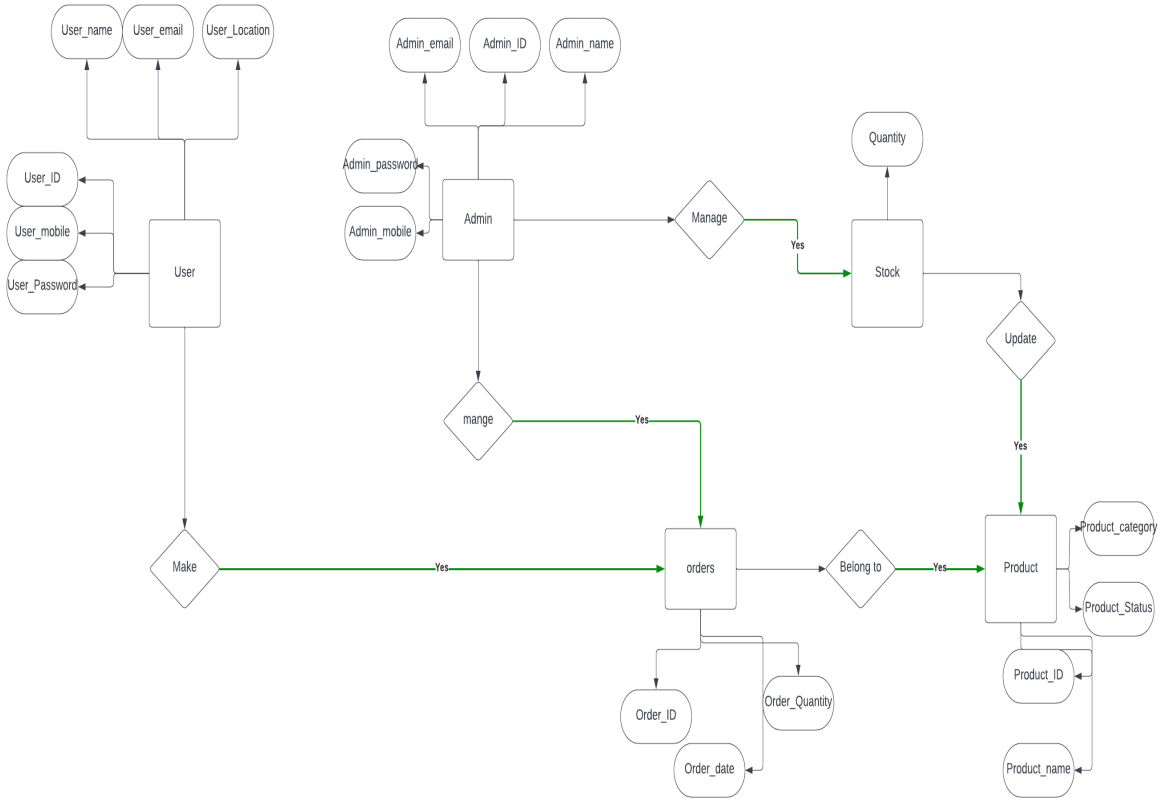
A Use Case Diagram is a type of UML diagram that represents the functional requirements of a system. It shows the interactions between various actors (users or other systems) and the system itself, capturing the key functionalities that the system must support. Use Case Diagrams are useful for understanding the system's requirements and the high-level interactions that occur.

2.4.2 Sequence Diagram



A Sequence Diagram is a type of UML diagram that illustrates how objects interact in a particular sequence of events. It focuses on the order of messages exchanged between objects and the timing of these messages. Sequence diagrams are particularly useful for detailing the interactions that occur in a specific use case or scenario

2.4.3 ER Diagram



An ER diagram, or Entity-Relationship diagram, is a visual representation of data within a database that shows how entities (things or objects) relate to each other. It's a conceptual and often logical model used in database design. Here's a breakdown of key components typically found in an ER diagram:

1- **Entities**:

   These are objects or concepts in the real world that are represented in the database. For example, in a university database, entities could include Student, Course, and Instructor.

2- **Attributes**:

   These are properties or characteristics of entities. For instance, a Student entity might have attributes like StudentID, Name, and DateOfBirth.

3- **Relationships**:

   These describe how entities are related to each other. Relationships are usually verbs or phrases that connect entities. For example, a student "enrolls in" a course, which is a relationship between Student and Course entities.

4- **Cardinality**:

   This defines the numerical constraints between entities in a relationship. It specifies how many instances of one entity are associated with instances of another entity. Common cardinalities include one-to-one, one-to-many, and many-to-many.

5- **Primary Keys**:

   These uniquely identify each record (instance) in an entity. They are essential for database operations like retrieval and updating of data.

6- **Foreign Keys**:

   These are attributes in one entity that refer to the primary key in another related entity. They establish relationships between tables in a relational database.

ER diagrams are crucial in the database design process as they provide a clear and concise way to communicate the structure of a database. They help database designers visualize and plan how data should be stored and accessed, ensuring efficient and effective database management.

CHAPTER 3: Front-End Development

3.1 INTRODUCTION TO FRONTEND DEVELOPMENT FOR UNIVERSITY INVENTORY MANAGEMENT SYSTEM

Frontend development for the University Inventory Management System focuses on creating an intuitive and user-friendly interface that enables university employees and administrators to interact with the system seamlessly. The front end serves as the visual layer through which users access and utilize the system's functionalities, such as browsing products, placing orders, and managing inventory.

3.2 KEY COMPONENTS OF FRONTEND DEVELOPMENT:

1- **User Interface Design:**

The user interface design plays a crucial role in the front-end development process. It involves creating visually appealing layouts, intuitive navigation menus, and interactive elements that enhance the user experience.

2- **Responsive Design:**

Given the diverse range of devices used by university employees and administrators, including desktops, laptops, tablets, and smartphones, it's essential to ensure that the frontend is responsive and adapts seamlessly to different screen sizes and resolutions

3- **Accessibility:**

Accessibility is a critical consideration in front-end development, ensuring that the system is usable by individuals with disabilities. This involves implementing features such as keyboard navigation, screen reader compatibility, and high contrast options.

4- **Integration with Backend Services**:

The frontend interacts with backend services, such as APIs, to fetch and display data, process user requests, and perform various actions. This integration ensures that the frontend remains synchronized with the backend and provides real-time updates to users.

5- **Authentication and Authorization:**

Frontend development includes implementing authentication and authorization mechanisms to ensure that only authorized users can access specific features and data within the system. This involves designing login screens, password recovery flows, and role-based access controls.

6- **Data Visualization:**

Data visualization is an essential aspect of frontend development, enabling users to interpret and analyze information effectively. This may involve creating charts, graphs, and dashboards to present inventory statistics, order trends, and other relevant data.

3.3 TECHNOLOGIES:

- **HTML (Hypertext Markup Language):**

  HTML forms the foundation of the frontend, providing the structure and content of web pages. It defines the layout of elements such as headers, navigation menus, forms, and tables.

- **CSS (Cascading Style Sheets):**

  CSS is used to style and design the HTML elements, including colors, fonts, spacing, and layout. It ensures consistency and visual appeal across different parts of the user interface.

- **JavaScript:**

  JavaScript adds interactivity and dynamic behavior to the frontend. It enables features such as form validation, real-time updates, and interactive elements like dropdown menus and sliders.

- **Tailwind CSS:**

  Tailwind CSS is a utility-first CSS framework that provides a set of pre-built utility classes for styling HTML elements. It allows for rapid development and customization of styles without the need for writing custom CSS.

- **React:**

  React is a JavaScript library for building user interfaces, particularly single-page applications. It enables the creation of reusable UI components, state management, and efficient rendering through the use of a virtual DOM.

3.4 FRONTEND WITH REACT

Front end development with React for an Inventory Management System involves creating user interfaces that are interactive, responsive, and visually appealing.

React is a popular JavaScript library for building user interfaces, and it allows developers to create reusable components that can be easily managed and updated.

