



SRI KRISHNA COLLEGE OF TECHNOLOGY

(An Autonomous Institution)

Approved by AICTE | Affiliated to Anna University Chennai|
Accredited by NBA - AICTE| Accredited by NAAC with 'A' Grade
KOVAIPUDUR, COIMBATORE 641042



INVENTORY MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

MOHAMED SAHEED ALI M	-	727822TUCS114
RITHESH NARAYANNAN P	-	727822TUCS154
NITIN J S	-	727822TUCS131
MOHAN KUMAR B	-	727822TUCS116

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

MARCH 2024

BONAFIDE CERTIFICATE

Certified that this project report INVENTORY MANAGEMENT SYSTEM is the bonafide work of MOHAMED SAHEED ALI M 727822TUCS114, RITHESH NARAYANNAN P 727822TUCS154, NITIN J S 727822TUCS131, MOHAN KUMAR B 727822TUCS116 who carried out the project work under my supervision.

SIGNATURE

Mr. S. SAM PETER

SUPERVISOR

Assistant Professor,

Department of Computer Science
and Engineering,

Sri Krishna College of

Technology, Coimbatore-641042

SIGNATURE

Dr. M.UDHAYAMOORTHY

HEAD OF THE DEPARTMENT

Associate Professor,

Department of Computer Science and
Engineering,

Sri Krishna College of

Technology, Coimbatore-641042

Certified that the candidate was examined by me in the Project Work Viva Voce

examination held on _____ at Sri Krishna College of Technology,

Coimbatore-641042.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First and foremost we thank the Almighty for being our light and for showering his gracious blessings throughout the course of this project.

We express our gratitude to our beloved Principal, Dr. M.G. Sumithra, for providing all facilities.

We are grateful to our beloved Head, Computing Sciences Dr.T.

Senthilnathan, for her tireless and relentless support.

With the grateful heart, our sincere thanks to our Head of the Department Dr. M.Udhayamoorthi, Department of Computer Science and Engineering for the motivation and all support to complete the project work.

We thank **Mrs. Divya P**, Assistant Professor, Department of Computer Science and Engineering, for his motivation and support.

We are thankful to all the Teaching and Non-Teaching Staff of Department of Computer Science and Engineering and to all those who have directly and indirectly extended their help to us in completing this project work successfully.

We extend our sincere thanks to our family members and our beloved friends, who had been strongly supporting us in all our endeavour

ABSTRACT

The Inventory Management System developed for a retail environment harnesses the latest technologies, including React, Spring Boot, and MongoDB, to deliver a sophisticated platform for comprehensive inventory oversight. Featuring a robust authentication system

and role-based access controls, the system ensures that employees can efficiently manage stock levels, process orders, and access real-time inventory updates securely. Administrators benefit from a feature-rich dashboard that facilitates detailed inventory tracking, stock adjustments, and advanced reporting capabilities to analyze product trends, inventory turnover, and supply chain performance. The system's intuitive user interface simplifies inventory management tasks such as adding, deleting, and modifying stock items. A dynamic calendar view provides a comprehensive overview of stock levels and upcoming supply needs, enabling proactive inventory management and reducing the risk of stockouts or overstock situations. Integrated email notifications keep employees informed about critical stock alerts, order confirmations, and updates, while real-time push notifications deliver immediate updates directly within the web interface, enhancing responsiveness and operational efficiency. Leveraging React's interactive frontend, Spring Boot's powerful RESTful APIs, and MongoDB's flexible and scalable database architecture, the system automates routine inventory tasks, streamlines processes, and enhances transparency across the supply chain. Its scalability ensures it can accommodate growing inventory demands and evolving business needs, while advanced reporting tools offer actionable insights for strategic decision-making and resource management. In summary, the Inventory Management System is a cutting-edge solution that meets the diverse needs of retail inventory management, delivering a streamlined, efficient, and user-centric experience that supports both day-to-day operations and longterm strategic goals.

CHAPTER.NO	TITLE	PAGE NO
1	INTRODUCTION 1	
	1.1 PROJECT OVERVIEW	
	1.2 OBJECTIVE	

2	SYSTEM SPECIFICATION	2
3	PROPOSED SYSTEM	4
	3.1PROPOSED SYSTEM	
	3.2ADVANTAGES	
4	METHODOLOGIES	6
5	IMPLEMENTATION AND RESULT	12
6	BACKEND SYSTEM SPECIFICATION	41
7	SYSTEM ARCHITECTURE	45
8	IMPLEMENTATION AND FUNCTIONALITY	48
9	CONCLUSION AND FUTURE SCOPE	61

LIST OF FIGURES

Figure No	TITLE	Page No
4.1	Process-Flow Diagram	6
4.2	Use Case Diagram	8
4.3	Class Diagram	9
4.4	ER Diagram	10
5.1	Login	11
5.2	User Register	12
5.3	Dashboard page	12
5.4	InventoryTracking page	13

	5.5	ProductionInfo Management	13
page	5.6	Order Management page	14
	5.7	Reorder Management page	14
	5.8	Reporting and Analytics	15
page	5.9	User Access page	15
	6.1	Backend System	41
		Specification	
	7.1	System Architecture	45
Functionality	8.1	Implementation And	48
	9.1	Conclusion	62

LIST OF ABBREVIATIONS

Abbreviation

Acronym

HTML	HYPERTEXT MARKUP LANGUAGE
CSS	CASCADING STYLESHEET
JS	JAVASCRIPT
SDLC	SOFTWARE DEVELOPMENT LIFE CYCLE
PHP	HYPER TEXT PREPROCESSOR

CHAPTER 1

INTRODUCTION

This project aims to offer a seamless and efficient solution for businesses and retailers seeking to manage their inventory through an online platform. In this chapter, we will explore the problem statement, provide an overview, and outline the main objectives of the inventory management system.

1.1 PROBLEM STATEMENT

How can we develop an inventory management system that enables businesses to efficiently track stock levels, manage orders, and maintain accurate inventory records, while ensuring a user-friendly interface, robust data security, and real-time updates for optimal operational efficiency??

1.2 OVERVIEW

In inventory management, businesses often face issues like inefficient stock tracking, order management difficulties, and inaccurate records. To address these, we propose an inventory management system that provides real-time tracking, streamlined order processes, and reliable record-keeping, all with a user-friendly interface and strong data security, ensuring efficient and secure management experience.

1.3 OBJECTIVE

The primary objective of this project is to develop an inventory management system that offers businesses an efficient and user-friendly platform for tracking stock levels, managing orders, and maintaining accurate records. By automating routine tasks and addressing traditional inefficiencies, the system aims to enhance operational efficiency and improve inefficiencies, the system aims to enhance operational efficiency and improve resource management.

CHAPTER 2

SYSTEM SPECIFICATION

In this chapter, we are going to see the software that we have used to build the website.

This chapter gives you a small description about the software used in the project.

VS CODE

Visual Studio Code is a source code editor developed by Microsoft for Windows, Linux, and macOS. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

VS Code is an excellent code editor for React projects. It is lightweight, customizable, and has a wide range of features that make it ideal for React development. It has built-in support for JavaScript, JSX, and TypeScript, and enables developers to quickly move between files and view detailed type definitions. It also has a built-in terminal for running tasks. Additionally, VS Code has an extensive library of tools, extensions that allow developers to quickly add features like code snippets, debugging and linting support to their projects.