3.5 COMPONENT AND LAYOUT SCREENS

3.5.1 Login

3.5.2 Add user

```
import Swal from "sweetalert2";
// Add new users
export const AddUser = async () => {
  const { value: inputValue } = await Swal.fire({
    title: "Add New User",

    html: `
    <input  placeholder="Enter user ID" id="swal-input1" class="swal2-input  max-w-md bg-white rounded-lg shadow-md " >
    <input type="password" placeholder="Enter the password"  id="swal-input2" class="swal2-input  max-w-md bg-white rounded-lg shadow-md ">
    <div >
      <select  id="swal-input4" class="swal2-input  max-w-md bg-white rounded-lg shadow-md  " style="width: 57%;"><option value="Admin">Admin</option><option value="User">User</option></select>
    </div>
    <input type="number" placeholder="Enter phone number"  id="swal-input3" class="swal2-input  max-w-md bg-white rounded-lg shadow-md ">
    <input type="password" placeholder="Enter the stutas"  id="swal-input4" class="swal2-input  max-w-md bg-white rounded-lg shadow-md ">
    <input type="password" placeholder="Enter the password"  id="swal-input5" class="swal2-input  max-w-md bg-white rounded-lg shadow-md ">
    <input type="password" placeholder="Enter the Location"  id="swal-input5" class="swal2-input  max-w-md bg-white rounded-lg shadow-md ">

    `,
    showCancelButton: true,
  });
};
```

3.5.3 Index

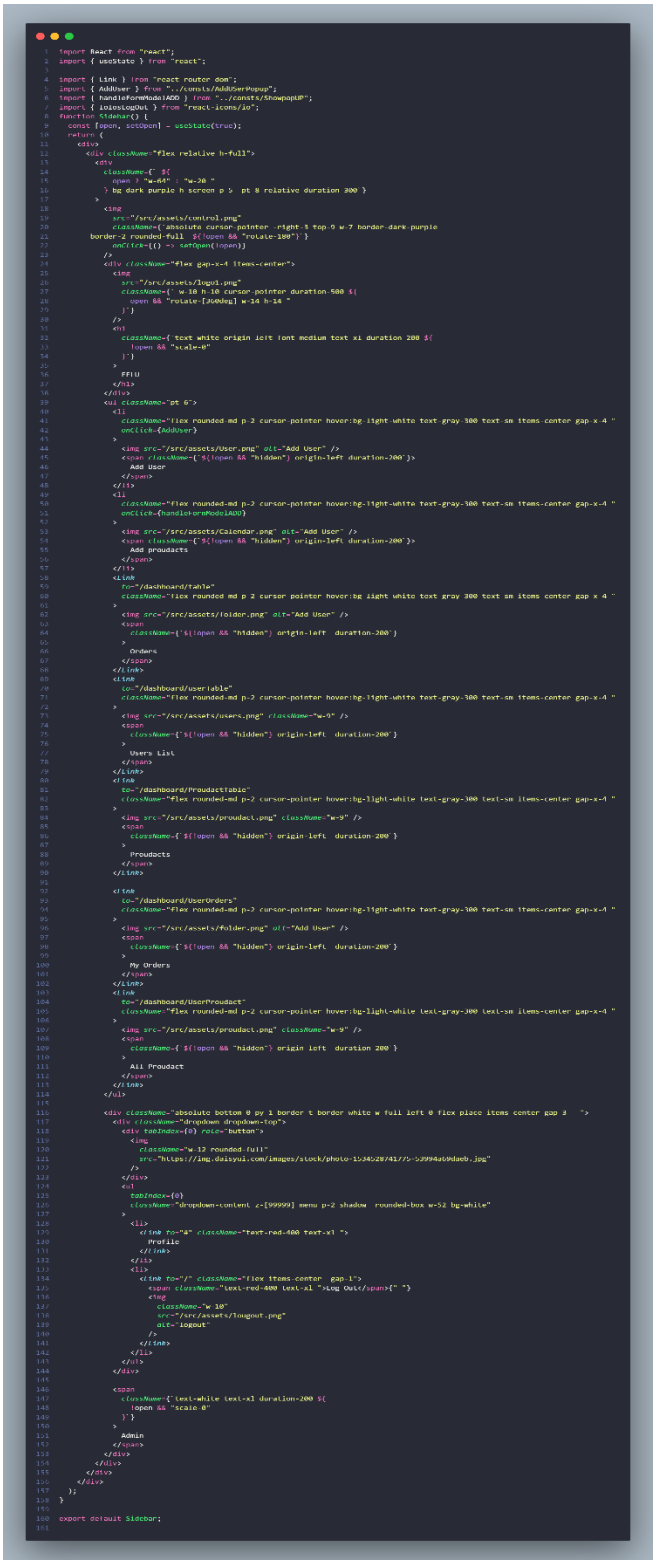
```jsx
import React from "react";
import { Route, Routes } from "react-router-dom";
import Sidebar from "../components/sidebar";
import Table from "../components/table";
import Updeteorder from "../components/updeteorder";
import TableAllUser from "../components/TableAllUser";
import ProudactTable from "../components/ProudactTable";
import UserOrders from "../components/myOrder";
import UserProudact from "../components/userProudact";
import Updeteuser from "../components/updateUser";

const Index = () => {
  return (
    <>
      <div className="w-full flex gap-1 bg-[#dfdfdf]">
        <Sidebar />
        <Routes>
          <Route path="UserOrders" element={<UserOrders />} />
          <Route path="UserProudact" element={<UserProudact />} />
          <Route path="table" element={<Table />} />
          <Route path="userTable" element={<TableAllUser />} />
          <Route path="UpdeteOrder" element={<Updeteorder />} />
          <Route path="Updeteuser" element={<Updeteuser />} />
          <Route path="ProudactTable" element={<ProudactTable />} />
        </Routes>
      </div>
    </>
  );
};

export default Index;

```

3.5.4 sidebar



```jsx
import React from "react";
import { useState } from "react";

import { Link } from "react-router-dom";
import { AddUser } from "../consts/AddUserPopup";
import { handleFormModelADD } from "../consts/ShowpopUP";
import { IoIosLogOut } from "react-icons/io";
function Sidebar() {
  const [open, setOpen] = useState(true);
  return (
    <div>
      <div classNameName="flex relative h-full">
        <div
          classNameName={` ${
            open ? "w-64" : "w-20 "
          } bg-dark-purple h-screen p-5  pt-8 relative duration-300 `}
        >
          <img
            src="/src/assets/control.png"
            classNameName={`absolute cursor-pointer -right-3 top-9 w-7 border-dark-purple
          border-2 rounded-full  ${!open && "rotate-180"} `}
            onClick={() => setOpen(!open)}
          />
          <div classNameName="flex gap-x-4 items-center">
            <img
              src="/src/assets/logo1.png"
              classNameName={` w-10 h-10 cursor-pointer duration-500 ${
                open && "rotate-[360deg] w-14 h-14 "
              }`}
            />
            <h1
              classNameName={`text-white origin-left font-medium text-xl duration-200 ${
                !open && "scale-0"
              }`}
            >
              FFIU
            </h1>
          </div>
          <ul classNameName="pt-6">
            <li
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
              onClick={AddUser}
            >
              <img src="/src/assets/User.png" alt="Add User" />
              <span classNameName={`${!open && "hidden"} origin-left duration-200`}>
                Add User
              </span>
            </li>
            <li
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
              onClick={handleFormModelADD}
            >
              <img src="/src/assets/Calendar.png" alt="Add User" />
              <span classNameName={`${!open && "hidden"} origin-left duration-200`}>
                Add Proudacts
              </span>
            </li>
            <Link
              to="/dashboard/table"
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
            >
              <img src="/src/assets/folder.png" alt="Add User" />
              <span
                classNameName={`${!open && "hidden"} origin-left  duration-200`}
              >
                Orders
              </span>
            </Link>
            <Link
              to="/dashboard/userTable"
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
            >
              <img src="/src/assets/users.png" classNameName="w-9" />
              <span
                classNameName={`${!open && "hidden"} origin-left  duration-200`}
              >
                Users List
              </span>
            </Link>
            <Link
              to="/dashboard/ProudactTable"
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
            >
              <img src="/src/assets/proudact.png" classNameName="w-9" />
              <span
                classNameName={`${!open && "hidden"} origin-left  duration-200`}
              >
                Proudacts
              </span>
            </Link>

            <Link
              to="/dashboard/UserOrders"
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
            >
              <img src="/src/assets/folder.png" alt="Add User" />
              <span
                classNameName={`${!open && "hidden"} origin-left  duration-200`}
              >
                My Orders
              </span>
            </Link>
            <Link
              to="/dashboard/UserProudact"
              classNameName="flex rounded-md p-2 cursor-pointer hover:bg-light-white text-gray-300 text-sm items-center gap-x-4 "
            >
              <img src="/src/assets/proudact.png" classNameName="w-9" />
              <span
                classNameName={` ${!open && "hidden"} origin-left  duration-200 }
              >
                All Proudact
              </span>
            </Link>
          </ul>

          <div classNameName="absolute bottom-0 py-1 border-t border-white w-full left-0 flex place-items-center gap-3   ">
            <div classNameName="dropdown dropdown-top">
              <div tabIndex={0} role="button">
                <img
                  classNameName="w-12 rounded-full"
                  src="https://img.daisyui.com/images/stock/photo-1534528741775-53994a69daeb.jpg"
                />
              </div>
              <ul
                tabIndex={0}
                classNameName="dropdown-content z-[99999] menu p-2 shadow  rounded-box w-52 bg-white"
              >
                <li>
                  <Link to="#" classNameName="text-red-400 text-xl ">
                    Profile
                  </Link>
                </li>
                <li>
                  <Link to="/" classNameName="flex items-center  gap-1">
                    <span classNameName="text-red-400 text-xl ">Sing Out</span>{" "}
                    <img
                      classNameName="w-10"
                      src="/src/assets/logout.png"
                      alt="logout"
                    />
                  </Link>
                </li>
              </ul>
            </div>

            <span
              classNameName={`text-white text-xl duration-200 ${
                !open && "scale-0"
              }`}
            >
              Admin
            </span>
          </div>
        </div>
      </div>
    </div>
  );
}

export default Sidebar;
```

3.5.5 Delete user

```
// sure delete element
import Swal from "sweetalert2";
export const showPopupDlete = () => {
  Swal.fire({
    title: "Are you sure?",
    text: "You won't be able to revert this!",
    icon: "warning",
    showCancelButton: true,
    confirmButtonColor: "#3085d6",
    cancelButtonColor: "#d33",
    confirmButtonText: "Yes, delete it!",
  }).then((result) => {
    if (result.isConfirmed) {
      Swal.fire({
        title: "Deleted!",
        text: "Your file has been deleted.",
        icon: "success",
      });
    }
  });
};
```

3.5.6 Update product

```jsx
import React from "react";

function UserOrders() {
  return (
    <div className="w-full p-7 scrollbar scrollbar-thumb-sky-400 scrollbar-track-sky-100  ">
      <div clssName="flex items-center justify-between flex-wrap mt-9 gap-1 py-3 "></div>
      <h2 className="text-xl  font-semibold rounded py-1 text-center  text-white bg-indigo-600 my-5">
        ALL Order
      </h2>
      <div className="relative  overflow-x-auto shadow-md sm:rounded-lg">
        <table className="text-sm min-w-full overflow-x-scroll text-left rtl:text-right text-gray-500 dark:text-gray-400">
          <thead className="text-xs text-white uppercase bg-indigo-600 dark:bg-gray-700 dark:text-gray-400">
            <tr>
              <th scope="col" className="px-6 py-3">
                ID
              </th>
              <th scope="col" className="px-6 py-3">
                Order Name
              </th>
              <th scope="col" className="px-6 py-3">
                Date
              </th>
              <th scope="col" className="px-6 py-3">
                Stutes
              </th>
              <th scope="col" className="px-6 py-3">
                Admin Name
              </th>
            </tr>
          </thead>
          <tbody>
            <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
              <th
                scope="row"
                className="px-6 py-6 font-medium text-gray-900 whitespace-nowrap dark:text-white"
              >
                344 56 77
              </th>
              <td className="px-6  py-6 ">Laptop</td>
              <td className="px-6 py-6 ">28/5/2024</td>
              <td className="px-6 py-6 ">
                <span className="bg-green-600 text-white px-2 rounded-lg font-bold ">
                  Approved
                </span>
              </td>
              <td className="px-6 py-6 text-red-500 font-bold ">Nourhan</td>
            </tr>
            <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
              <th
                scope="row"
                className="px-6 py-6 font-medium text-gray-900 whitespace-nowrap dark:text-white"
              >
                344 56 77
              </th>
              <td className="px-6  py-6 ">Laptop</td>
              <td className="px-6 py-6 ">28/5/2024</td>
              <td className="px-6 py-6 ">
                <span className="bg-red-600 text-white px-1 rounded-lg font-bold">
                  Un Aproved
                </span>
              </td>
              <td className="px-6 py-6 text-red-500 font-bold ">Nourhan</td>
            </tr>
            <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
              <th
                scope="row"
                className="px-6 py-6 font-medium text-gray-900 whitespace-nowrap dark:text-white"
              >
                344 56 77
              </th>
              <td className="px-6  py-6 ">Laptop</td>
              <td className="px-6 py-6 ">28/5/2024</td>
              <td className="px-6 py-6 ">
                <span className="bg-green-600 text-white px-2 rounded-lg font-bold ">
                  Approved
                </span>
              </td>
              <td className="px-6 py-6 text-red-500 font-bold ">Nourhan</td>
            </tr>

          </tbody>
        </table>
      </div>
    </div>
  );
}

export default UserOrders;

```

3.5.7 Product table

```
import React from "react";
import { FaEdit } from "react-icons/fa";
import { MdDelete } from "react-icons/md";
import { Link } from "react-router-dom";
import { showPopupDlete } from "../consts/ShowpopUP";
function ProudactTable() {
  return (
    <div className="w-full">
      <div className="w-full p-7 scrollbar scrollbar-thumb-sky-400 scrollbar-track-sky-100  ">
        <div className="flex items-center justify-between flex-wrap mt-9 gap-1 py-3 "></div>
        <h2 className="text-xl flex items-center justify-center gap-2 font-semibold rounded py-1 text-center  text-white bg-indigo-600 my-5">
          Proudacts
          <img className="w-9" src="/src/assets/shop.png" alt="" />
        </h2>
        <div className="relative  overflow-x-auto shadow-md sm:rounded-lg">
          <table className="text-sm min-w-full overflow-x-scroll text-left rtl:text-right text-gray-500 dark:text-gray-400">
            <thead className="text-xs text-white uppercase bg-indigo-600 dark:bg-gray-700 dark:text-gray-400">
              <tr>
                <th scope="col" className="px-6 py-3">
                  Name
                </th>
                <th scope="col" className="px-6 py-3">
                  Quantity
                </th>
                <th scope="col" className="px-6 py-3">
                  Category
                </th>
                <th scope="col" className="px-6 py-3">
                  Admin Name
                </th>
                <th scope="col" className="px-6 py-3">
                  Action
                </th>
              </tr>
            </thead>
            <tbody>
              <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
                <th
                  scope="row"
                  className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"
                >
                  Apple MacBook Pro 17"
                </th>
                <td className="px-6 py-4">Silver</td>
                <td className="px-6 py-4">Laptop</td>
                <td className="px-6 py-4">mohamed ahmed</td>
                <td className="px-6  pt-7 flex items-center justify-between text-black text-lg text-right">
                  <Link to="/dashboard/UpdeteOrder">
                    <FaEdit className=" hover:text-green-500 hover:scale-125  duration-700 cursor-pointer " />
                  </Link>
                  <MdDelete
                    className=" hover:text-red-600 hover:scale-125 duration-700  cursor-pointer"
                    onClick={showPopupDlete}
                  />
                </td>
              </tr>
              <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
                <th
                  scope="row"
                  className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"
                >
                  Apple MacBook Pro 17"
                </th>
                <td className="px-6 py-4">Silver</td>
                <td className="px-6 py-4">Laptop</td>
                <td className="px-6 py-4">mohamed ahmed</td>
                <td className="px-6  pt-7 flex items-center justify-between text-black text-lg text-right">
                  <Link to="/dashboard/UpdeteOrder">
                    <FaEdit className=" hover:text-green-500 hover:scale-125  duration-700 cursor-pointer " />
                  </Link>
                  <MdDelete
                    className=" hover:text-red-600 hover:scale-125 duration-700  cursor-pointer"
                    onClick={showPopupDlete}
                  />
                </td>
              </tr>
              <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
                <th
                  scope="row"
                  className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"
                >
                  Apple MacBook Pro 17"
                </th>
                <td className="px-6 py-4">Silver</td>
                <td className="px-6 py-4">Laptop</td>
                <td className="px-6 py-4">mohamed ahmed</td>
                <td className="px-6  pt-7 flex items-center justify-between text-black text-lg text-right">
                  <Link to="/dashboard/UpdeteOrder">
                    <FaEdit className=" hover:text-green-500 hover:scale-125  duration-700 cursor-pointer " />
                  </Link>
                  <MdDelete
                    className=" hover:text-red-600 hover:scale-125 duration-700  cursor-pointer"
                    onClick={showPopupDlete}
                  />
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  );
}

export default ProudactTable;
```