2.1 LOCAL STORAGE

Local storage is a type of web storage for storing data on the client side of a web browser. It allows websites to store data on a user's computer, which can then be accessed by the website again when the user returns. Local storage is a more secure alternative to cookies because it allows websites to store data without having to send it back and forth with each request. Local storage is a key-value pair storage meaning it stores data in the form of a key and corresponding value. It is similar to a database table in that it stores data in columns and rows, except that local storage is used to store user information such as preferences and settings, or to store data that is not

website. Local storage is supported by all modern web browsers, including Chrome, Firefox, Safari, and Edge. It is accessible through the browser's JavaScript API. Local storage is a powerful tool for websites to store data on the client side. It is secure, efficient, and can be used to store data that does not need to be shared with other websites.

Local Storage is a great way to improve the performance of a website by caching data. Local storage in web browsers allows website data to be stored locally on the user's computer. It is a way of persistently storing data on the client side, which is not sent to the server with each request. This allows users to store data such as preferences, login information, and form data without needing to send it to a server. It is typically stored in a browser's cookie file, but it can also be stored in other locations such as HTML5 Local Storage and Indexed DB. The data stored in local storage is persistent and can be accessed by the website even if the user closes the browser or navigates to another page. It is a great way for websites to store user-specific data, as it is secure, reliable, and fast. It is also a great way for developers to store data that does not need to be sent to the server with each request.

One of the key benefits of using local storage is its reliability. Unlike server-side storage, which can be affected by network outages or other server issues, local storage is stored locally on the user's machine, and so is not affected by these issues. Another advantage of local storage is its speed. Because the data is stored locally, it is accessed quickly, as there is no need to send requests to a server. This makes it ideal for storing data that needs to be accessed quickly, such as user preferences or session data. Local storage is also secure, as the data is stored on the user's machine and not on a server. This means that the data is not accessible by anyone other than the user, making it a good choice for storing sensitive information.

CHAPTER 3

PROPOSED SYSTEM

This chapter gives a small description about the proposed idea behind the development of our website

3.1 PROPOSED SYSTEM

This system provides numerous advantages from various perspectives. The Inventory Management Portal enables users to efficiently track and manage inventory with real-time updates, detailed product information, and streamlined order processing. Payment methods include online transactions and offline options, catering to user preferences and ensuring flexibility.

Once inventory data is updated, it is reviewed by the administration team responsible for managing inventory levels and stock movements. The administrative staff ensures accurate and timely updates, with specialized personnel handling data entry, verification, and reconciliation. Completed updates are promptly reflected in the system, ensuring that inventory data is always current and ready for operational use.

The system significantly reduces administrative workload and enhances operational efficiency. During peak periods or high inventory volumes, where manual processes may cause delays or errors, the portal effectively mitigates these challenges. Users can manage inventory directly online, bypassing potential issues such as limited office hours or manual processing delays, which can otherwise hinder operational flow.

Moreover, the system's advanced features, such as automated reorder alerts, detailed analytics, and real-time stock tracking, further streamline inventory management. These enhancements not only improve operational efficiency but also provide valuable insights for decision-making, contributing to more strategic inventory control.

3.2 ADVANTAGES

- **Real-Time Inventory Tracking:** Provides up-to-the-minute visibility into inventory levels, enabling businesses to make informed decisions on stock management and prevent issues such as stockouts or overstocking.

- Automated Reorder Alerts: Automatically triggers alerts when inventory falls below preset thresholds, ensuring timely restocking and maintaining optimal inventory levels without manual intervention.
- Enhanced Operational Efficiency: Streamlines inventory management tasks such as order processing, data entry, and stock tracking, reducing administrative workload and minimizing errors associated with manual processes.
- Integration with Barcode and RFID Systems: Facilitates accurate and efficient inventory updates through integration with barcode and RFID technologies, enhancing inventory verification and reducing manual data entry.
- Cost Reduction: Helps lower operational costs by optimizing inventory levels, reducing overstocking and stockouts, and improving inventory turnover, leading to better financial performance.
- Comprehensive Reporting and Analytics: Provides detailed reports and insights into inventory performance, turnover, and valuation, helping businesses analyze trends, forecast demand, and make data-driven decisions.

CHAPTER 4

METHODOLOGIES

This chapter gives a small description about how our system works.

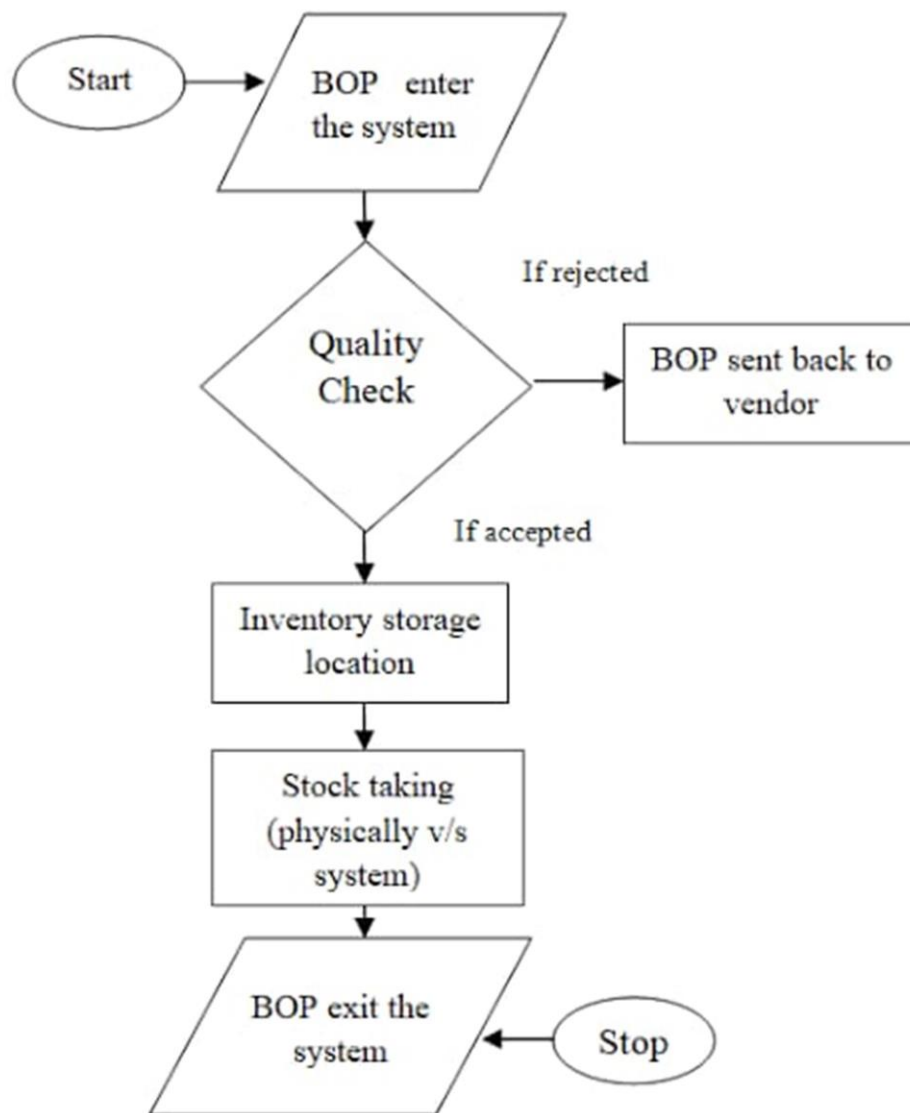


Fig 4.1.Process flow diagram

1. User Registration:

Users can register on the platform by providing essential details such as name, email address, and password.

Optional information like address and contact number may be included for communication and account recovery purposes.

2. Product Information Management:

Upon registration/login, users can manage product information by browsing through the catalog or using search functionality.

Each product listing includes details such as product name, description, SKU (Stock Keeping Unit), price, and supplier information.

3. Inventory Tracking:

Users can track inventory levels in real-time. The system provides visibility into the quantity and location of each item in stock.

Users can monitor stock levels, view current inventory status, and make informed decisions regarding replenishment or reordering.