3.5.8 Product catalogue

```jsx
import React from "react";
import { IoIosAddCircle } from "react-icons/io";
import { AddedDone } from "../consts/ShowpopUP";

function UserProudact() {
  return (
    <div className="w-full">
      <div className="w-full p-7 scrollbar scrollbar-thumb-sky-400 scrollbar-track-sky-100  ">
        <div className="flex items-center justify-between flex-wrap mt-9 gap-1 py-3 "></div>
        <h2 className="text-xl flex items-center justify-center gap-2 font-semibold rounded py-1 text-center  text-white bg-indigo-600 my-5">
          All Proudacts
          <img className="w-9" src="/src/assets/shop.png" alt="" />
        </h2>
        <div className="relative  overflow-x-auto shadow-md sm:rounded-lg">
          <table className="text-sm min-w-full overflow-x-scroll text-left rtl:text-right text-gray-500 dark:text-gray-400">
            <thead className="text-xs text-white uppercase bg-indigo-600 dark:bg-gray-700 dark:text-gray-400">
              <tr>
                <th scope="col" className="px-6 py-3">
                  id
                </th>
                <th scope="col" className="px-6 py-3">
                  Proudct name
                </th>
                <th scope="col" className="px-6 py-3">
                  countity
                </th>

                <th scope="col" className="px-6 py-3">
                  ADD
                </th>
              </tr>
            </thead>
            <tbody>
              <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
                <th
                  scope="row"
                  className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"
                >
                  Apple MacBook Pro 17"
                </th>
                <td className="px-6 py-4">Silver</td>
                <td className="px-6 py-4">Laptop</td>

                <td className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white">
                  <button
                    onClick={AddedDone}
                    className="flex items-center justify-center gap-1 hover:brightness-110 hover:animate-pulse font-bold py-3 px-6 rounded-full bg-gradient-to-r from-blue-500 to-pink-500 text-white"
                  >
                    <span className="font-medium">ADD</span>
                    <IoIosAddCircle className="text-2xl" />
                  </button>
                </td>
              </tr>
              <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
                <th
                  scope="row"
                  className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"
                >
                  Apple MacBook Pro 17"
                </th>
                <td className="px-6 py-4">Silver</td>
                <td className="px-6 py-4">Laptop</td>

                <td className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white">
                  <button
                    onClick={AddedDone}
                    className="flex items-center justify-center gap-1 hover:brightness-110 hover:animate-pulse font-bold py-3 px-6 rounded-full bg-gradient-to-r from-blue-500 to-pink-500 text-white"
                  >
                    <span className="font-medium">ADD</span>
                    <IoIosAddCircle className="text-2xl" />
                  </button>
                </td>
              </tr>
              <tr className="bg-white border-b dark:bg-gray-800 dark:border-gray-700 hover:bg-gray-50 dark:hover:bg-gray-600">
                <th
                  scope="row"
                  className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white"
                >
                  Apple MacBook Pro 17"
                </th>
                <td className="px-6 py-4">Silver</td>
                <td className="px-6 py-4">Laptop</td>

                <td className="px-6 py-4 font-medium text-gray-900 whitespace-nowrap dark:text-white">
                  <button
                    onClick={AddedDone}
                    className="flex items-center justify-center gap-1 hover:brightness-110 hover:animate-pulse font-bold py-3 px-6 rounded-full bg-gradient-to-r from-blue-500 to-pink-500 text-white"
                  >
                    <span className="font-medium">ADD</span>
                    <IoIosAddCircle className="text-2xl" />
                  </button>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  );
}

export default UserProudact;
```

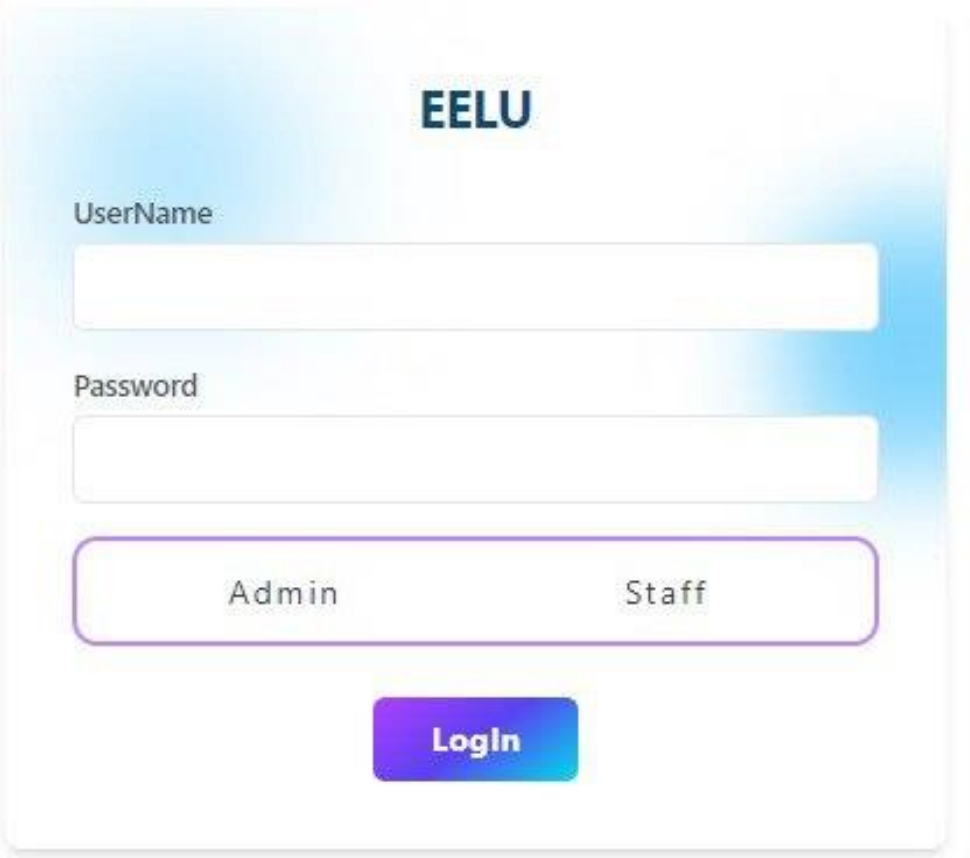
CHAPTER 4: User Interface

4.1 INTRODUCTION

User Interface (UI) design for the University Inventory Management System focuses on creating an intuitive, visually appealing, and efficient interface that enables users to interact with the system seamlessly. The UI serves as the bridge between users and the functionality of the system, providing an accessible and engaging experience for both employees and administrators.

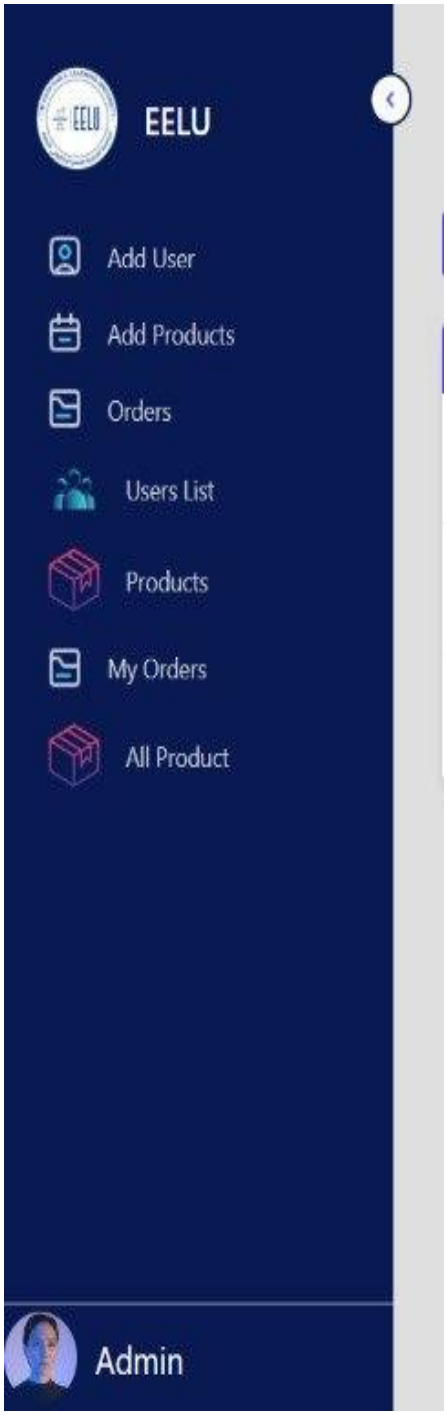
4.2 PAGES

4.2.1 Login page



The login interface for the University Inventory Management System serves as the entry point for users, allowing them to securely access their accounts and utilize the system's functionalities. The design of the login interface should prioritize usability, security, and accessibility to ensure a seamless and efficient login experience for employees and administrators.

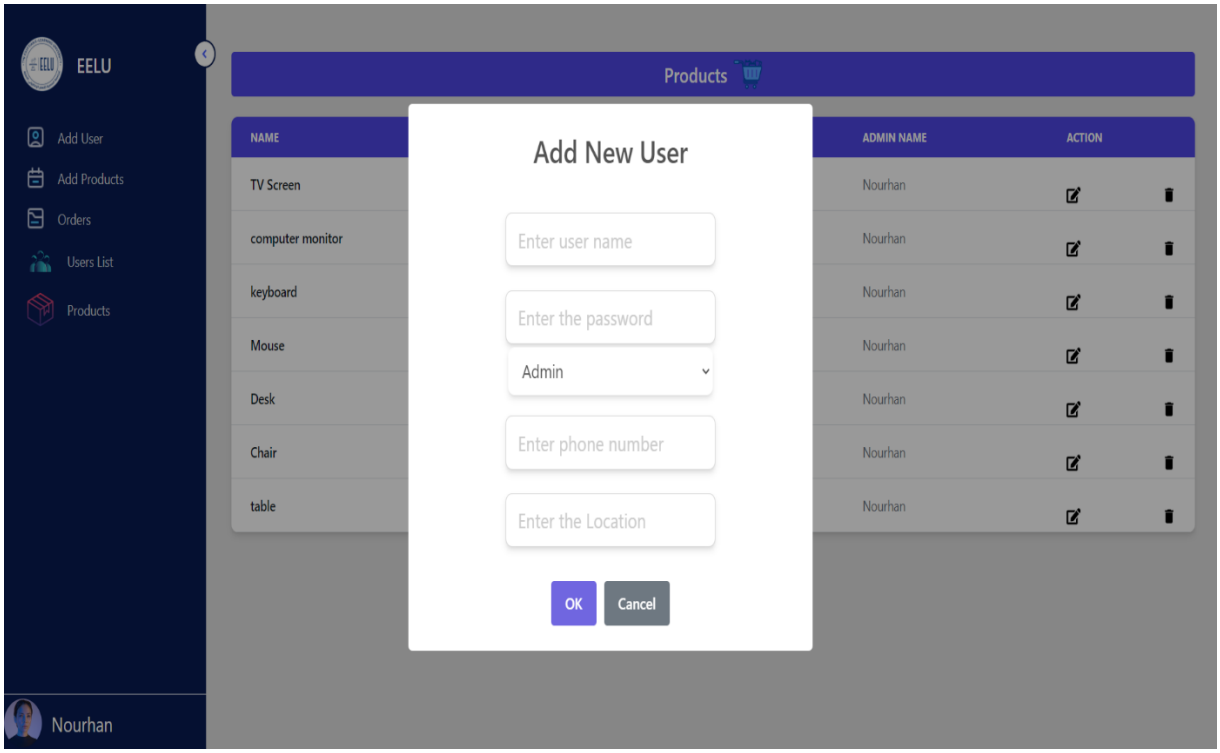
4.2.2 Sidebar



The sidebar interface design for the University Inventory Management System serves as a navigation hub, allowing users to access different sections and functionalities of the system quickly and efficiently. The design of the sidebar should prioritize ease of use, organization, and accessibility to enhance user productivity and facilitate seamless navigation within the system.

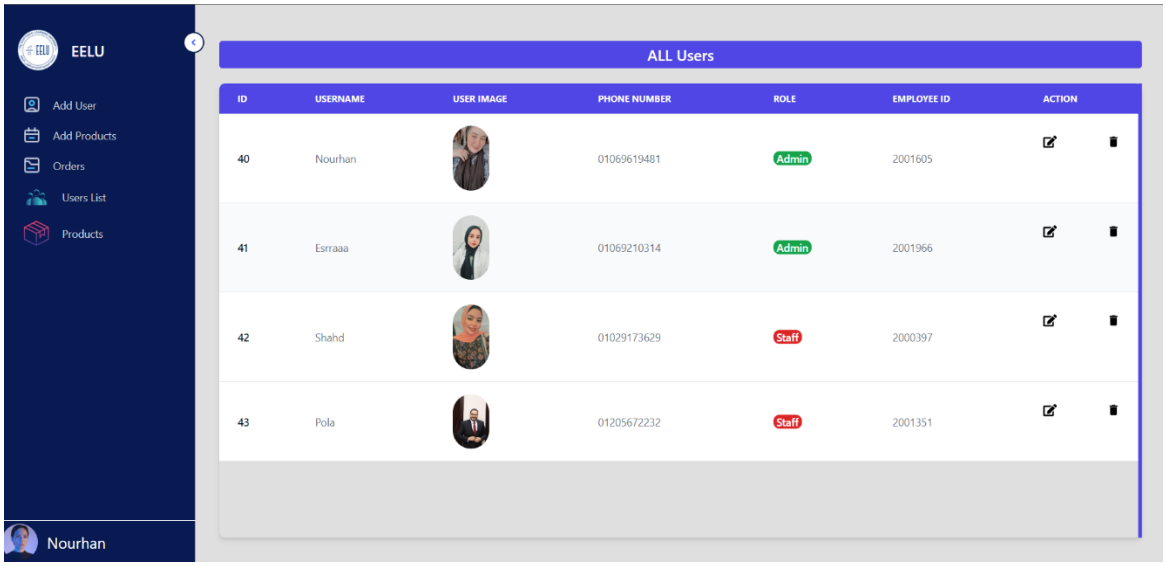
4.2.3 Admin view

4.2.3.1 Add new users

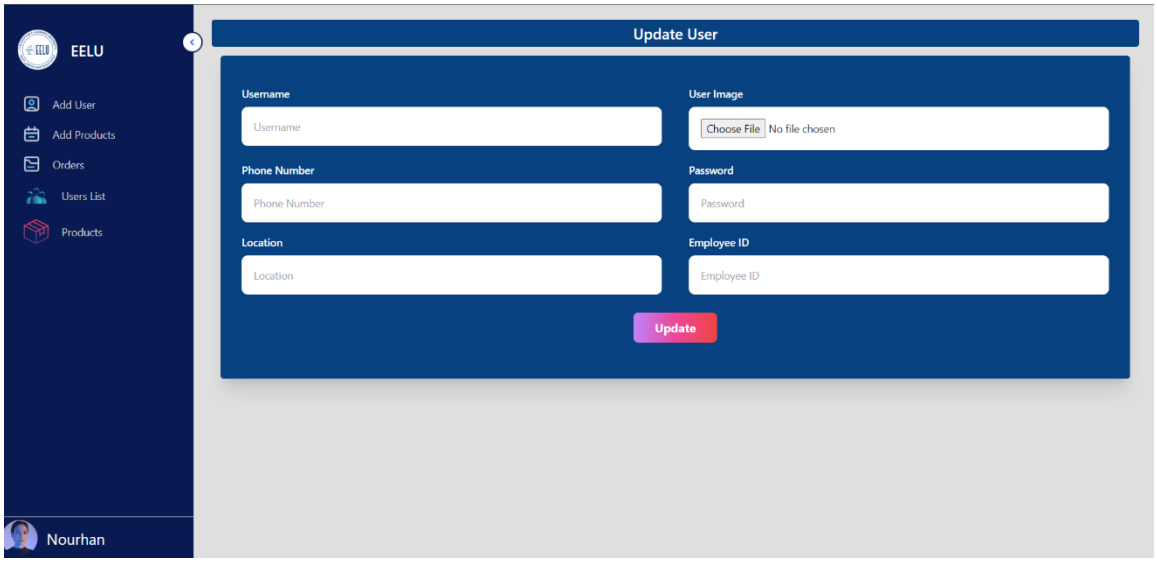


The "Add User" in the University Inventory Management System serves as a user interface for administrators to create new user accounts within the system. This page should provide administrators with intuitive controls and clear guidance to facilitate the process of adding new users effectively.