4. Order Management:

Users can manage various types of orders including purchase orders, sales orders, and transfer orders.

The system allows users to create, view, and update orders while automatically updating stock levels based on order processing.

5. Stock Movement:

Records and tracks stock movements, including receiving, transferring, and fulfilling orders.

6. Barcode or RFID Integration:

Integrates with barcode/RFID systems for efficient updates, stock counts, and verification.

7. Reorder Management:

Sets reorder points and sends alerts when inventory levels fall below thresholds.

8. Reporting and Analytics:

Users can generate various reports and perform analytics on inventory performance and trends.

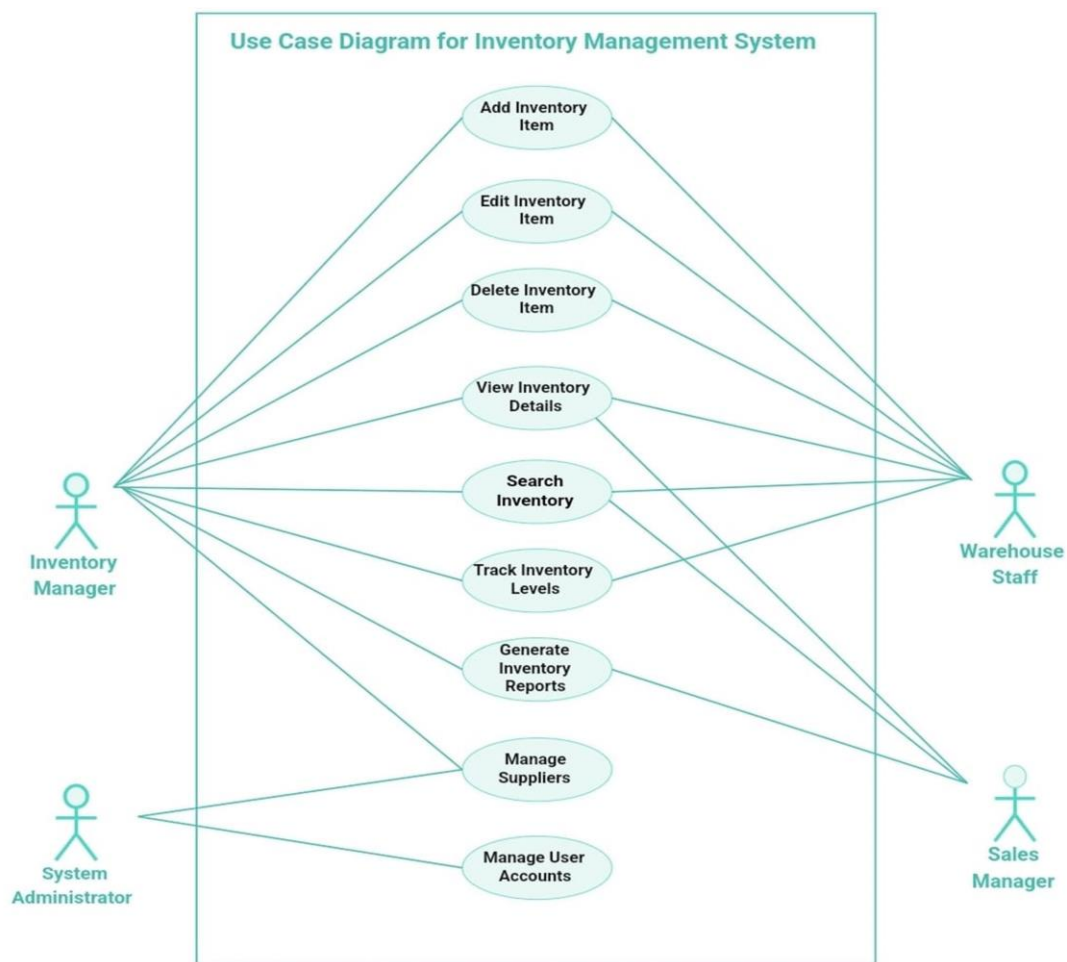
Reports may include stock levels, inventory turnover, stock valuation, and other key metrics to support data-driven decision-making.

9. User Access Control and Security:

Implements role-based access controls and data encryption to protect sensitive information.

Use Case Diagram:

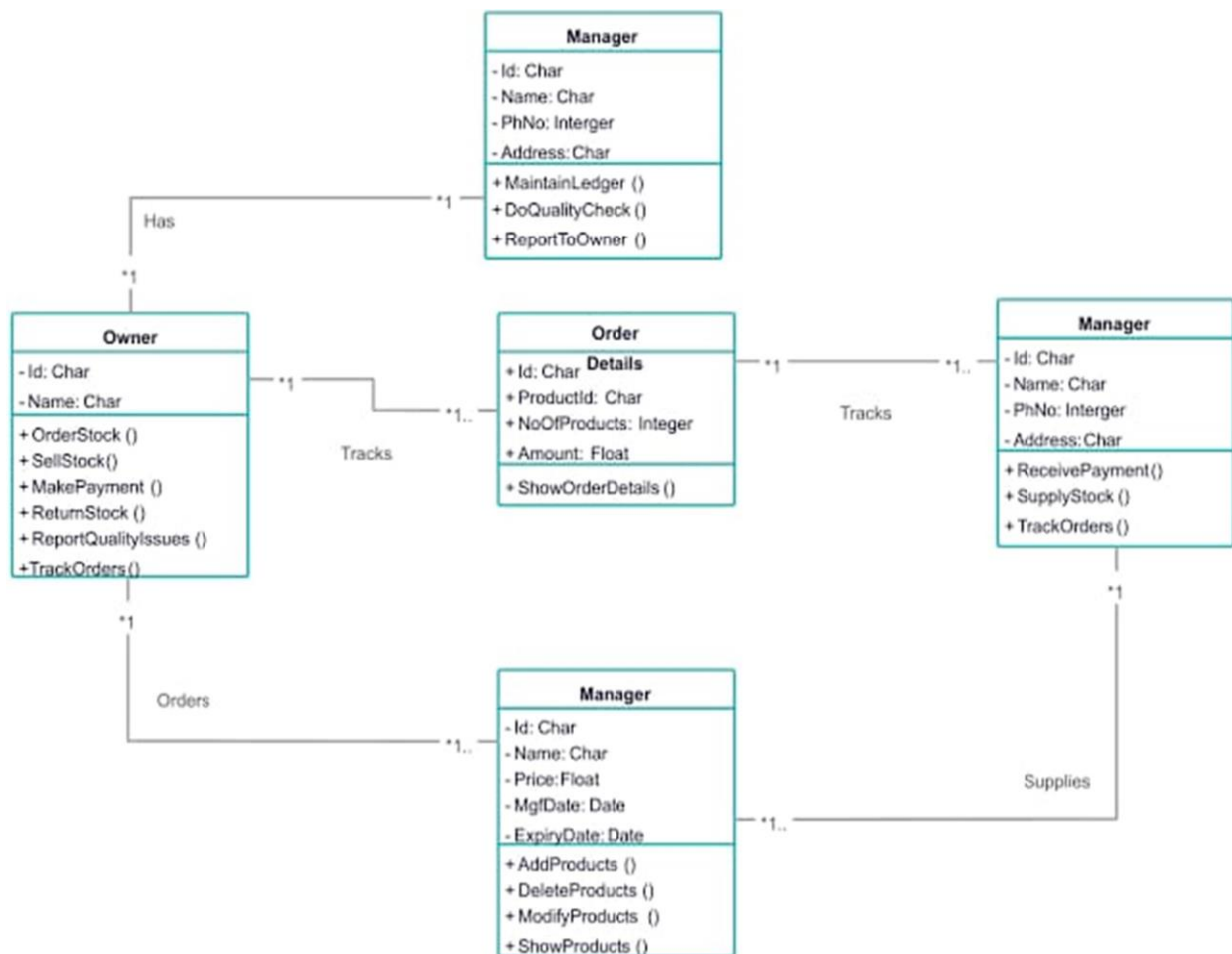
The Inventory Management Portal enables users to efficiently manage inventory by tracking stock levels, processing orders, and maintaining product information. Users can browse and update product details, monitor real-time inventory levels, and handle stock movements. Admins oversee inventory management, manage reorder points, and generate reports, ensuring a streamlined and effective inventory management process for both users and administrators.