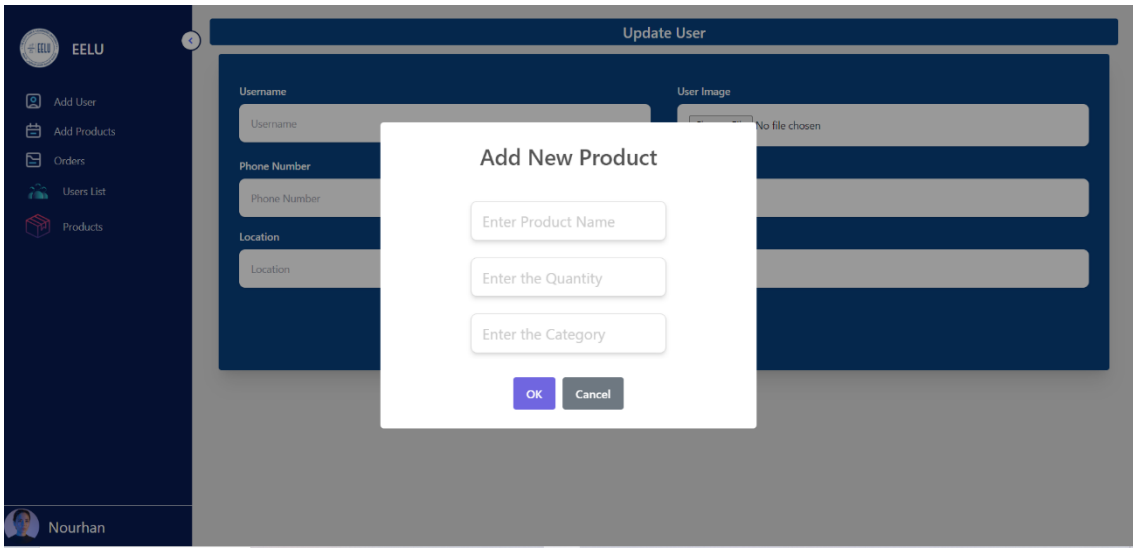
4.2.3.2 User info



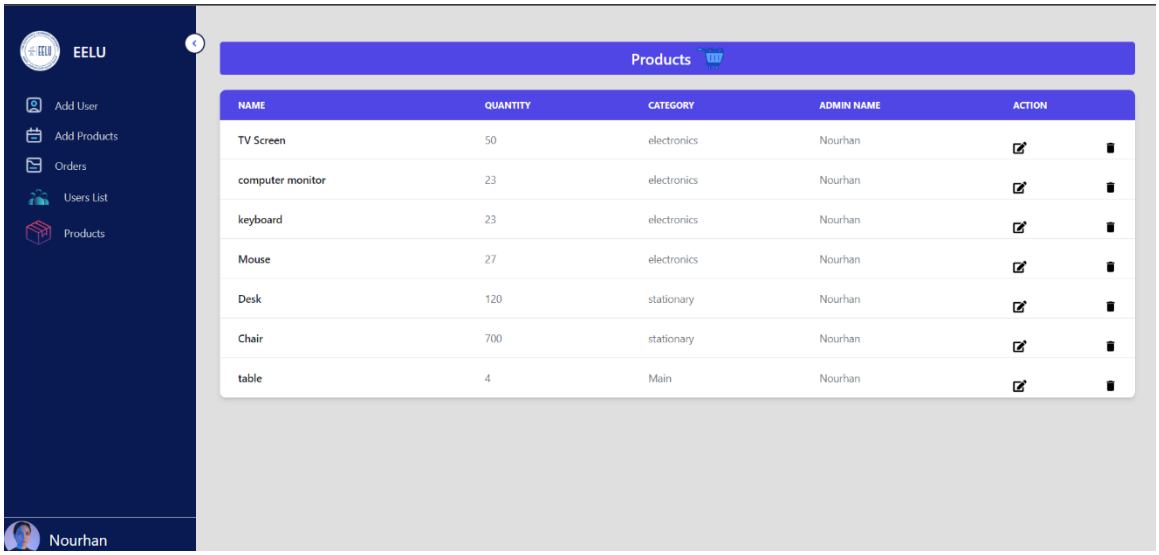
4.2.3.3 Update user info



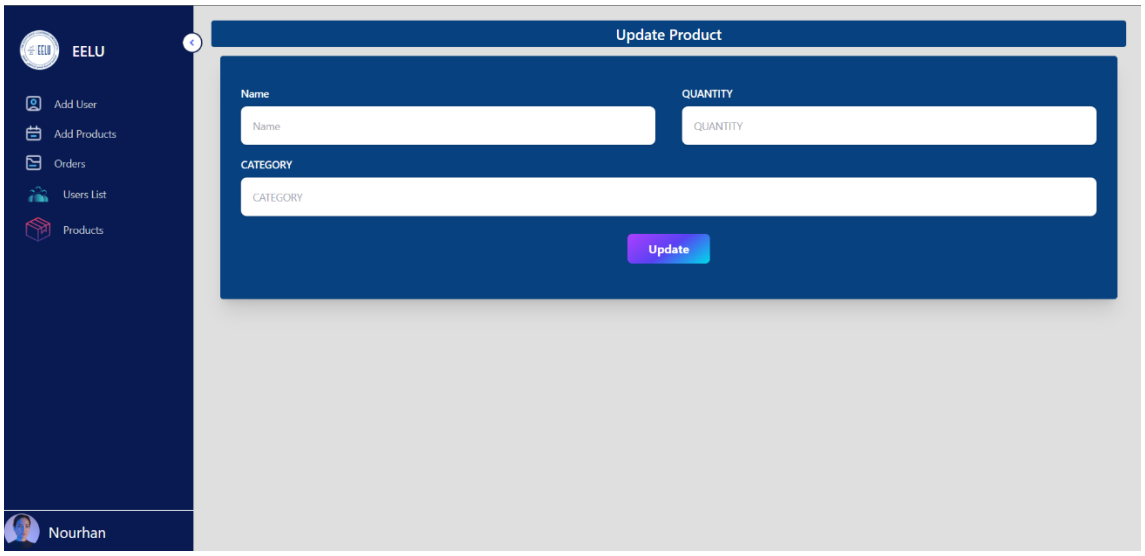
4.2.3.4 Add New Product



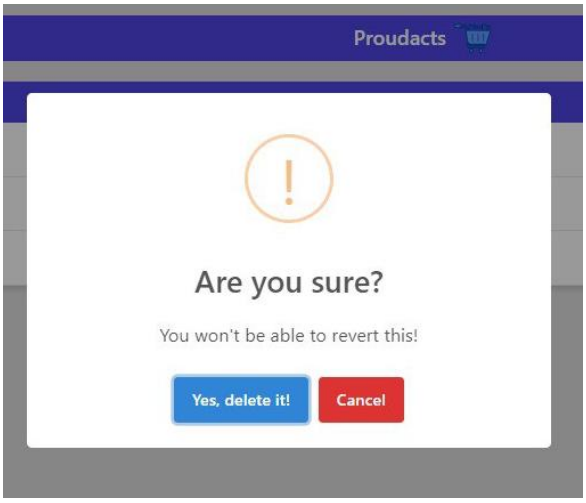
4.2.3.5 Product info

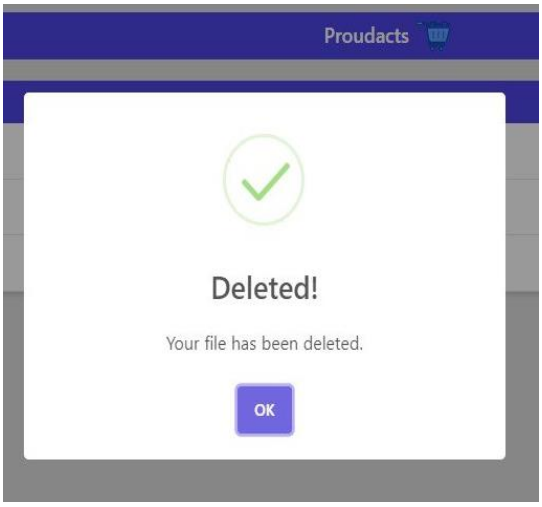


4.2.3.6 Update product info

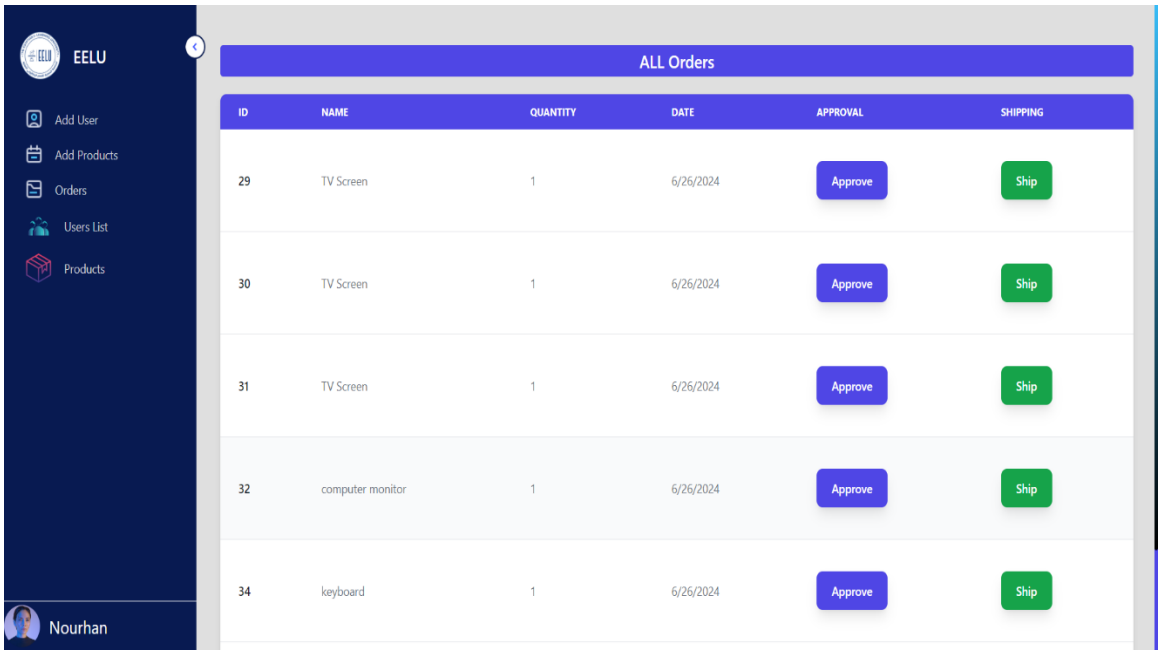


4.2.3.7 Sweet alert





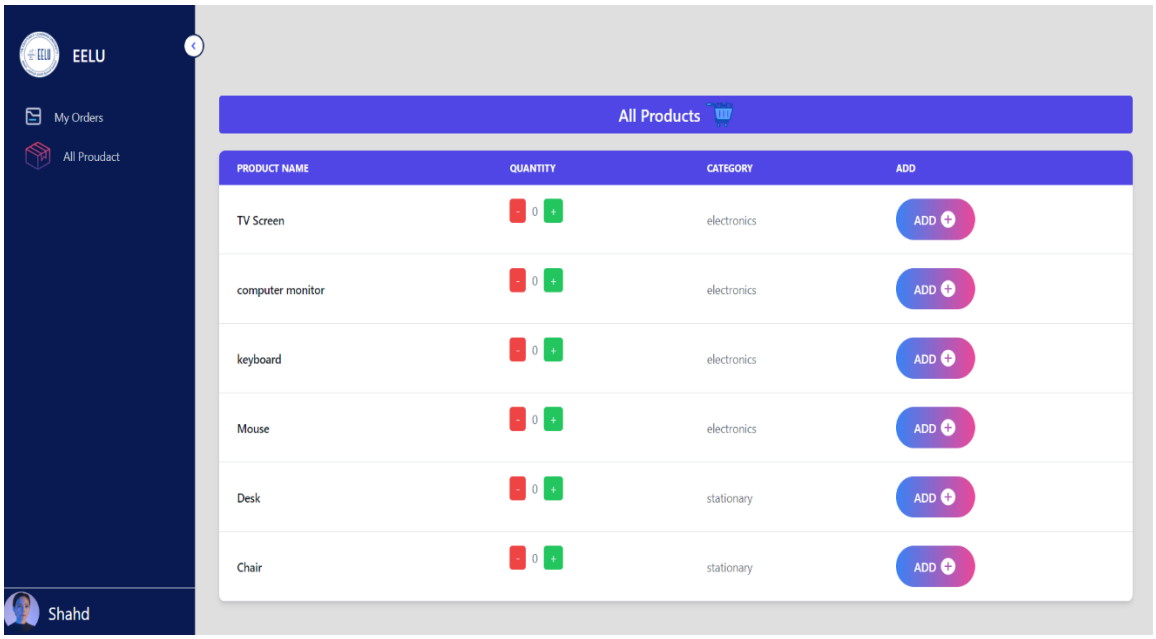
4.2.3.8 Order management



In the University Inventory Management System, the "Orders Management" interface provides administrators with a comprehensive view of all pending orders, allowing them to review, approve, or reject orders as necessary. This interface plays a crucial role in facilitating efficient order processing and ensuring smooth inventory management operations.

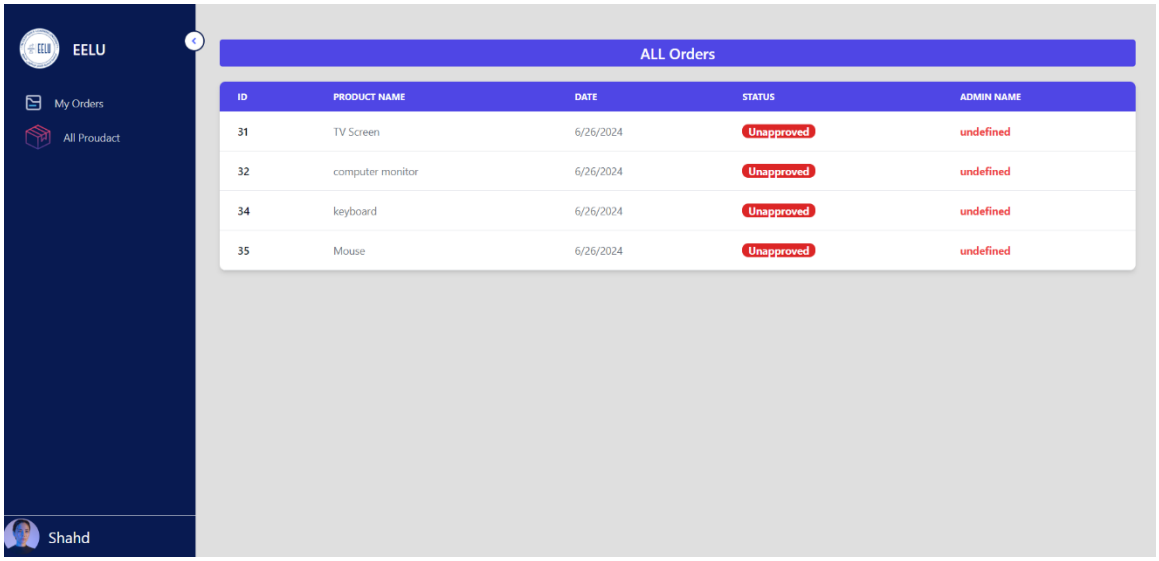
4.2.4 User view

4.2.4.1 Product catalogue



in the University Inventory Management System, the "Product Catalog" interface provides users with a comprehensive view of all available products in the inventory, enabling them to browse, select, and add products to their orders. This interface plays a crucial role in facilitating efficient order placement and ensuring users have access to the necessary supplies for their work.

4.2.4.2 My order info



CHAPTER 5: Back-End Development

5.1 INTRODUCTION

Back-end development refers to creating the underlying infrastructure and logic for a web application or website. In other words, it encompasses everything that the user doesn't directly see on the screen **such as**:

- **Databases:**
  Storing and retrieving data.
- **Business algorithms:**
  Processing data and enforcing business rules.
- **APIs (Application Programming Interfaces):**
  Providing endpoints for interaction with other applications.
- **Server-side services:**
  Managing tasks like authentication and real-time payment processing.

5.2 COMMON BACK-END PROGRAMMING LANGUAGES

The backend of the University Inventory Management System is built using Python and the Django framework, with a focus on creating a robust and efficient API to facilitate seamless communication between the frontend and backend components.

Python is chosen for its simplicity, readability, and extensive ecosystem, making it an ideal language for backend development. Django, a high-level web framework, provides a comprehensive set of tools and features that accelerate the development process, enforce security, and promote scalability and maintainability.

5.3 KEY FEATURES OF THE BACKEND

- **Data Management:**
  The backend handles all aspects of data management, including storage, retrieval, and manipulation of inventory data. It ensures data integrity and consistency, supporting complex queries and transactions required for inventory operations.
- **Business Logic:**
  The backend is responsible for executing the business logic of the inventory management system. This includes processing orders, managing user roles and permissions, and

enforcing rules related to inventory levels and product availability.

- **User Authentication and Authorization:**

  Django's built-in authentication system is utilized to manage user accounts, authenticate users, and control access to various parts of the system. This ensures that only authorized personnel can perform sensitive operations such as adding products or approving orders.

- **API Development:**

  A RESTful API is developed using Django REST framework to enable communication between the frontend and backend. The API provides endpoints for various operations such as viewing products, placing orders, approving orders, and managing inventory. This separation of concerns allows for a modular and flexible architecture.

- **Security:**

  Django's security features, such as protection against SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF), are leveraged to ensure that the system is secure from common web vulnerabilities.

- **Scalability and Performance:**

  The backend is designed to be scalable, capable of handling increasing loads as the university's inventory needs grow. Performance optimization techniques, such as efficient database queries and caching, are employed to ensure fast response times.

5.4 API UNIVERSITY INVENTORY MANAGEMENT SYSTEM

An Application Programming Interface (API) is a crucial component of the University Inventory Management System, enabling seamless communication between the frontend and backend. The API serves as a bridge, allowing the frontend to interact with backend services to perform various operations such as retrieving product information, placing orders, and managing inventory. For this project, a RESTful API is developed using Django REST framework.

5.5 KEY ASPECTS OF THE API:

5.5.1 Purpose and Functionality:

The primary purpose of the API is to provide a standardized way for the frontend to interact with the backend. It exposes endpoints for various CRUD (Create, Read, Update, Delete) operations, enabling users to view products, place orders, manage user accounts, and perform administrative tasks.

5.5.2 RESTful Design:

The API follows REST (Representational State Transfer) principles, which include:

- **Statelessness:**
  Each API call from the frontend is independent and contains all the information necessary to process the request.
- **Resource-based:**
  The API is organized around resources (e.g., products, orders, users), with each resource having its own URL.
- **Use of HTTP methods:**
  Standard HTTP methods (GET, POST, PUT, DELETE) are used to perform CRUD operations on resources.

5.5.3 Endpoints:

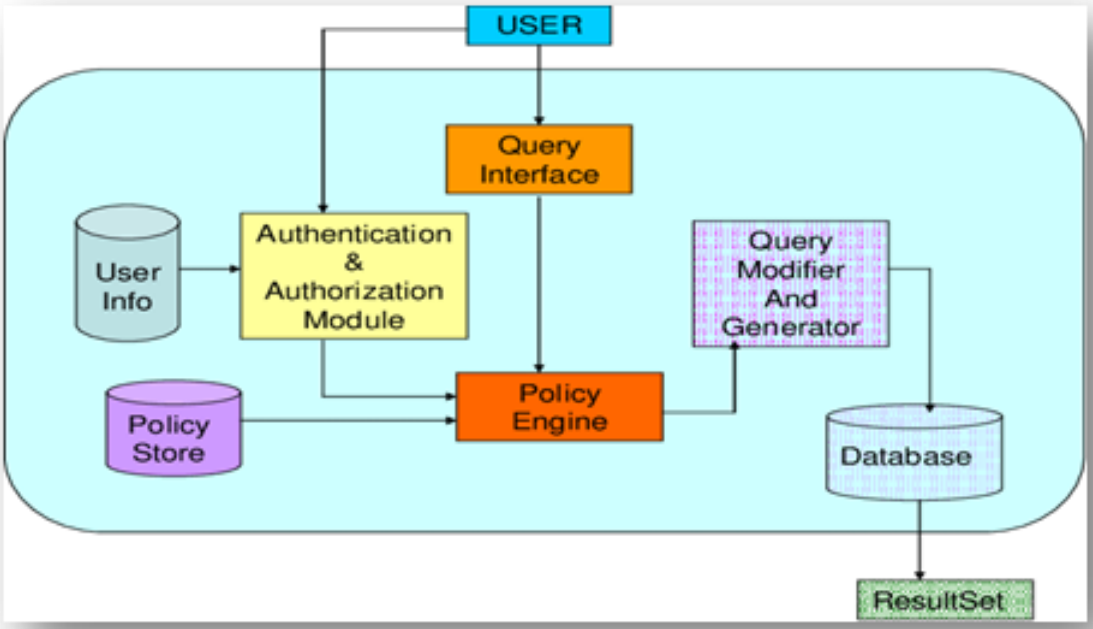
- **Key endpoints in the API might include:**

GET /api/products/: Retrieve a list of all products.