4.2 Use Case Diagram

Class Diagram:

The Inventory Management Portal comprises classes such as User, Product, Order, StockMovement, and Admin. Users can manage inventory by updating product details, tracking stock levels, and processing orders. Products are managed by admins, who can add, edit, or remove product entries. Orders link users to product inventory and stock movements, with methods like UpdateStock() and GenerateReport(). Admins oversee the system, managing inventory data, setting reorder points, and generating analytical reports. This diagram illustrates the system's core functionalities and relationships.

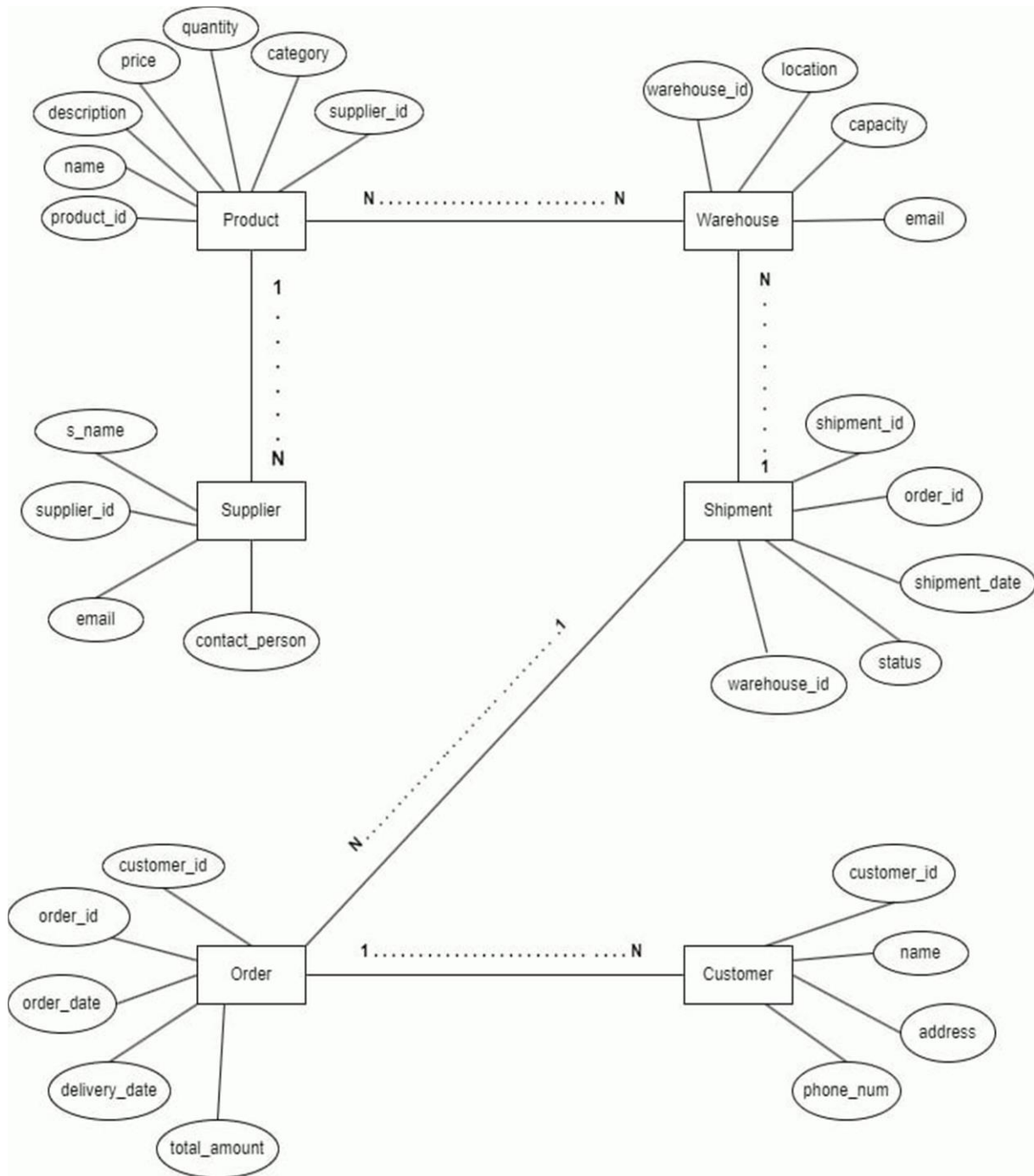


4.3 Class Diagram

ER Diagram:

The ER Diagram for Inventory Management visualizes entities like Products, Orders, and Stock Movements, showing their relationships and attributes. It highlights how users can update product details and track inventory, while admins manage stock levels, reorder points, and generate reports.

This diagram provides a clear framework for efficient inventory management and data flow.



4.4 ER Diagram

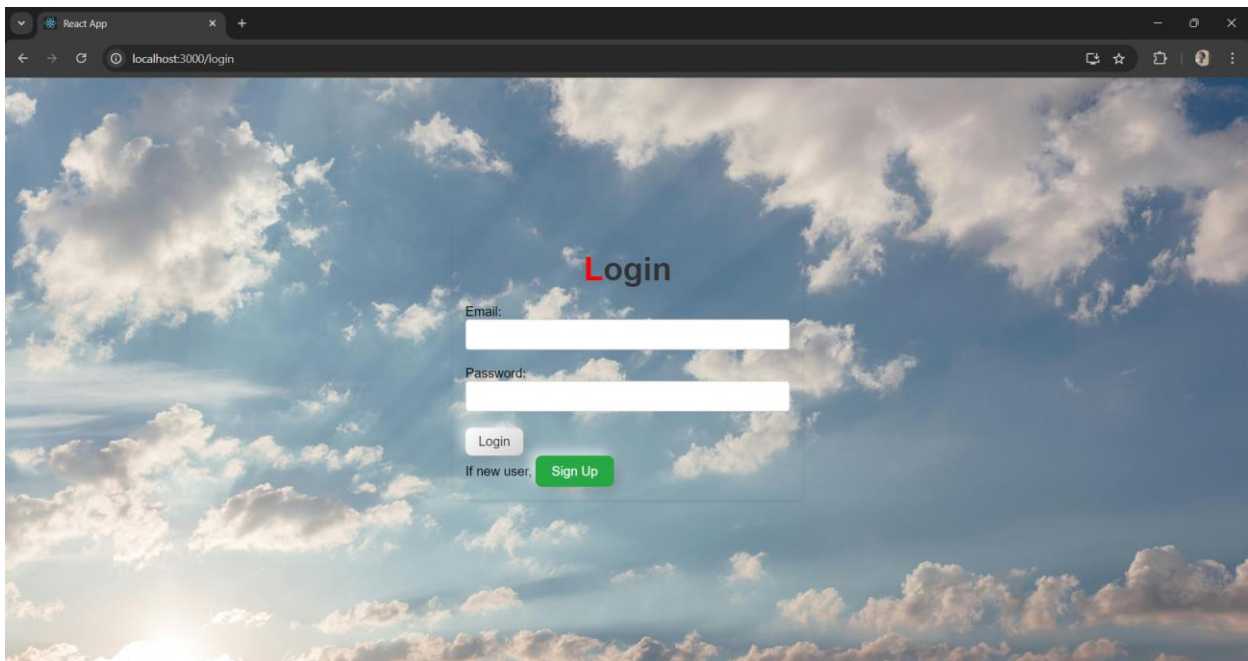
CHAPTER 5

IMPLEMENTATION AND RESULT

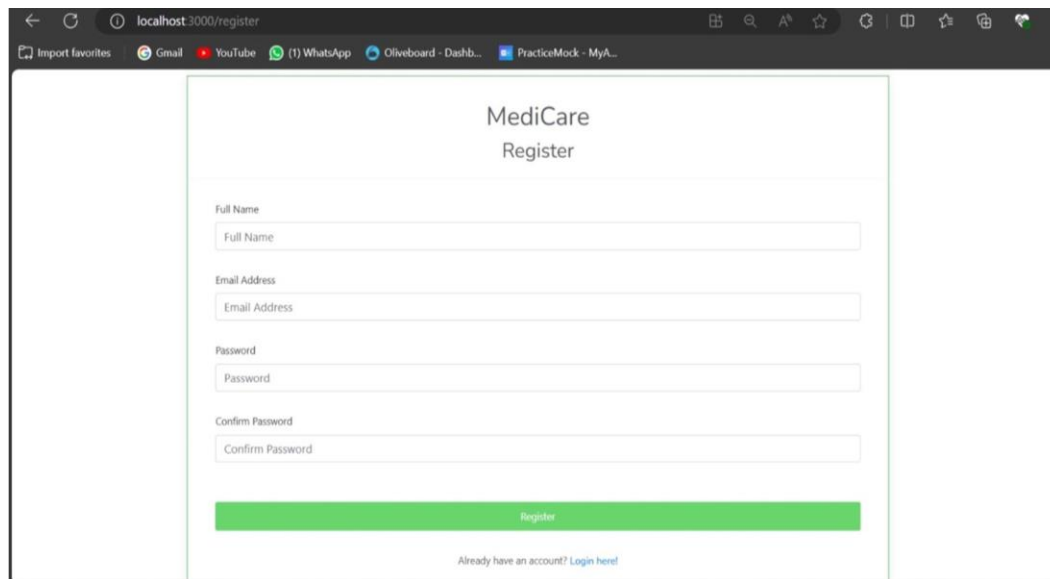
This chapter gives a description about the output that we produced by developing the website of our idea.

5.1 LOGIN

When User enters our website he will be asked about his login details like email id and password. The login details will be verified with the details given while the user creates an account.

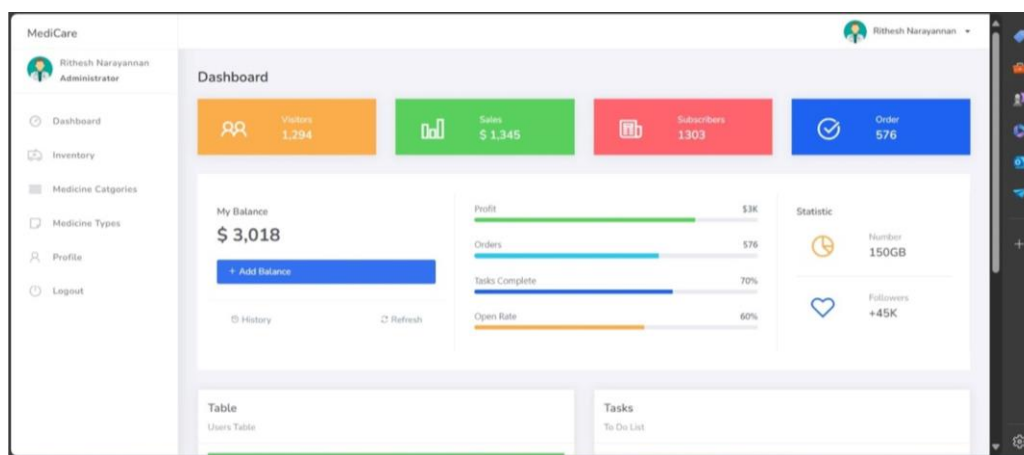


5.2 USER REGISTER

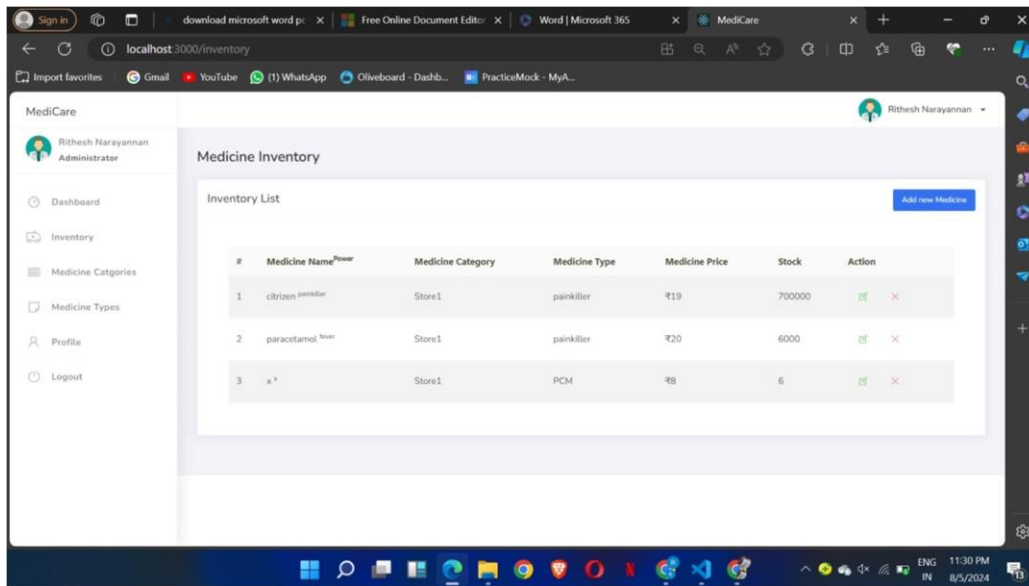


A screenshot of a web browser showing a registration form for 'MediCare'. The browser's address bar shows 'localhost:3000/register'. The form is titled 'MediCare Register' and contains four input fields: 'Full Name', 'Email Address', 'Password', and 'Confirm Password'. Below these fields is a green 'Register' button. At the bottom of the form, there is a link that says 'Already have an account? Login here!'.

5.3 DASHBOARD PAGE



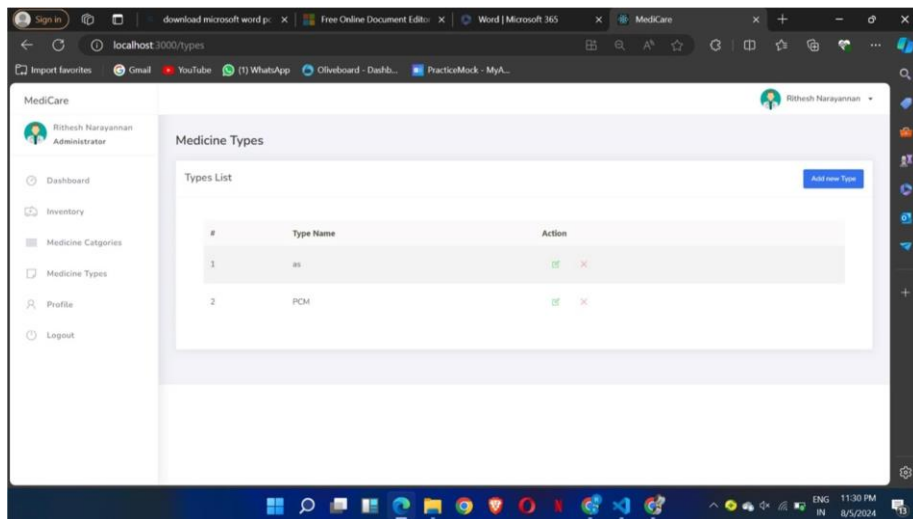
5.4 INVENTORY TRACKING PAGE



The screenshot shows the MediCare application interface for the Inventory Tracking page. The browser address bar displays `localhost:3000/inventory`. The left sidebar contains navigation links: Dashboard, Inventory, Medicine Categories, Medicine Types, Profile, and Logout. The main content area is titled "Medicine Inventory" and features an "Inventory List" table. A blue button labeled "Add new Medicine" is located in the top right corner of the table area. The table lists three items with columns for #, Medicine Name, Medicine Category, Medicine Type, Medicine Price, Stock, and Action.

#	Medicine Name	Medicine Category	Medicine Type	Medicine Price	Stock	Action
1	citizen painkiller	Store1	painkiller	₹19	700000	Edit Delete
2	paracetamol	Store1	painkiller	₹20	6000	Edit Delete
3	x	Store1	PCM	₹8	6	Edit Delete

5.5 PRODUCTINFO MANAGEMENT PAGE



The screenshot shows the MediCare application interface for the Product Info Management page. The browser address bar displays `localhost:3000/types`. The left sidebar contains navigation links: Dashboard, Inventory, Medicine Categories, Medicine Types, Profile, and Logout. The main content area is titled "Medicine Types" and features a "Types List" table. A blue button labeled "Add new Type" is located in the top right corner of the table area. The table lists two items with columns for #, Type Name, and Action.

#	Type Name	Action
1	as	Edit Delete
2	PCM	Edit Delete

5.6 ORDER MANAGEMENT PAGE

Add New Item

Product Name

Product SKU



Product Price

Description

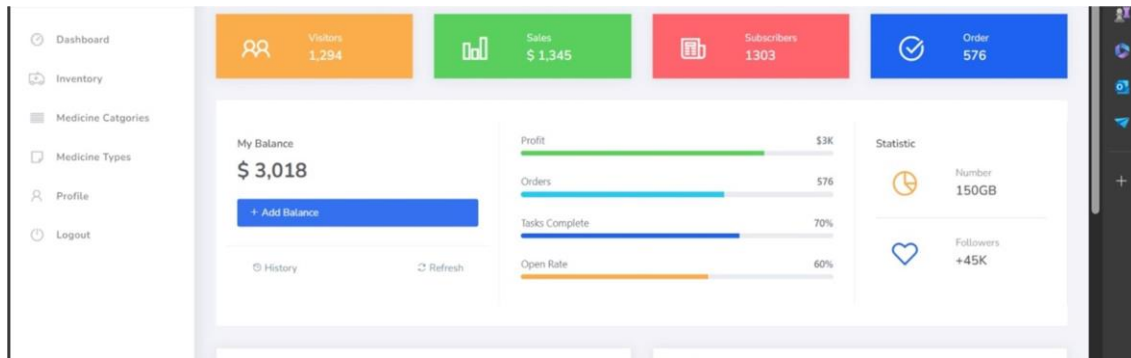
V

5.7 REORDER MANAGEMENT PAGE

The screenshot displays the MediCare web application interface. The browser address bar shows the URL `localhost:3000/categories`. The application has a sidebar menu on the left with the following items: Dashboard, Inventory, Medicine Categories (selected), Medicine Types, Profile, and Logout. The main content area is titled "Medicine Categories" and features a "Categories List" table. The table has three columns: #, Category Name, and Action. It contains one row with the ID "1", the name "Store1", and action icons for edit and delete. An "Add new Category" button is located in the top right corner of the table area. The user profile "Rithesh Narayanan" is visible in the top right corner of the application. The Windows taskbar at the bottom shows the time as 11:30 PM on 8/5/2024.

#	Category Name	Action
1	Store1	 

5.8 REPORTING AND ANALYTICS PAGE



5.9 USER ACCESS PAGE

The screenshot shows the User Access page of the MedCare system. The page includes a sidebar with navigation links: Dashboard, Inventory, Medicine Categories, Medicine Types, Profile, and Logout. The main content area is divided into two sections:

- Table:** A table titled 'Users Table' with columns for ID, First Name, Last Name, and Handle. It contains three rows of user data.
- Tasks:** A section titled 'To Do List' with a table of tasks. Each task has a checkbox, a description, and an 'Action' column with edit and delete icons.

The tasks listed are:

- Planning new project structure
- Update Profile
- Add new Post
- Finalize the design proposal

The page also shows a timestamp 'Updated 3 minutes ago' at the bottom of the tasks section.

CODING

Login:

```

import ImgBack from '../assets/img-back.jpeg'; function
  Login() {
    const [email, setEmail] = useState("");    const
    [password, setPassword] = useState("");

    const navigate = useNavigate();

    const handleSubmit = (e) => {
      e.preventDefault();
      localStorage.setItem('isAuthenticated', 'true');    navigate('/dashboard');
    } else {
      alert('Invalid credentials');
    }
  };
return (
  <div className="login-register-container">
    <div className="image-container">
      <img src={ImgBack} alt="Inventory Management" />
    </div>
    <div className="form-card-container">
      <div className="card">

```

```

    <h2>Login</h2>
    <form onSubmit={handleSubmit}>
      <div className="form-group">
        <label htmlFor="email">Email:</label>
        <input
type="email" id="email"
value={email}
onChange={(e) => setEmail(e.target.value)}
required placeholder="Email"
/>
      </div>
      <div className="form-group">
        <label htmlFor="password">Password:</label>
        <input type="password"
id="password"
value={password}
onChange={(e) => setPassword(e.target.value)}
required placeholder="Password"
/>
      </div>
      <button type="submit">Login</button>
    </form>
    <div className="links">
      <span>Don't have an account? <a
    </div>
  </div>
</div>
);

```



```
} export default
```

```
Login;
```

Register page:

```
function Register() {
```

```
    const [email, setEmail] = useState("");    const [password,
setPassword] = useState("");    const [confirmPassword,
setConfirmPassword] = useState("");
```

```
    const navigate = useNavigate();
```

```
    const handleSubmit = (e) => {
        e.preventDefault();
        // Handle registration logic here
        console.log('Register details:', { email, password, confirmPassword });
        navigate('/dashboard');
    };
};
```

```
    return (
        <div className="login-register-container">
            <div className="image-container">
                <img src={ImgBack} alt="Inventory Management" />
            </div>
            <div className="form-card-container">
                <div className="card">
                    <h2>Register</h2>
                    <form onSubmit={handleSubmit}>
                        <div className="form-group">
                            <label htmlFor="email">Email:</label>
                            <input
type="email"
```

```

id="email"
value={email}
onChange={(e) =>
  setEmail(e.target.value)}
required

      placeholder="Email"
    />
  </div>
  <div className="form-group">
    <label htmlFor="password">Password:</label>
    <input
      type="password"
      id="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      required

      placeholder="Password"
    />
  </div>
  <div className="form-group">
    <label htmlFor="confirmPassword">Confirm Password:</label>
    <input
      type="password"
      id="confirmPassword"
      value={confirmPassword}
      onChange={(e) => setConfirmPassword(e.target.value)}
      required

      placeholder="Confirm Password"
    />
  </div>
  <button type="submit">Register</button>

```

```

    </form>
    <div className="links">
      <span>Already have an account? <a href="/login">Login</a></span>
    </div>
  </div>
</div>
</div>
);
}

```

export default Register;

Dashboard page:

```

function Dashboard() {
  return (
    <div className="dashboard-container">
      <h2>Dashboard</h2>
      <div className="dashboard-sections">
        <div className="dashboard-card purchases">
          <FontAwesomeIcon icon={faShoppingBag} className="card-icon" />
          <div className="card-content">
            <h3>PURCHASES</h3>
            <p>1671405388.73</p>
            <span>New Purchase</span>
          </div>
        </div>
      </div>
      <div className="dashboard-card inventory">

```

```

<FontAwesomeIcon icon={faBoxOpen} className="card-icon" />

<div className="card-content">
  <h3>INVENTORY</h3>

  <p>894</p>

  <span>New Inventory</span>

</div>

</div>

<div className="dashboard-card sales">

  <FontAwesomeIcon icon={faChartLine} className="card-icon" />

  <div className="card-content">

    <h3>SALES</h3>

    <p>2889695579.63</p>

    <span>New Sale</span>

  </div>

</div>

<div className="dashboard-card profit">

  <FontAwesomeIcon icon={faDollarSign} className="card-icon" />

  <div className="card-content">

    <h3>PROFIT</h3>

    <p>2813631873.43</p>

    <span>New Sale</span>

  </div>

</div>

<div className="dashboard-card sale-orders">

```

```
<FontAwesomeIcon icon={faClipboardList} className="card-icon" />

<div className="card-content">
  <h3>SALE ORDERS</h3>

  <p>1933</p>

  <span>New Sale</span>
</div>

</div>

<div className="dashboard-card purchase-orders">

  <FontAwesomeIcon icon={faShoppingCart} className="card-icon" />

  <div className="card-content">

    <h3>PURCHASE ORDERS</h3>

    <p>362</p>

    <span>New Purchase</span>
  </div>

</div>

<div className="dashboard-card suppliers">

  <FontAwesomeIcon icon={faTruck} className="card-icon" />

  <div className="card-content">

    <h3>SUPPLIERS</h3>

    <p>212</p>

    <span>New Suppliers</span>
  </div>

</div>

<div className="dashboard-card customers">
```

```

    <FontAwesomeIcon icon={faUsers} className="card-icon" />

    <div className="card-content">
      <h3>CUSTOMERS</h3>

      <p>856</p>

      <span>New Customer</span>

    </div>

  </div>

</div>

</div>

</div>

);

}      export default

```

Dashboard;

Layout Page:

```

function Layout({ children }) {
  const [isSidebarHidden, setSidebarHidden] = useState(false);

  const toggleSidebar = () => {
    setSidebarHidden(!isSidebarHidden);
  };

  return (
    <div className={`layout ${isSidebarHidden ? 'sidebar-hidden' : ''}`>
      <TopNavbar
        onToggleSidebar={toggleSidebar}
        username="John Doe"
        profileImageUrl="path/to/profile-image.jpg"

```

```

    />
    <div className="content">
      <Navbar />
      <div className="main-content">
        {children}
      </div>
    </div>
  </div>
);
}

```

export default Layout; Navbar

Page:

```

function Navbar({ isSidebarCollapsed }) {  return
(
  <nav className={`navbar ${isSidebarCollapsed ? 'collapsed' : ''}`>
    <ul className="navbar-list">
      <li>
        <NavLink to="/dashboard" className={({ isActive }) => isActive ?
'active' : ''}>
          <FaTachometerAlt />
          <span>Dashboard</span>
        </NavLink>
      </li>
      <li>
        <NavLink to="/inventory-tracking" className={({ isActive }) => isActive
? 'active' : ''}>
          <FaCubes />
          <span>Inventory Tracking</span>

```

```

        </NavLink>
    </li>
    <li>
        <NavLink to="/product-info-management" className={{ { isActive }}
=> isActive ? 'active' : ""}>
            <FaInfoCircle />
            <span>Product Info</span>
        </NavLink>
    </li>
    <li>
        <NavLink to="/order-management" className={{ { isActive }} => isActive
? 'active' : ""}>
            <FaClipboardList />
            <span>Order Management</span>
        </NavLink>
    </li>
    <li>
        <NavLink to="/stock-movement" className={{ { isActive }} => isActive ?
'active' : ""}>
            <FaExchangeAlt />
            <span>Stock Movement</span>
        </NavLink>
    </li>
    <li>
        <NavLink to="/reorder-management" className={{ { isActive }} =>
isActive ? 'active' : ""}>
            <FaRedo />
            <span>Reorder Management</span>
        </NavLink>
    </li>

```



```

      <li>
        <NavLink to="/reporting-analytics" className={{ { isActive }} => isActive
? 'active' : ""}>
          <FaChartLine />
          <span>Reporting & Analytics</span>
        </NavLink>
      </li>
      <li>
        <NavLink to="/user-access-control" className={{ { isActive }} => isActive
? 'active' : ""}>
          <FaUserShield />
          <span>User Access</span>
        </NavLink>
      </li>
    </ul>
  </nav>
);
}

```

export default Navbar;

TopNav Bar Page:

```

function TopNavbar({ username, profileImageUrl = profileImage, onToggleSidebar
}) {
  const [isDropdownVisible, setDropdownVisible] = useState(false);  const
navigate = useNavigate();

  const toggleDropdown = () => {
    setDropdownVisible(!isDropdownVisible);
  };
}

```

```

const handleLogout = () => {
  navigate('/login');
};

return (
  <div className="top-navbar">
    <div className="hamburger-icon" onClick={onToggleSidebar}>
      <FaBars />
    </div>
    <div className="profile-section">
      <span className="username">{username}</span>
      <img src={profileImageUrl} alt="Profile" className="profile-icon" />
      <FaCaretDown className="caret-icon" onClick={toggleDropdown} />
      {isDropdownVisible && (
        <div className="dropdown-menu">
          <Link to="/profile">Profile</Link>
          <a href="#logout" onClick={handleLogout}>Logout</a>
        </div>
      )}
    </div>
  </div>
);
}

export default TopNavbar;

```

InventoryTracking:

```

function InventoryTracking() {
  return (

```

```

<div className="inventory-tracking">
  <h3>Inventory Tracking</h3>
  <p>Real-time tracking of inventory levels and locations.</p>
  <div className="inventory-stats">
    <div className="stat">
      <h4>Total Items</h4>
      <p>1500</p>
    </div>
    <div className="stat">
      <h4>Out of Stock</h4>
      <p>25</p>
    </div>
    <div className="stat">
      <h4>Low Stock</h4>
      <p>75</p>
    </div>
  </div>
  <div className="inventory-list">
    <h4>Recent Inventory Updates</h4>
    <ul>
      <li>Item A - 50 units added</li>
      <li>Item B - 20 units removed</li>
      <li>Item C - 100 units transferred</li>
      { /* Add more inventory updates here */ }
    </ul>
  </div>
  <div className="inventory-actions">
    <button>Add New Item</button>
    <button>Update Stock</button>
    <button>Transfer Stock</button>
  </div>
</div>

```

```

        </div>
    </div>
);
}

```

```
export default InventoryTracking;
```

OrderManagement:

```

function OrderManagement() {
    return (
        <div className="order-management">
            <h3>Order Management</h3>
            <p>Manage purchase, sales, and transfer orders.</p>
            <div className="order-details">
                <div className="detail">
                    <label>Order ID:</label>
                    <input type="text" placeholder="Enter order ID" />
                </div>
                <div className="detail">
                    <label>Customer Name:</label>
                    <input type="text" placeholder="Enter customer name" />
                </div>
                <div className="detail">
                    <label>Order Date:</label>
                    <input type="date" />
                </div>
                <div className="detail">
                    <label>Status:</label>
                    <select>
                        <option value="pending">Pending</option>

```

```

        <option value="completed">Completed</option>
        <option value="canceled">Canceled</option>
    </select>
</div>
</div>
<div className="order-actions">
    <button>Add Order</button>
    <button>Update Order</button>
    <button>Delete Order</button>
</div>
</div>
);
}

```

```
export default OrderManagement;
```

ProductInfoManagement:

```

function ProductInfoManagement() {
    return (
        <div className="product-info-management">
            <h3>Product Information Management</h3>
            <p>Manage product details, SKUs, prices, and more.</p>
            <div className="product-details">
                <div className="detail">
                    <label>Product Name:</label>
                    <input type="text" placeholder="Enter product name" />
                </div>
                <div className="detail">
                    <label>SKU:</label>
                    <input type="text" placeholder="Enter SKU" />
                </div>
            </div>
        </div>
    );
}

```

```

    </div>
    <div className="detail">
      <label>Price:</label>
      <input type="number" placeholder="Enter price" />
    </div>
    <div className="detail">
      <label>Supplier:</label>
      <input type="text" placeholder="Enter supplier details" />
    </div>
    <div className="detail">
      <label>Description:</label>
      <textarea placeholder="Enter product description"></textarea>
    </div>
  </div>
  <div className="product-actions">
    <button>Add Product</button>
    <button>Update Product</button>
    <button>Delete Product</button>
  </div>

</div>
);
}

```

```
export default ProductInfoManagement;
```

Profile:

```

function Profile() {
  return (
    <div className="profile-container">

```

```

<h2>Profile</h2>
<form>
  <div className="form-group">
    <label htmlFor="username">Username</label>
    <input
type="text"          id="username"
name="username"
autocomplete="username"
required
    />
  </div>
  <div className="form-group">
    <label htmlFor="email">Email</label>
    <input
type="email"
id="email"
name="email"
autocomplete="email"
required
    />
  </div>
  <div className="form-group">
    <label htmlFor="current-password">Current Password</label>
    <input          type="password"
id="current-password"
name="current-password"
autocomplete="current-password"
required
    />
  </div>

```

```

    <div className="form-group">
      <label htmlFor="new-password">New Password</label>
      <input
type="password"          id="new-
password"                name="new-
password"                autocomplete="new-
password"
      />
    </div>
    <button type="submit">Update Profile</button>
  </form>
</div>
);
}

```

```
export default Profile;
```

ReorderManagement:

```

function ReorderManagement() {
  return (
    <div className="reorder-management">
      <h3>Reorder Management</h3>
      <p>Manage reorder points and alerts.</p>
      <div className="reorder-details">
        <div className="detail">
          <label>Product Name:</label>
          <input type="text" placeholder="Enter product name" />
        </div>
        <div className="detail">
          <label>Reorder Point:</label>

```



```

        <input type="number" placeholder="Enter reorder point" />
    </div>
    <div className="detail">
        <label>Current Stock:</label>
        <input type="number" placeholder="Enter current stock" />
    </div>
    <div className="detail">
        <label>Supplier:</label>
        <input type="text" placeholder="Enter supplier name" />
    </div>
    <div className="detail">
        <label>Email Alert:</label>
        <input type="email" placeholder="Enter email for alerts" />
    </div>
</div>

<div className="reorder-actions">
    <button>Add Reorder Point</button>
    <button>Update Reorder Point</button>
    <button>Remove Reorder Point</button>
</div>

</div>

);
}

```

```
export default ReorderManagement;
```

ReportingAndAnalytics:

```
function ReportingAndAnalytics() {
    const inventoryData = {
```

```

      labels: ['Product A', 'Product B', 'Product C', 'Product D', 'Product E'],
      datasets: [{
        label: 'Stock Levels',
        data: [100, 200, 150, 80, 120],
        backgroundColor: 'rgba(75, 192, 192, 0.2)',
        borderColor: 'rgba(75, 192, 192, 1)',
        borderWidth: 1
      }]
    };

    const stockPieData = {
      labels: ['In Stock', 'Out of Stock', 'Low Stock'],
      datasets: [{
        label: 'Stock Status',
        data: [60, 10, 30],
        backgroundColor: ['rgba(75, 192, 192, 0.2)', 'rgba(255, 99, 132, 0.2)', 'rgba(255, 206, 86, 0.2)'],
        borderColor: ['rgba(75, 192, 192, 1)', 'rgba(255, 99, 132, 1)', 'rgba(255, 206, 86, 1)'],
        borderWidth: 1
      }]
    };

    return (
      <div className="reporting-analytics">
        <h3>Reporting and Analytics</h3>
        <p>Generate and view inventory performance reports.</p>
        <div className="charts-container">
          <div className="left-charts">
            <div className="chart">
              <h4>Stock Levels</h4>
              <Bar data={inventoryData} />
            </div>
          </div>
        </div>
      )

```

```

        <div className="chart pie-chart">
            <h4>Stock Status</h4>
            <Pie data={stockPieData} />
        </div>
    </div>
</div>
);
}

```

```
export default ReportingAndAnalytics;
```

StockMovement:

```

function StockMovement() {
    return (
        <div className="stock-movement">
            <h3>Stock Movement</h3>
            <p>Track stock movements and manage warehouses.</p>
            <div className="movement-details">
                <div className="detail">
                    <label>Movement ID:</label>
                    <input type="text" placeholder="Enter movement ID" />
                </div>
                <div className="detail">
                    <label>Item Name:</label>
                    <input type="text" placeholder="Enter item name" />
                </div>
                <div className="detail">
                    <label>Quantity:</label>
                    <input type="number" placeholder="Enter quantity" />
                </div>
            </div>
        </div>
    );
}

```

```

    <div className="detail">
      <label>From Location:</label>
      <input type="text" placeholder="Enter source location" />
    </div>
    <div className="detail">
      <label>To Location:</label>
      <input type="text" placeholder="Enter destination location" />
    </div>
    <div className="detail">
      <label>Date:</label>
      <input type="date" />
    </div>
  </div>
  <div className="movement-actions">
    <button>Add Movement</button>
    <button>Update Movement</button>
    <button>Delete Movement</button>
  </div>

</div>
);
}

export default StockMovement;

```

UserAccessControl:

```

function UserAccessControl() {  const
[users, setUsers] = useState([]);

  const [newUser, setNewUser] = useState({ name: "", email: "", role: "" });

  const handleInputChange = (e) => {    const {
name, value } = e.target;    setNewUser({
...newUser, [name]: value });
  };

  const handleAddUser = () => {
setUsers([...users, newUser]);    setNewUser({
name: "", email: "", role: "" });
  };

  return (
    <div className="user-access-control">
      <h3>User Access Control</h3>
      <p>Manage user roles and access controls.</p>
      <div className="controls-container">
        <div className="control">
          <h4>Add New User</h4>
          <input
type="text"
name="name"
placeholder="Name"
value={newUser.name}
onChange={handleInputChange}
/>>
          <input
type="email"
name="email"

```

```

placeholder="Email"
value={newUser.email}
onChange={handleInputChange}
      />      <select
name="role"      value={newUser.role}
onChange={handleInputChange}
      >
        <option value="">Select Role</option>
        <option value="Admin">Admin</option>
        <option value="Editor">Editor</option>
        <option value="Viewer">Viewer</option>
      </select>
      <button onClick={handleAddUser}>Add User</button>
    </div>
    <div className="control">
      <h4>Current Users</h4>
      <table>
        <thead>
          <tr>
            <th>Name</th>
            <th>Email</th>
            <th>Role</th>
          </tr>
        </thead>
        <tbody>
          {users.map((user, index) => (
            <tr key={index}>
              <td>{user.name}</td>
              <td>{user.email}</td>
              <td>{user.role}</td>

```

```
        </tr>
      )}
    </tbody>
  </table>
</div>
</div>

);
}

export default UserAccessControl;
```