GET /api/products/{id}/: Retrieve details of a specific product.

5.6 AUTHENTICATION AND AUTHORIZATION:

To secure the API, authentication and authorization mechanisms are implemented. Django REST framework supports various authentication methods, such as:



- **Token-based authentication:**

  Users receive a token upon logging in, which must be included in the header of subsequent API requests.

- **Session-based authentication:**

  Maintains user sessions on the server side.

Authorization rules ensure that only authorized users can perform certain actions, such as adding products or approving orders.

5.7 DATABASE MANAGEMENT IN INVENTORY MANAGEMENT SYSTEM

The database management component of the University Inventory Management System is crucial for storing, organizing, and managing the data related to inventory items, user accounts, orders, and other essential aspects of the system. Effective database management ensures data integrity, security, and efficient access, which are vital for the smooth functioning of the system.

5.7.1 Database Selection:

For this project, a relational database such as PostgreSQL or MySQL is typically chosen due to its robustness, scalability, and support for complex queries and transactions. Django's ORM (Object-Relational Mapping) provides an abstraction layer that allows for seamless interaction with the database, making it easier to perform CRUD (Create, Read, Update, Delete) operations.

5.7.2 Data Modeling:

Data modeling involves defining the structure of the database in terms of tables, fields, and relationships. In the University Inventory Management System, this includes tables for users, products, orders, categories, and roles. Each table has specific fields that store relevant data, and relationships are established to link related tables, such as linking orders to users and products.

5.7.3 Schema Design:

The schema design defines how data is logically organized in the database. It includes defining primary keys, foreign keys, indexes, and constraints to ensure data integrity and optimize query performance. For example, the product table might include fields like product_id (primary key), name, description, quantity, price, and category_id (foreign key).

5.7.4 Data Integrity and Validation:

Data integrity is maintained through constraints and validation rules. Constraints such as unique, not null, and foreign key constraints ensure that the data adheres to the defined rules. Validation rules are enforced at both the application level (in Django models) and the database level to prevent invalid data entry.

5.7.5 Data Security:

Security measures are implemented to protect the data from unauthorized access and breaches. This includes using strong authentication and authorization mechanisms, encrypting sensitive data, and implementing regular backups. Django provides built-in security features to help mitigate common security threats.

5.7.6 Database Operations:

CRUD operations are essential for managing the data within the system. Django's ORM simplifies these operations, allowing developers to interact with the database using Python code instead of writing raw SQL queries. This abstraction improves productivity and reduces the likelihood of errors.

5.7.7 Performance Optimization:

To ensure efficient data retrieval and manipulation, performance optimization techniques are employed. This includes indexing frequently queried fields, optimizing database queries, and using caching mechanisms to reduce the load on the database.

5.7.8 Migration Management:

Django's migration system allows for easy management of database schema changes over time. Migrations are used to apply changes to the database schema in a controlled manner, ensuring that the database structure remains consistent with the application's data models.

5.8 SCREENS

5.8.1 Product Views.py

The ProductViewSet class in Django REST Framework provides a comprehensive API for managing Product instances, handling CRUD operations.

```python
from rest_framework.response import Response
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from rest_framework import status
from .models import Product
from .serializer import ProductSerializer
from .permissions import ProductPermission
from django.shortcuts import get_object_or_404


class ProductViewSet(viewsets.ModelViewSet):
    queryset = Product.objects.all()
    serializer_class = ProductSerializer
    permission_classes = [IsAuthenticated, ProductPermission]

    def create(self, request, *args, **kwargs):
        product_data = request.data.copy()
        product_data["employee"] = request.user.id
        serializer = self.get_serializer(data=product_data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)

    def retrieve(self, request, *args, **kwargs):
        product = get_object_or_404(Product, id=kwargs['pk'])
        product_serialized = ProductSerializer(instance=product)
        return Response(product_serialized.data, status=status.HTTP_200_OK)

    def list(self, request):
        queryset = Product.objects.all()
        Product_serialized = ProductSerializer(instance=queryset, many=True)
        return Response(Product_serialized.data, status=status.HTTP_200_OK)

    def partial_update(self, request, *args, **kwargs):
        instance = self.get_object()
        serializer = ProductSerializer(instance, data=request.data,
                                       partial=True)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data, status=status.HTTP_200_OK)

    def destroy(self, request, *args, **kwargs):
        instance = self.get_object()
        instance.delete()
        return Response({'Message': 'Deleted Sucessfuly'},
                        status=status.HTTP_200_OK)
```

5.9 ORDER VIEWS.PY

The AdminShipmentViewSet and UserShipmentViewSet classes provide APIs for managing Orders with different permission levels for admin and regular users.

```python
from rest_framework.response import Response
from .serializer import ShipmentsSerializer
from .models import Shipments
from rest_framework import viewsets, status
from rest_framework.permissions import IsAuthenticated
from .premissions import ShipmentsAdminPermission, ShipmentsUserPermission


class AdminShipmentViewSet(viewsets.ModelViewSet):
    queryset = Shipments.objects.all()
    serializer_class = ShipmentsSerializer
    permission_classes = [IsAuthenticated, ShipmentsAdminPermission]

    def list(self, request):
        queryset = self.get_queryset()
        shipment_serialized = ShipmentsSerializer(instance=queryset, many=True)
        return Response(shipment_serialized.data, status=status.HTTP_200_OK)

    def retrieve(self, request, *args, **kwargs):
        instance = self.get_object()
        shipment_serialized = ShipmentsSerializer(instance=instance)
        return Response(shipment_serialized.data, status=status.HTTP_200_OK)

    def partial_update(self, request, *args, **kwargs):
        instance = self.get_object()
        serializer = ShipmentsSerializer(instance,
                                         data=request.data, partial=True)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data, status=status.HTTP_200_OK)

    def destroy(self, request, *args, **kwargs):
        instance = self.get_object()
        instance.delete()
        return Response({'Message': 'Deleted Sucessfuly'},
                        status=status.HTTP_200_OK)


class UserShipmentViewset(viewsets.ModelViewSet):
    queryset = Shipments.objects.all()
    serializer_class = ShipmentsSerializer
    permission_classes = [IsAuthenticated, ShipmentsUserPermission]

    def list(self, request):
        queryset = self.get_queryset().filter(employee=self.request.user)
        shipment_serialized = ShipmentsSerializer(instance=queryset, many=True)
        return Response(shipment_serialized.data, status=status.HTTP_200_OK)

    def retrieve(self, request, *args, **kwargs):
        instance = self.get_object()
        if not instance.employee == request.user:
            return Response(status=status.HTTP_404_NOT_FOUND)
        shipment_serialized = ShipmentsSerializer(instance=instance)
        return Response(shipment_serialized.data, status=status.HTTP_200_OK)

    def create(self, request, *args, **kwargs):
        shipment_data = request.data.copy()
        shipment_data["employee"] = request.user.id
        serializer = ShipmentsSerializer(data=shipment_data)
        serializer.is_valid(raise_exception=True)
        serializer.save()
        return Response(serializer.data, status=status.HTTP_201_CREATED)

```

CHAPTER 7: References

[1]
"Front-End Development," *GeeksforGeeks*, Nov. 29, 2023. https://www.geeksforgeeks.org/front-end-development/

[2]
"Backend Development Complete Guide," *GeeksforGeeks*, Nov. 29, 2023. https://www.geeksforgeeks.org/backend-development/

[3]
"Django Tutorial," *www.w3schools.com*. https://www.w3schools.com/django/

[4]
JD, "Why Differentiate User Requirements vs. System Requirements?," *Shaping Software*, Jun. 09, 2008. https://shapingsoftware.com/user-requirements-vs-system-requirements/

[5]
Atlassian, "Gitflow Workflow | Atlassian Git Tutorial," *Atlassian*, 2019. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow