SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution)

(Approved by AICTE and Affiliated to Anna University, Chennai)

ACCREDITED BY NAAC WITH "A" GRADE

BONAFIDE CERTIFICATE

Certified that this project report titled "**INVENTORY MANAGEMENTSYSTEM**" is the Bonafede work of **"MOHAMMED IFRAN S(727721EUCS076), NIRANJANAGURUMALLESH M (727721EUCS092) NITHISH KUMAR P (727721EUCS095)"** who carried out the project work under my supervision.

SIGNATURE	SIGNATURE
Ms. S. BIRUNTHA	**Dr. K. SASI KALA RANI**
SUPERVISOR,	**HEAD OF THE DEPARTMENT**
ASSISTANT PROFESSOR	

Department of Computer Science and Engineering

Sri Krishna College of Engineering and Technology

Kuniyamuthur,

Coimbatore.

This Project report is submitted for Autonomous Project Viva-Voce examination held on ……...

INTERNAL EXAMINER	**EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our sincere thanks to the management and **Dr. J. JANET M.E., Ph.D.,** Principal, Sri Krishna College of Engineering and Technology, Coimbatore for providing us the facilities to carry out this mini project work.

We are thankful to **Dr. K. SASI KALA RANI M.E., Ph.D.,** Professor and Head, Department of Computer Science and Engineering, for her continuous evaluation and comments given during the course of the mini project work.

We express our deep sense of gratitude to our supervisor **Ms. BIRUNTHA, (Ph.D.)**, Assistant Professor, Department of Computer science and Engineering for her valuableadvice, guidance and support during the course of our mini project work.

We express our heartfelt sense of gratitude and thanks to our beloved parents, family and friends who have helped during the mini project course.

ABSTRACT

The Inventory Management System is a comprehensive solution designed to revolutionize inventory control processes. It aims to provide businesses with a powerful toolset for optimizing their stock management, reducing costs, and improving operational efficiency. In today's competitive landscape, effective inventory management is crucial for maintaining a competitive edge and meeting customer demands.

This project seeks to develop an intuitive and user-friendly system that encompasses essential features such as real-time inventory tracking, demand forecasting, automated reorder management, and seamless supplier integration. These functionalities are crucial for businesses to make informed decisions, prevent stockouts or overstocking, and streamline their interactions with suppliers. The Inventory Management System will leverage state-of-the-art technology, including data analytics and predictive algorithms, to provide real-time insights into inventory levels and movements. By forecasting demand with precision, businesses can optimize their stocking levels, reduce carrying costs, and enhance customer satisfaction. Automation is at the heart of this system, automating the replenishment process by generating purchase orders when inventory falls below predefined thresholds. This eliminates manual errors and ensures timely restocking, ultimately improving efficiency.

Furthermore, the system emphasizes collaboration with suppliers, facilitating automated order placement and tracking. This not only simplifies communication but also strengthens supply chain relationships, contributing to a more robust and efficient inventory management ecosystem. Data security and privacy are paramount, and the system will employ a robust database management system to safeguard user data, inventory information, and interactions. User-centric privacy settings will give businesses control over their data, instilling confidence in the system's security.

With real-time inventory tracking capabilities, the system provides a bird's-eye view of inventory levels, locations, and movements, ensuring that businesses always have up-to-the-minute insights into their stock. This real-time visibility enables timely decision-making, preventing costly overstocking and the inconvenience of stockouts.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Social media platforms have evolved into dynamic and interactive online spaces that bring together people from all walks of life, transcending geographical boundaries and cultural barriers. The Social Media Application project envisions tapping into this global phenomenon by developing a user-centric platform that caters to the needs and preferences of individuals, businesses, and communities alike.

In the era of digital transformation, social media platforms have become an integral part of modern society, revolutionizing the way people connect, communicate, and share information. The Social Media Application project aims to create a feature-rich platform that facilitates virtual interactions, content sharing, and user engagement within a diverse online community. This comprehensive introduction provides an overview of the project's objectives, significance, and the growing influence of social media in shaping human interactions.

1.2 BACKGROUND

The increasing popularity of social media platforms has fundamentally transformed the dynamics of communication and human relationships. From individuals connecting with friends and family to businesses engaging with their target audience, social media has become a powerful tool for global interactions. The rise of influencers, viral content, and online communities showcases the potential of these platforms in shaping trends, opinions, and even societal movements. The Social Media Application project aims to tap into this growing trend by offering a unique and user-centric experience that caters to diverse audiences.

1.3 OBJECTIVES

The primary objective of the Social Media Application project is to develop a user-friendly and intuitive platform that fosters connections and interactions among users. The application will offer essential social media functionalities, including user profiles, news feed, content sharing, likes, comments, and privacy settings. By providing

a wide range of features, the application seeks to cater to individuals of all age groups, from casual users to social media enthusiasts.

The project team aims to create a platform that not only fulfils the core requirements of a social media application but also brings innovation and enhanced user experience to the forefront. The application's design and features will be guided by user feedback and market research, ensuring that it meets the evolving needs of the target audience.

1.4 SIGNIFICANCE

The significance of the Social Media Application lies in its potential to bridge geographical boundaries and facilitate meaningful connections among individuals, communities, and businesses. The platform will enable users to share their thoughts, ideas, and experiences, fostering a sense of belonging and online identity. Furthermore, the application's interactive elements, such as likes, comments, and shares, will encourage engagement and promote user-generated content, enhancing the overall user experience.

1.5 TARGET AUDIENCE

The target audience for the Social Media Application comprises individuals of all age groups who are interested in connecting with others, sharing content, and engaging in online conversations. From tech-savvy millennials to seniors looking to reconnect with old friends, the application's design and features will cater to a broad spectrum of users.

The platform will be designed to accommodate both casual users seeking a simple and intuitive social media experience, as well as enthusiasts who wish to engage more deeply with the platform's interactive features. Businesses and content creators will also find value in the platform's potential to reach a wide audience and foster meaningful relationships with their customers.

1.6 PROJECT SCOPE

The scope of the Social Media Application project encompasses the design, development, and deployment of a web-based platform accessible across various devices and operating systems. The application will be built using modern web technologies and frameworks, ensuring a scalable and efficient system architecture. To optimize user experience, the project will involve conducting user experience (UX) analysis, gathering feedback, and conducting usability testing to identify pain points and preferences.

The application's scope extends to providing essential social media features, such as user profiles, news feeds, content sharing, likes, comments, and privacy settings. Additionally, the platform will incorporate modern graphic design elements and an intuitive user interface, contributing to an enjoyable and visually appealing user experience.

1.7 METHODOLOGY

The development process will follow an agile methodology, enabling incremental progress and iterative improvements based on user feedback. Regular sprints will allow the development team to adapt to changing requirements, incorporate new features, and prioritize user needs.

1.8 ORGANIZATION OF THE REPORT

This project report is structured to provide a comprehensive understanding of the Social Media Application project. It will include sections on project overview, literature review, system analysis, system design, implementation details, testing procedures, conclusion, and future work. Each section will delve into specific aspects of the project, detailing the rationale behind decisions and the technical aspects of implementation.

The report will provide an in-depth analysis of the existing literature on social media platforms, user behaviors, and best practices in user interface design. It will also outline the system requirements and project objectives to serve as a foundation for the subsequent stages of development.

CHAPTER 2

REQUIREMENTS

2.1 INTRODUCTION

The Requirements document outlines the functional and non-functional requirements of the Social Media Application project. It provides a detailed overview of the features and capabilities that the application must possess to deliver an engaging and user-centric social media experience. The document aims to define the scope of the project, outline user expectations, and guide the development team in building a successful platform.

2.2 PROJECT OVERVIEW

The Social Media Application project aims to create a feature-rich platform that allows users to connect, share content, and interact with others in a virtual community. It will provide essential social media functionalities such as user profiles, news feed, content sharing, likes, comments, and privacy settings. The application will be accessible through web browsers and will cater to users of various devices and operating systems.

2.3 USER REQUIREMENTS

2.3.1 USER REGISTRATION AND AUTHENTICATION

The application should allow users to register with a unique email address or through third-party authentication services (e.g., Google, Facebook). User authentication must be secure, and passwords should be stored using industry-standard encryption techniques. Users should have the option to reset their passwords in case of forgetfulness or security concerns.

2.3.2 USER PROFILES

Each user should have a personalized profile where they can add their name, profile picture, bio, and other optional details. Users must have the ability to edit and update their profile information as needed.

2.3.3 INVENTRY MANAGEMENT

The inventory management system should present a well-structured catalog of products and items, including detailed information like product name, description, quantity, and supplier details. It must support essential inventory operations like adding, updating, and tracking items, as well as offering insights into inventory trends and forecasting for efficient supply chain management.

2.3.4 INTERACTIONS AND ENGAGEMENT

Users should be able to like, comment on, and share posts from other users. The application must provide real-time updates for new interactions (likes, comments, shares) on a user's posts. Comments should support multimedia attachments and be formatted in a visually appealing manner.

2.3.5 PRIVACY AND SECURITY

Users should have control over their privacy settings, allowing them to manage who can view their profile, interact with their posts, and send friend requests. The application must implement security measures to safeguard user data and prevent unauthorized access.

2.3.6 SEARCH AND DISCOVERY

The application should offer a search functionality that allows users to find specific users, posts, or topics of interest. The discovery feature should enable users to explore trending content and discover new users based on their interests.

2.4 NON-FUNCTIONAL REQUIREMENTS

2.4.1 PERFORMANCE

The application must be responsive and load quickly, even during peak usage times. The platform should be able to handle a large number of concurrent users without significant performance degradation.

2.4.2 SCALABILITY

The system should be designed to scale horizontally to accommodate future growth in user base and content.

2.4.3 RELIABILITY

The application should have high availability, with minimal downtime for maintenance or updates. Data backups should be performed regularly to prevent data loss in case of system failures.

2.4.4 SECURITY

The application must implement robust security measures to protect user data and prevent unauthorized access. User authentication and authorization must be secure to prevent account breaches.

2.4.5 USABILITY

The user interface should be intuitive and user-friendly, allowing users of all ages and technical backgrounds to navigate the platform effortlessly. The application should adhere to modern design standards and best practices for optimal user experience.

2.4.6 ACCESSIBILITY

The platform should be accessible to users with disabilities, complying with web accessibility guidelines (e.g., WCAG 2.1).

2.4.7 COMPATIBILITY

The application should be compatible with major web browsers and accessible on various devices, including desktops, laptops, tablets, and smartphones.

2.4.8 PERFORMANCE METRICS

The application should be equipped with performance monitoring tools to measure response times, server load, and user engagement metrics.

2.5 CONSTRAINTS

2.5.1 TECHNOLOGY STACK

The project must be developed using React JS for the frontend and Spring Boot for the backend. The database management system must be MySQL or PostgreSQL.

2.5.2 BUDGET AND TIMEFRAME

The development of the Social Media Application is constrained by the available budget and timeframe. The project should adhere to the predefined timeline to meet the set deadlines.

2.6 ASSUMPTIONS

Users have a basic understanding of how to use social media platforms and web applications. The project team will have access to necessary development tools, hardware, and software licenses.

2.7 DEPENDENCIES

The project's successful implementation depends on the availability of required APIs for third-party authentication and other integrations. The project team's coordination and collaboration are critical to meeting the project's objectives within the specified timeframe.

2.8 RISKS

- Data breaches and security vulnerabilities may expose user data to unauthorized access.
- Inadequate testing may lead to bugs and performance issues, impacting user experience.
- User engagement and adoption may be affected by competition from existing social media platforms.

CHAPTER 3

MODULE DESCRIPTIONS

3.1 PROPOSED SYSTEM

The Inventory Management project consists of various modules that work together to deliver a seamless and engaging user experience. Each module represents a distinct functional component of the application, and this document provides a detailed description of each module's features, functionalities, and interactions.

3.1.1 USER AUTHENTICATION MODULE

3.1.1.1 DESCRIPTION:

The User Authentication module is responsible for handling user registration, login, and authentication processes. It ensures that only registered users with valid credentials can access the application's features and functionalities.

3.1.1.2 FEATURES:

User Registration: Allows new users to create accounts by providing essential details like email, username, and password.

Login: Existing users can log in to the application using their registered email and password.

Third-party Authentication: Enables users to log in through third-party services like Google or Facebook.

Password Reset: Allows users to reset their passwords in case they forget them.

3.1.1.3 INTERACTIONS:

When a new user registers, the module stores the user's information securely in the database. During login, the module verifies the user's credentials against the stored information to grant access.

3.1.2 USER PROFILE MODULE

3.1.2.1 DESCRIPTION:

The User Profile module manages user profile information and settings. It allows users to personalize their profiles, manage privacy settings, and update their details.

3.1.2.2 FEATURES:

Profile Creation: Users can create personalized profiles with their names, profile pictures, bios, and optional information. Profile Editing: Users can modify their profile information and update their profile pictures. Privacy Settings: Allows users to control the visibility of their profile others and manage interaction settings.

3.1.2.3 INTERACTIONS:

When a user creates or edits their profile, the module updates the user's information in the database. Privacy settings are applied to control the visibility of a user's profile and their posts in the news feed.

3.2.1 INVENTORY MANAGEMENT MODULE

3.2.1.1 DESCRIPTION:

The Inventory Management module is responsible for efficiently managing and organizing the inventory of products and items within the system. It ensures that inventory items are accurately cataloged and readily accessible for inventory-related operations.

3.2.1.2 FEATURES:

Catalog Organization: Categorizes and organizes inventory items systematically, including essential details like product name, description, quantity, and supplier information.

Inventory Operations: Supports fundamental inventory tasks such as adding new items, updating item details, and tracking stock levels. Reporting and Analytics: Provides insights into inventory trends, turnover rates, and forecasting to aid in supply chain management.

3.2.1.3 INTERACTIONS:

The module interacts with the database to retrieve and display comprehensive information about inventory items, facilitating easy item management. Users can perform inventory-related tasks, including adding new items or updating existing ones, directly through the module. The module generates reports and analytics to provide users with valuable insights into inventory performance and helps streamline inventory operations.

3.3.1 BILLING MODULE

3.3.1.1 DESCRIPTION:

The Billing Module is responsible for managing and processing financial transactions within the system. It ensures accurate and efficient billing for products or services offered, keeping track of payments and invoices.

3.3.1.2 FEATURES:

Transaction Processing: Handles financial transactions for products or services, ensuring accuracy in billing and payments. Invoice Generation: Generates invoices for customers, including itemized details of products or services, quantities, prices, and total amounts. Payment Tracking, Records and tracks payments made by customers and updates the billing status accordingly. Billing Reports: Provides comprehensive billing reports and financial insights for efficient financial management.

3.3.1.3 INTERACTIONS:

The module interacts with user accounts and product or service databases to create invoices based on purchases or services rendered. Users can view and pay invoices through the module, with payment updates reflected in real-time. Billing reports and financial data are accessible to authorized users, offering valuable insights into revenue generation and financial health.

3.2.2 INVENTORY SECURITY MODULE

3.2.2.1 DESCRIPTION:

The Inventory Security module is dedicated to safeguarding sensitive inventory data and ensuring secure access control.

3.2.2.2 FEATURES:

Access Control: Provides robust access control mechanisms, allowing administrators to define user roles and permissions for inventory-related actions. Data Encryption: Implements data encryption protocols to protect inventory data from unauthorized access. Audit Trail: Maintains an audit trail to track user interactions and changes made to the inventory system, ensuring accountability.

3.2.2.3 INTERACTIONS:

Administrators configure user roles and permissions through the module, defining who can perform specific inventory management tasks. Data encryption measures are applied to secure sensitive inventory data, preventing unauthorized access. The audit trail keeps a record of all inventory-related activities and user interactions, assisting in monitoring and maintaining data integrity.

3.1.3 Reordering and Stock Alerts Module

3.1.3.1 DESCRIPTION:

The Reordering and Stock Alerts module is responsible for monitoring inventory levels and ensuring that stock is replenished efficiently to prevent stockouts and overstocking.

3.1.3.2 FEATURES:

Stock Monitoring: Continuously tracks inventory levels for all products and items in the system. Reorder Point Management: Allows users to set reorder points for each item, indicating the minimum quantity at which an item should be reordered.

Automatic Reordering: Automatically generates purchase orders or replenishment requests when inventory falls below the defined reorder point. Stock Alerts: Sends notifications and alerts to designated users when inventory reaches critical levels or when restocking is required.

3.1.3.3 INTERACTIONS:

Users can configure reorder points for items, specifying the desired quantity for restocking. When inventory levels drop below the reorder point, the module generates purchase orders or alerts designated personnel. Stock alerts are sent via email, SMS, or within the system, ensuring timely action to maintain optimal inventory levels. The Reordering and Stock Alerts module plays a crucial role in preventing stockouts and excess inventory, optimizing inventory turnover, and ensuring that products are always available when needed.

3.1.4 ALERTS AND NOTIFICATIONS MODULE

3.1.4.1 DESCRIPTION:

The Alerts and Notifications module plays a vital role in keeping users informed about critical inventory-related events and updates in real-time.

3.1.4.2 FEATURES:

Real-time Alerts: Sends immediate alerts to users for inventory-related activities such as low stock levels, item expirations, and order status changes. Customizable Alerts: Allows users to personalize their notification preferences, specifying which events trigger alerts

Threshold Alerts: Monitors inventory thresholds and sends alerts when stock levels reach predefined minimum or maximum thresholds.

Order Status Updates: Provides timely notifications regarding order processing, shipment, and delivery status.

3.1.4.3 INTERACTIONS:

When inventory levels drop below set thresholds, the module sends instant alerts to designated users, ensuring timely restocking. Users can define their preferred notification settings, choosing which events they want to receive alerts for. Order status updates, including processing, shipping, and delivery, are communicated to users through real-time notifications, enhancing transparency in the supply chain.

3.1.5 FEEDBACK MODULE

3.1.5.1 DESCRIPTION:

The Feedback and Issue Reporting module enables users to provide valuable feedback, report inventory-related issues, and submit suggestions for enhancing the inventory management system.

3.1.5.2 FEATURES:

Feedback Submission: Users can easily submit feedback, report bugs, and offer suggestions through a dedicated feedback portal.

Issue Tracking: The module records and tracks reported issues, assigning them to relevant personnel for resolution.

Suggestion Management: Manages user suggestions, facilitating the review and implementation of valuable system enhancements.

3.2.4.3 INTERACTIONS:

When a user submits feedback or reports an issue, the module captures and stores the information for review and action. Assigned personnel can access reported issues and suggestions through the module, ensuring timely resolution and system improvement. The Feedback and Issue Reporting module promotes user engagement and helps enhance the efficiency and effectiveness of the inventory management system.

CHAPTER 4

DESIGN OF PROPOSED SYSTEM

4.1 OVERVIEW

4.1.1 IMPORTANCE OF A WELL-DESIGNED SYSTEM IN A SOCIAL MEDIA PLATFORM.

A well-designed inventory management system is a critical asset for any organization. It serves as the linchpin of efficient operations, enabling businesses to effectively track, control, and optimize their inventory. Such a system ensures that resources are allocated judiciously, reducing waste and costs. Moreover, it empowers decision-makers with real-time insights and forecasts, helping them make informed choices that positively impact the bottom line. By fostering accuracy, compliance, and customer satisfaction, a well-designed inventory management system lays the foundation for success in a competitive business environment.

4.2 USER EXPERIENCE (UX) ENHANCEMENT:

In the realm of inventory management, a well-designed system places paramount importance on user experience (UX). A user-friendly interface with streamlined workflows, clear data visualization, and responsive design ensures that users can efficiently navigate the system, enhancing their satisfaction and productivity. Customization options allow users to tailor their experience, while robust training resources and support channels empower them to use the system confidently. A strong focus on UX ultimately leads to more engaged and efficient users, contributing to the success of inventory management operations.

4.3 EFFICIENT CONTENT MANAGEMENT:

In the context of efficient content management, a well-designed system places a central emphasis on user experience (UX). It provides an intuitive and user-friendly interface, enabling streamlined content creation, organization, and distribution. Clear data presentation, robust search functionalities, and responsive design ensure that users can easily manage content, enhancing their productivity and satisfaction. Customization options allow users to tailor their content management experience, while comprehensive training resources and responsive support channels empower them to confidently use the system. A strong focus on UX ultimately leads to more engaged and efficient content management, contributing to the seamless flow of information within an organization.

4.4 SCALABILITY AND PERFORMANCE:

In the domain of scalability and performance, a well-designed system is paramount. For efficient content management, the system incorporates scalable architecture and optimized algorithms to ensure seamless performance, even when dealing with substantial data volumes and increased user demands.

4.5 SECURITY AND PRIVACY:

Security and privacy are of utmost importance in efficient content management. The system is meticulously designed to incorporate robust security measures, encryption protocols, and access controls. These safeguards are essential for protecting sensitive content and user information, preventing unauthorized access, and ensuring data integrity.

4.6 REAL-TIME INTERACTIVITY:

Efficient content management demands real-time interactivity, allowing users to create, edit, and collaborate on content seamlessly. The system integrates real-time updates, ensuring users receive immediate notifications of content changes and interactions, thereby fostering a dynamic and engaging content management environment.

4.7 OBJECTIVES OF THE DESIGN PHASE:

The objectives of the design phase in efficient content management are as follows:

4.7.1 USER-CENTRIC APPROACH:

The design phase prioritizes understanding user needs and behaviors. By gathering user feedback and conducting usability testing, the objective is to create a user-centric interface that enhances user satisfaction and maximizes productivity in content management.

4.7.2 EFFICIENT SYSTEM ARCHITECTURE:

This phase establishes the foundation for an efficient and scalable system architecture. It entails selecting appropriate technologies, frameworks, and database systems to ensure efficient data processing and system performance during content management.

4.7.3 SEAMLESS USER INTERACTION:

The aim is to craft a seamless user interaction experience, allowing users to navigate the system effortlessly, manage content efficiently, and collaborate with others naturally and intuitively during content management.

4.7.4 DATA MANAGEMENT AND SECURITY:

The design phase concentrates on defining data structures, establishing efficient data management processes, and implementing robust security protocols. These measures safeguard sensitive content, protect user data, and prevent data breaches.

4.8 SCOPE OF THE PROPOSED SYSTEM:

The scope of the content management system encompasses the development of a feature-rich and user-friendly platform. Key functionalities of the proposed system include:

4.8.1 USER PROFILES:

Users can create and manage personalized profiles, including profile pictures and privacy settings.

4.8.2 CONTENT ORGANIZATION:

Users will access a well-organized repository of content, categorized logically for easy discovery and management**.**

4.8.3 CONTENT CREATION:

Users can efficiently create and publish various types of content, including text, images, videos, and links.

4.8.4 COLLABORATION:

The system supports collaborative content creation and editing, enabling users to work together seamlessly.

4.8.5 SECURITY AND ACCESS CONTROL:

Users will have control over access to their content, ensuring that privacy and security are maintained throughout the content management process.

4.8.6 SEARCH AND DISCOVERY:

The system provides robust search and discovery functionalities, enabling users to find relevant content easily and discover new content based on their interests and preferences.

4.9 THREE-TIER ARCHITECTURE EXPLANATION

Three-tier architecture, also known as multi-tier architecture, is a widely used software design pattern that divides an application into three separate layers: Presentation Layer, Application Layer, and Data Layer. Each layer has distinct functionalities and communicates with the other layers through well-defined interfaces. This architectural approach enhances the modularity, scalability, and maintainability of the system.

4.9.1 PRESENTATION LAYER:

The Presentation Layer, also known as the user interface (UI) layer, is the topmost layer that interacts directly with the end-users. Its primary focus is to present information to users in a user-friendly and visually appealing manner. This layer handles user input and displays the results from the Application Layer to the users. It is responsible for gathering user input, processing it, and sending requests to the Application Layer for further processing. Technologies like HTML, CSS, JavaScript, and frameworks like React JS or AngularJS are commonly used in this layer to create dynamic and responsive user interfaces. A well-designed Presentation Layer ensures an intuitive and engaging user experience, facilitating seamless interactions with the application.

4.9.2 APPLICATION LAYER:

The Application Layer, also known as the business logic layer, acts as an intermediary between the Presentation Layer and the Data Layer. It contains the core business logic of the application and processes user requests received from the Presentation Layer. This layer is responsible for executing application-specific functionalities, performing calculations, and making decisions based on the data received from the Data Layer. It ensures that business rules and logic are applied consistently throughout the application. In a three-tier architecture, the Application Layer is often implemented using server-side technologies and frameworks like Spring Boot (Java) or Express (Node.js). An efficient Application Layer enhances the overall performance and responsiveness of the application, providing users with a seamless and efficient experience.

4.9.3 DATA LAYER:

The Data Layer, also known as the data access layer, is responsible for managing the storage and retrieval of data. It interacts with the application's data sources, which could be a Relational Database Management System (RDBMS) or any other data storage mechanism. The Data Layer handles data manipulation, storage, and retrieval operations based on the requests received from the Application Layer. It ensures data integrity, security, and consistency, guarding against unauthorized access and data corruption. Commonly used technologies In the Data Layer include SQL for database querying and Object-Relational Mapping (ORM) frameworks like Hibernate or Sequalae. A robust and efficient Data Layer ensures the reliability and security of the application's data, safeguarding sensitive user information.

4.10 UTILIZATION OF REACT JS FOR DYNAMIC UI

React JS is a popular JavaScript library that is widely used for building dynamic and interactive user interfaces. It was developed by Facebook and is now maintained by a community of developers. React JS is based on the concept of components, which are self-contained, reusable pieces of code that encapsulate the UI and its behavior. Here's how React JS is utilized for creating dynamic UI

4.10.1 VIRTUAL DOM:

React JS uses a Virtual DOM to efficiently update the actual DOM. When the state of a component changes, react creates a virtual representation of the UI in memory. It then calculates the difference (diffing) between the virtual and actual DOM and applies only the necessary changes to the real DOM. This process significantly improves rendering performance.

4.10.2 STATE MANAGEMENT:

React JS allows components to have state, which represents the data that can change over time. When the state of a component is updated, react automatically re-renders the component and its children to reflect the changes. This enables the creation of dynamic UIs that respond to user interactions.

4.10.3 DECLARATIVE SYNTAX:

React JS uses a declarative syntax, where developers describe how the UI should look based on its current state. This makes the code more predictable and easier to reason about. Developers only need to specify what should be rendered, and react takes care of updating the UI accordingly.

4.11 RESPONSIVE DESIGN PRINCIPLES

Responsive design is a design approach that aims to create web applications that adapt and respond to different screen sizes and devices. It ensures that the application's user interface looks and functions well across a wide range of devices, from desktop computers to mobile phones and tablets. Here are the principles of responsive design:

4.11.1 FLUID GRIDS:

Responsive design uses relative units like percentages instead of fixed units like pixels for layout and sizing. This allows the UI to adjust fluidly to different screen sizes, ensuring that elements resize proportionally. Flexible Images and Media: Images and media elements in a responsive design are also set to resize dynamically based on the screen size. This prevents images from overflowing or becoming too small on different devices. Media Queries: Media queries are CSS rules that apply specific styles based on the device's characteristics, such as screen size, resolution, and orientation. Media queries enable developers to create custom styles for different devices.

4.11.2 DESIGN CONSTRAINTS AND STANDARDS

In the development of an Instagram-like social media application, it is essential to take into account various design constraints and adhere to industry standards and best practices. These constraints and standards play a pivotal role in ensuring the functionality, security, and overall user experience of the platform. Here, we discuss key design constraints and standards that should guide the development process.

4.11.2.1 DESIGN CONSTRAINTS:

1. Device and Platform Compatibility: One of the foremost design constraints is ensuring compatibility across a wide range of devices and platforms. This includes smartphones, tablets, and desktop computers running various operating systems such as iOS, Android, and popular web browsers like Chrome, Firefox, Safari, and Edge. The

application must adapt seamlessly to different screen sizes, resolutions, and input methods to provide a consistent user experience.

2. Network Constraints: Users may access the application under various network conditions, including high-speed Wi-Fi and slower mobile data connections. Designing the application to perform well under these varying network constraints is essential to ensure that users can access and interact with the platform without interruptions.

3. Data Privacy Regulations: Compliance with data privacy regulations is a critical constraint in the development of a social media application. Regulations like GDPR (General Data Protection Regulation) in Europe and CCPA (California Consumer Privacy Act) in California require stringent protection of user data

4. Content Moderation: Implementing effective content moderation is both a constraint and a necessity. Striking a balance between allowing user freedom and preventing the spread of harmful or inappropriate content is a constant challenge. The application must employ AI-based content moderation algorithms to detect and manage such content while avoiding over-restrictive censorship.

5. Scalability: As the user base of the application grows, scalability becomes a critical constraint. The architecture of the platform must be designed to handle increasing numbers of users, posts, and data without sacrificing performance. Scalability should be achieved through strategies such as load balancing, distributed databases, and microservices.

6. Resource Constraints: Many users access social media applications on mobile devices, which have limited resources such as CPU, RAM, and storage. Efficient resource management is essential to prevent excessive battery drain and ensure smooth performance on these devices.

4.11.2.2 STANDARDS AND BEST PRACTICES:

1. User Interface (UI) Design: Following established UI design standards and best practices is crucial for delivering a user-friendly experience. Adherence to design guidelines specific to the platform (e.g., Material Design for Android, Human Interface Guidelines for iOS) ensures consistency and intuitiveness in user interactions.

2. Accessibility Standards: Ensuring accessibility is a standard that promotes inclusivity. Compliance with accessibility standards such as WCAG (Web Content Accessibility Guidelines) makes the application usable by individuals with disabilities. This includes providing alternative text for images, keyboard navigation, and semantic HTML.

3. Security Standards: Robust security practices are a standard requirement for any application. This includes input validation, authentication, authorization, encryption (using HTTPS), and secure coding practices to protect user data and prevent security breaches.

4. Performance Optimization: Performance optimization is a best practice that focuses on minimizing load times, reducing server requests, and using efficient algorithms. Techniques like lazy loading of images and content caching enhance the application's speed and responsiveness.

5. Responsive Design: Responsive design is a standard approach for ensuring that the application adapts seamlessly to various screen sizes and orientations. It ensures that users have a consistent experience whether they are using a smartphone, tablet, or desktop computer.

6. Code Quality and Documentation: Code quality and documentation are standards that contribute to maintainability and collaboration among developers.

4.11.3 DESIGN ALTERNATIVES

In the development of an inventory management system, it is crucial to explore various design alternatives to inform decisions regarding system architecture, features, and user experience. This section outlines key design alternatives and considerations that can shape the project's direction.

1. Monolithic vs. Microservices Architecture:

- Monolithic Architecture: A monolithic system is a single, cohesive unit, simplifying development but potentially limiting scalability as the system grows. It may be suitable for smaller inventory management needs.

- Microservices Architecture: Microservices break down the system into smaller, independent services communicating through APIs. This approach enhances scalability but introduces complexity in managing multiple services and inter-service communication.

2. Native Desktop Apps vs. Web-Based System:

- Native Desktop Apps: Developing separate native applications for different desktop platforms (e.g., Windows, macOS) provides optimal performance and access to platform-specific features but requires more development effort.

- Web-Based System: Creating a web-based system allows universal access through web browsers but may sacrifice some performance and offline capabilities.

3. Data Storage and Databases:

- Relational Databases: Traditional relational databases like PostgreSQL or MySQL are suitable for structured data and well-defined inventory systems but may require complex schema design.

- NoSQL Databases: NoSQL databases such as MongoDB offer flexibility for unstructured data and can handle user-generated content efficiently, making them ideal for inventory management.

4. Authentication and Access Control:

- Role-Based Access Control (RBAC): Implementing RBAC allows fine-grained control over user permissions, enhancing security and compliance.

- Single Sign-On (SSO): Utilizing SSO solutions simplifies user authentication and reduces the need for users to remember multiple login credentials.

5. Real-Time Updates:

- WebSocket: Implementing WebSocket technology enables real-time updates, providing instant notifications of inventory changes. This approach ensures dynamic user experiences but adds backend complexity.

- Polling: Using polling mechanisms to check for inventory updates simplifies backend architecture but may introduce latency.

6. Inventory Forecasting Algorithms:

- Historical Data Analysis: Using historical data to predict future demand is a common approach, but it may not capture sudden market changes or trends.

- Machine Learning Models: Implementing machine learning algorithms for demand forecasting allows the system to adapt to changing conditions but requires substantial data and expertise.

7. Inventory Tracking Technology:

- Barcode Scanning: Utilizing barcode scanning for inventory tracking is efficient and cost-effective, but it may not provide real-time tracking.

- RFID Technology: RFID technology offers real-time tracking capabilities but can be more expensive to implement.

8. Inventory Reporting:

- Predefined Reports: Offering predefined reports for common inventory metrics simplifies reporting but may not cover all unique business requirements.

9. Mobile Access:

- Mobile App: Developing a dedicated mobile app for inventory management ensures optimized performance and features but requires additional development resources.

- Mobile-Responsive Web App: Creating a mobile-responsive web app allows access from mobile devices with a unified codebase but may sacrifice some performance and features.

10. Data Backup and Recovery:

- Cloud-Based Backup: Storing data backups in the cloud enhances data security and disaster recovery but may incur ongoing costs.

- On-Premises Backup: Local, on-premises backup solutions offer more control but may require additional hardware and maintenance

.

4.11.3 MODULE DESCRIPTION AND DATA ESTIMATION PROCESS

In software development, a module is a self-contained unit of code that performs a specific function or task within an application. This module-based approach helps in organizing and managing the complexity of large software systems. In this discussion, we will delve into the importance of module description and the data estimation process when designing and developing software modules.

4.11.3.1 MODULE DESCRIPTION:

A module description is a detailed documentation of a software module's purpose, functionality, inputs, outputs, dependencies, and interactions with other modules. It serves as a blueprint for developers, testers, and stakeholders, helping them understand the module's role within the system and its contribution to achieving overall project goals.

4.11.3.2 KEY COMPONENTS OF A MODULE DESCRIPTION:

1. Module Name: A unique identifier for the module.

2. Module Purpose: A concise statement that describes the module's primary objective or function within the application.

3. Functionality: A detailed explanation of what the module does, including its main features and capabilities.

4. Inputs: A list of the data or information that the module requires to perform its functions. This may include parameters, user inputs, or data from other modules.

5. Outputs: A description of the results, data, or actions produced by the module.

6. Dependencies: Identification of other modules or external components that the module relies on. This helps in understanding the interdependencies within the software system.

7. Interactions: How the module communicates or interacts with other modules, databases, or external services. This includes details on data exchange formats and communication protocols.

8. Error Handling: Strategies for handling errors, exceptions, and unexpected scenarios within the module.

9. Testing Requirements: Guidelines for testing the module, including test cases and expected outcomes.

10. Security Considerations: Any security measures or precautions specific to the module, such as access controls or data encryption.

Importance of Module Description:

1. Clarity and Understanding: A well-documented module description enhances clarity and ensures that all stakeholders, including developers, testers, project managers, and clients, have a common understanding of the module's purpose and functionality.

2. Efficient Development: Developers can work more efficiently when they have a clear module description. It serves as a guide for writing code, making it easier to implement and debug the module.

3. Effective Testing: Testers can design comprehensive test cases based on the module description, covering all possible scenarios and ensuring thorough testing.

4. Collaboration: Module descriptions facilitate collaboration among team members, as they provide a common reference point for discussions and decision-making.

5. Documentation: Module descriptions serve as valuable documentation for future maintenance and updates, helping new team members understand the module's behavior.

4.11.3.3 DATA ESTIMATION PROCESS:

Data estimation is a critical step in software development that involves predicting the amount of data the application will handle, store, and process. Accurate data estimation is essential for designing database schemas, selecting appropriate storage solutions, and ensuring that the application performs optimally.

Steps in the Data Estimation Process:

1. Requirements Analysis: Begin by thoroughly analyzing the project's requirements. Identify the types of data the application will handle, such as user profiles, posts, comments, images, and user interactions.

2. Data Modeling: Create a data model that represents the relationships between different types of data. Use techniques like Entity-Relationship Diagrams (ERDs) to visualize the data structure.

3. Data Volume Estimation: Estimate the volume of data that the application is expected to generate and store. Consider factors such as the

expected number of users, the frequency of data creation, and data retention policies.

4. Data Growth Projection: Predict how the volume of data will grow over time. This projection should take into account factors such as user acquisition rates and usage patterns.

5. Data Types and Schemas: Define the data types and database schemas that will be used to store the data. This includes specifying field types, constraints, and indexes.

6. Data Storage Solutions: Select appropriate data storage solutions based on the estimated data volume and requirements. Choose between relational databases (e.g., MySQL, PostgreSQL), NoSQL databases (e.g., MongoDB, Cassandra), cloud storage (e.g., Amazon S3), or a combination of these.

7. Data Access Patterns: Consider how data will be accessed and retrieved by different parts of the application. Optimize data access patterns to minimize latency and ensure efficient data retrieval.

8. Data Backup and Recovery: Develop data backup and recovery strategies to protect against data loss or corruption. Define backup schedules and disaster recovery plans.

4.11.3.4 BENEFITS OF DATA ESTIMATION:

1. Resource Planning: Accurate data estimation helps in resource planning, including hardware provisioning, database design, and infrastructure scaling.

2. Cost Management: It allows for better cost management by avoiding over-provisioning of resources and optimizing storage costs.

3. Performance Optimization: Data estimation helps identify potential performance bottlenecks early in the development process, allowing for proactive optimization.

4. Scalability: Planning for data growth ensures that the application can scale seamlessly to accommodate increasing user data.

5. Data Integrity: It contributes to data integrity and reliability by implementing appropriate data storage and backup strategies.

In conclusion, module descriptions and data estimation are essential aspects of software development. A well-documented module description ensures clarity and effective collaboration among team members, while accurate data estimation is crucial for building scalable, performant, and cost-effective applications. These processes are fundamental in delivering successful software projects that meet both user and business requirements.

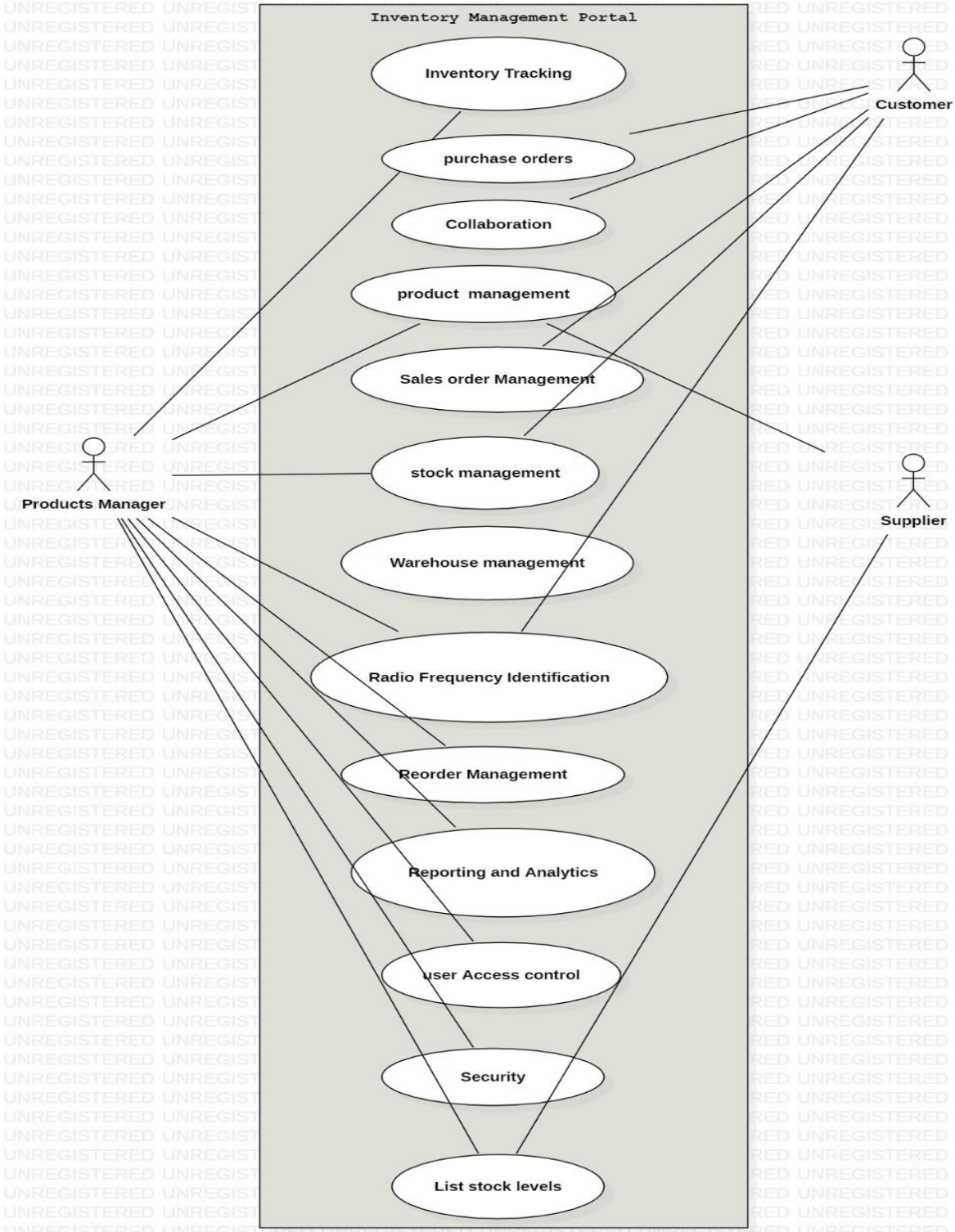

Fig 4.1: USECASE DIAGRAM

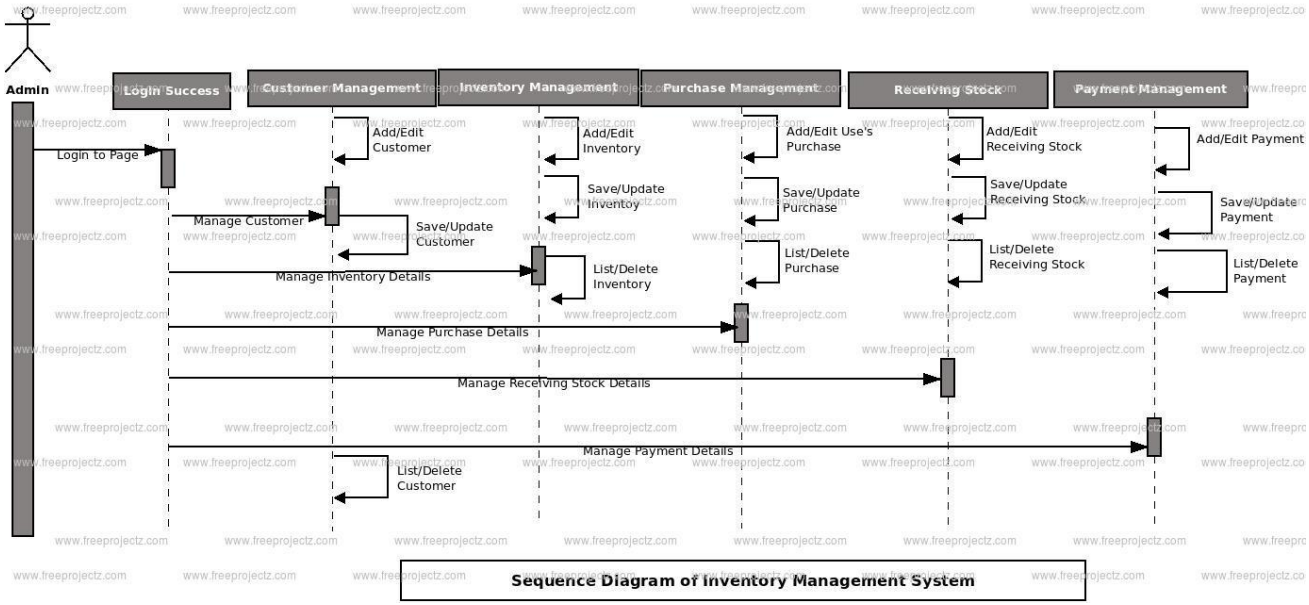

Fig 4.2: SEQUENCE DIAGRAM

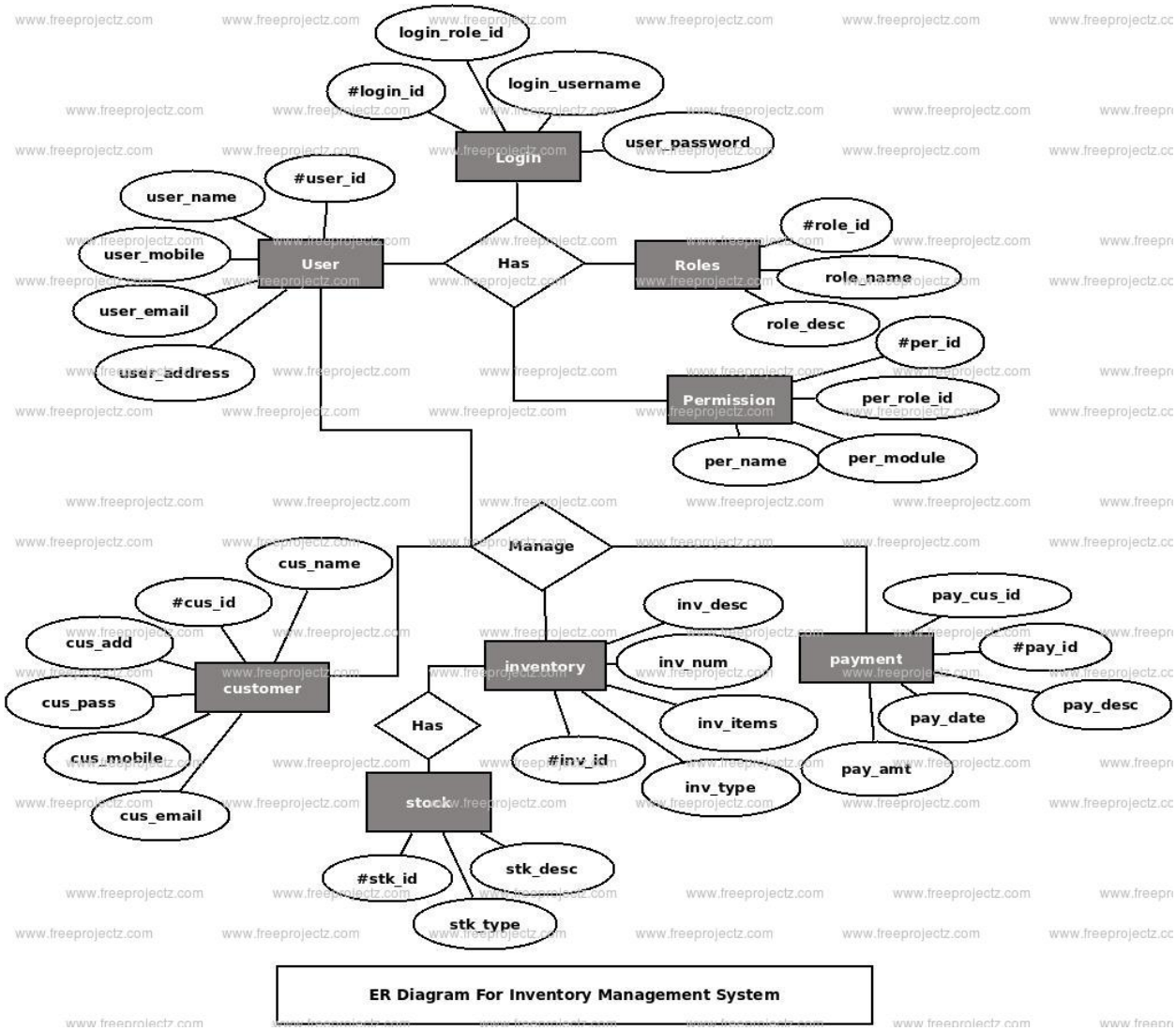

Fig 4.3: ER DIAGRAM

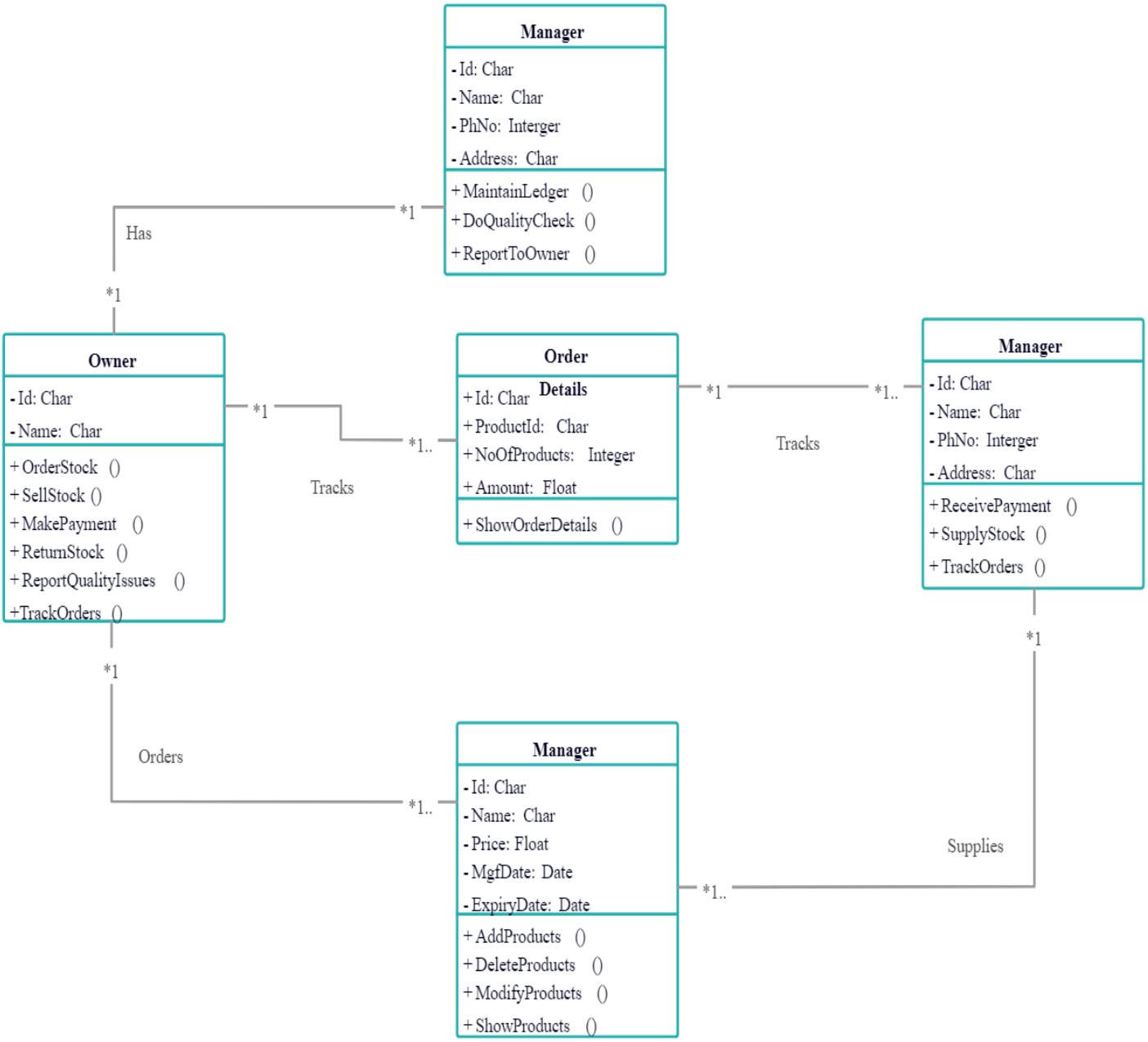

Fig 4.4: CLASS DIAGRAM

CHAPTER – 5

IMPLEMENTATION

5.1 INTRODUCTION

In today's interconnected business landscape, effective inventory management has become a cornerstone of operational success, offering a systematic approach to control, optimize, and streamline the management of goods and resources. Developing an Inventory Management System is a complex yet essential endeavor that empowers organizations to harness the power of cutting-edge technology to efficiently manage their inventory, reduce costs, and enhance overall operational efficiency.

In this project, we embark on the journey of creating an Inventory Management System using a meticulously selected stack of tools and frameworks. Our frontend will be powered by React, renowned for its flexibility and performance, providing an intuitive and user-friendly interface.

On the backend, Spring Boot Microservices will form the backbone of our system, ensuring scalability, security, and seamless data management. For the database, we have chosen MySQL, a reliable relational database system, to efficiently store and retrieve critical inventory data.

These technologies will collaboratively enable us to craft an Inventory Management System that not only optimizes inventory control but also enhances decision-making, reduces costs, and ensures a streamlined supply chain.

This project represents a voyage into the realm of modern inventory management, where innovation converges with user-centric design, and where technology bridges the gap between organizations and efficient resource management, ultimately leading to enhanced competitiveness and operational excellence.

5.2 TOOLS AND FRAMEWORKS INVOLVED IN SYSTEM

In the development of a Social Media application using React for the frontend, Spring Boot Microservices for the backend, and MySQL for the database, several tools and frameworks play a pivotal role in creating a robust and feature-rich system. Here's a brief overview of each:

5.2.1 REACT

React is a popular JavaScript library for building user interfaces. It's used for creating the user interface of the Social Media application, allowing developers to design interactive and dynamic web pages. It allows developers to create interactive and dynamic web applications with reusable UI components. React provides a component-based architecture, making it easier to manage and scale complex frontend applications. Reacts virtual DOM ensures efficient updates, making it suitable for complex user interfaces. It fosters a component-based architecture that simplifies code management and encourages a modular approach to frontend development. One of Reacts notable advantages is its capability to efficiently update the DOM when the underlying data changes. It accomplishes this by comparing a virtual representation of the DOM with the real DOM and then updating only the parts that have changed. This process minimizes the need for expensive and time-consuming manipulations of the actual DOM, resulting in a snappy and responsive user experience.

5.2.2 SPRINGBOOT

Spring Boot is a Java-based framework that simplifies the development of backend microservices. It provides tools and conventions for quickly building and deploying Java applications. Spring Boot is well-suited for creating the server-side components of the Social Media application, handling user authentication, content storage, and data retrieval. It provides a wide range of tools and conventions for building and deploying Java applications, including RESTful APIs. Spring Boot promotes rapid development and offers features like dependency injection, security, and data access. It's well-suited for creating scalable, secure, and maintainable backend services. Scalability is another area where Spring Boot shines. As the Social Media application gains users and data, the ability to scale the backend becomes paramount. Spring Boot's architecture and support for microservices ensure that the backend can handle increased loads without sacrificing performance or reliability.

5.2.3 MYSQL

MySQL is an open-source relational database management system (RDBMS). It is used to store and manage user data, posts, comments, and other essential information for the Social Media application. MySQL offers robust data storage capabilities and is known for its reliability and performance. In the context of the Social Media application, MySQL will store and manage user data, posts, comments, and other essential information in a structured and organized manner. Moreover, MySQL's ability to handle large datasets is advantageous. As the Social Media application grows and accumulates vast amounts of user-generated content, MySQL's performance ensures that data retrieval remains swift and responsive, maintaining a seamless user experience.

5.2.4 VISUAL STUDIO CODE

Visual Studio Code (VS Code) and GitHub are an indispensable duo in the toolkit of developers working on the Social Media application. These tools streamline the development process, enhance code quality, and foster collaboration within the development team. Visual Studio Code is a lightweight yet powerful code editor developed by Microsoft. Its popularity stems from its extensive support for a wide range of programming languages and its rich ecosystem of extensions. VS Code's versatility makes it an ideal choice for developers working on both frontend (React) and backend (Spring Boot) code. Features such as code navigation, debugging, and integrated version control enhance developer productivity.

5.2.5 GITHUB

GitHub, on the other hand, is a web-based platform for version control and collaboration. It provides essential capabilities for managing code changes and collaborating with team members. Through GitHub, developers can create branches, submit pull requests, conduct code reviews, and merge changes with ease. This process ensures that code quality is maintained and that changes are tracked meticulously. GitHub's integration with continuous integration and deployment (CI/CD) pipelines is

another invaluable asset. It automates the deployment process, allowing for seamless and reliable application updates. CI/CD pipelines ensure that changes are thoroughly tested and verified before they are deployed to the production environment, reducing the risk of introducing bugs or issues.

5.2.6 NODE JS

Node.js is a server-side runtime environment that allows JavaScript code to be executed on the server. While it's not the primary technology in the Social Media application stack, it plays a significant role in various frontend development tasks.

Node.js is particularly useful for setting up build workflows and task automation for the React frontend. Build tools like Webpack and Babel, which are commonly used in React development, are often configured and managed using Node.js. These tools are essential for bundling JavaScript, CSS, and other assets, optimizing them for production, and transpiring modern JavaScript code into versions that are compatible with older browsers. Furthermore, Node.js can be used for server-side rendering (SSR) in React applications. SSR enhances performance and search engine optimization (SEO) by pre-rendering pages on the server before sending them to the client. This approach improves initial page load times and ensures that search engines can index the content effectively. Node.js's lightweight, event-driven architecture makes it well-suited for tasks that involve I/O operations, such as reading and writing files, making network requests, and interacting with databases. In the context of our Social Media application, Node.js can be leveraged for various frontend development workflows and optimizations.

CHAPTER – 6
TESTING

6.1 INTRODUCTION

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive. A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlyingmotivation of program testing is to affirm software quality with methods that can economicallyand effectively apply to both strategic to both large and small-scale systems.

6.2 STRATEGIC APPROACH TO SOFTWARE TESTING

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral in along streamlines that decrease the level of abstractionon each turn. A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at system testing, where the software and other system elements are tested as a whole.

6.3 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing, we have is white box oriented and some modules the steps are conducted in parallel.

6.3.1 WHITE BOX TESTING

To follow the concept of white box testing we have tested each form. We have createdindependently to verify that Data flow is correct and all conditions are exercised to check theirvalidity.

- All loops are executed on their boundaries. This type of testing ensures that allindependent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity paths.

6.3.2 CONDITIONAL TESTING

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested so that each path that may generate on a particular condition is traced to uncover any possible errors. This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variables were declared. The definition-use chain method was used in this type of testing.

6.3.3 DATA FLOW TESTING

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variables were declared.

6.3.4 LOOP TESTING

In this type of testing all the loops are tested to all the limits possible. The followingexercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them. All theloops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards. For concatenatedloops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested asabove.
- Each unit has been separately tested by the development team itself and all the inputhave been validated.

6.4 TEST CASE

A test case, in software engineering, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is workingas it was originally established for it to do. The mechanism for determining whether a softwareprogram or system has passed or failed such a test is known as a test oracle. In some settings, an oracle could be a requirement or use case, while in others it could be a heuristic. It may take many test cases to determine that a software program or system is considered sufficiently scrutinized to be released. Test cases are often referred to as test scripts, particularly when written - when they are usually collected into test suites.

6.4.1 TEST CASE 1

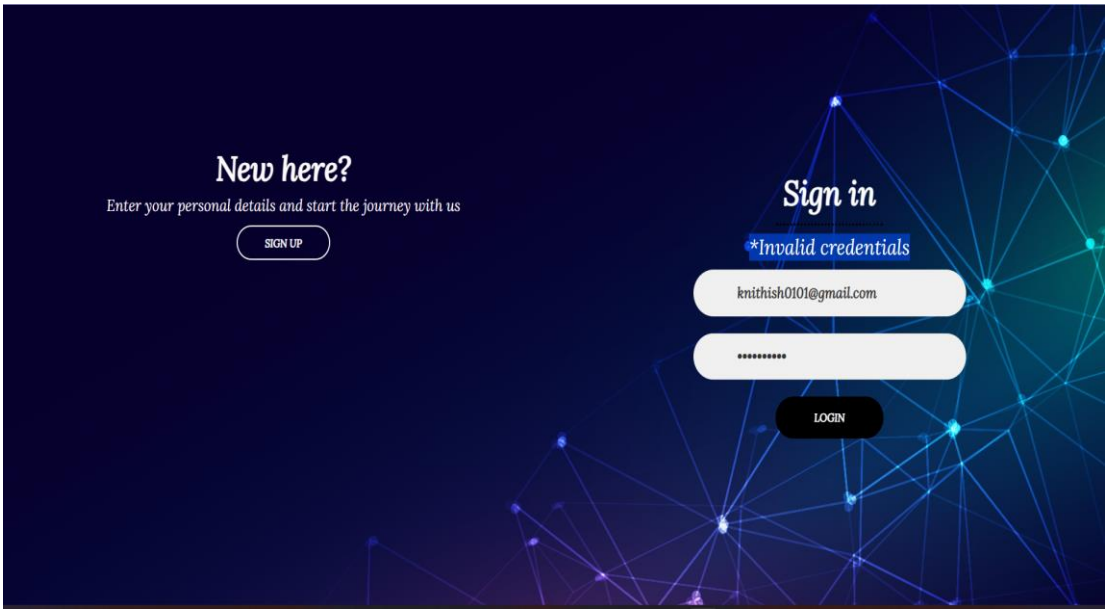

Fig 5: TEST CASE 1

EXPECTED: *Invalid credentials

ACTUAL: *Invalid credentials

6.4.2 TEST CASE 2

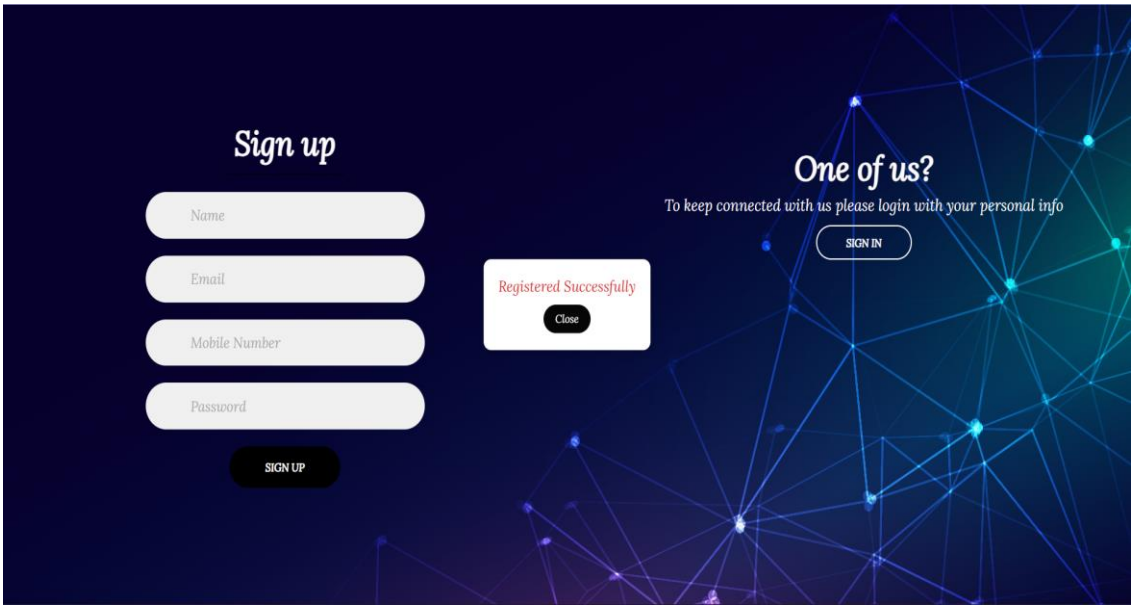

Fig 6: TEST CASE 2(i)

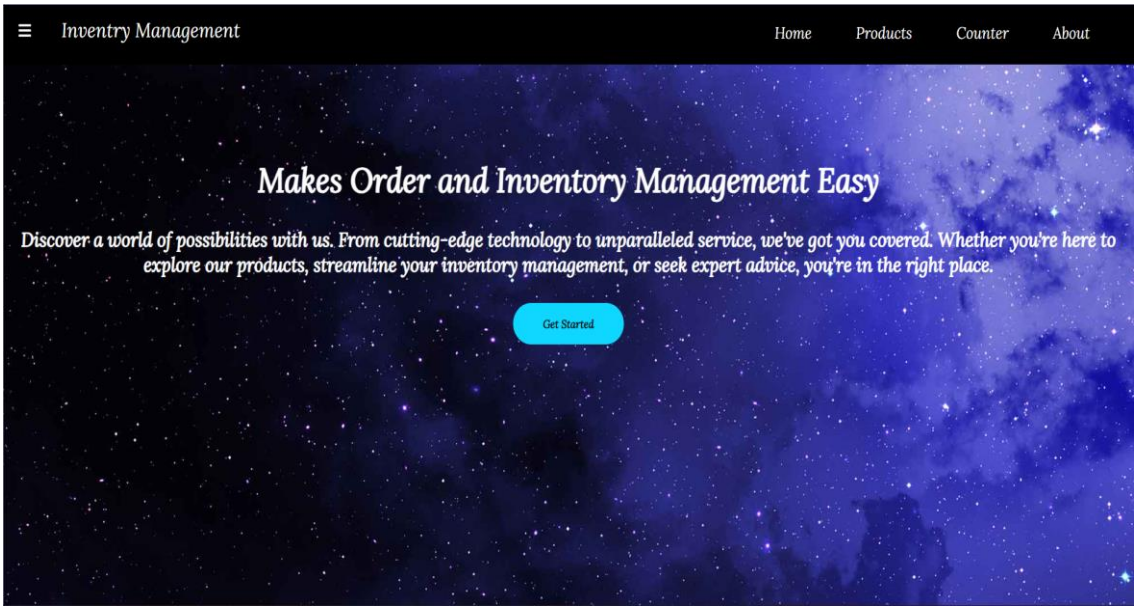

Fig 7: TEST CASE 2(ii)

EXPECTED: Successful Login

ACTUAL: Successful Logi

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

In conclusion, the development of an Inventory Management System using React for the frontend, Spring Boot Microservices for the backend, and MySQL for the database stands as a transformative endeavor with the potential to revolutionize inventory management practices. This project is poised to empower organizations with the tools needed to optimize resource allocation, reduce operational costs, and enhance competitiveness in an ever-evolving business landscape.

The core components and features outlined earlier form the cornerstone of a dynamic and efficient inventory management solution. As the project moves forward, exploring avenues for future advancements such as advanced AI integration for predictive analytics, IoT device integration for real-time monitoring, and global expansion to cater to diverse markets, promises to unlock new levels of efficiency and innovation in inventory management. This journey into modern inventory management technology holds the promise of reshaping how businesses manage their resources and adapt to the demands of a dynamic global market.

In summary, the development of an Inventory Management System using React for the frontend, Spring Boot Microservices for the backend, and MySQL for the database represents a strategic initiative to enhance inventory control and streamline operational efficiency. This project's key components, including user-friendly interfaces, real-time tracking, and customizable reports, lay the foundation for effective resource management. Looking ahead, potential areas for future enhancement encompass advanced AI integration.

7.2 FUTURE WORKS

Advanced AI Integration: Incorporating AI-driven functionalities such as demand forecasting, anomaly detection, and automated reordering to enhance inventory management and decision-making.

Enhanced Reporting and Analytics: Expanding data analytics capabilities to offer advanced reporting, data visualization, and predictive analytics for improved inventory insights.

IoT Device Integration: Connecting the system with IoT devices and sensors for real-time monitoring of inventory conditions, temperature, and humidity, allowing for proactive inventory control.

Supplier and Vendor Integration: Streamlining communication and transactions with suppliers and vendors through automated ordering and real-time updates, improving supply chain efficiency.

Multi-Warehouse Support: Extending the system's capabilities to manage inventory across multiple warehouses or locations, offering a comprehensive view of inventory assets.

Mobile Applications: Developing dedicated mobile applications for both iOS and Android platforms to enable convenient on-the-go access to critical inventory information and management.

Blockchain Implementation: Exploring blockchain technology for enhanced data security, traceability, and transparency in inventory management, especially in industries with stringent compliance requirements.

Global Expansion: Expanding the system's reach to international markets by supporting multiple currencies, languages, and regulatory requirements to cater to a diverse range of users.

Customization and Scalability: Enhancing the system's flexibility to accommodate unique business requirements, ensuring scalability as organizations grow and evolve.

REFERENCES

[1] React Official Documentation: https://reactjs.org/docs/getting-started.html

[2] Spring Boot Official Documentation: https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html

[3] MySQL Official Documentation: https://dev.mysql.com/doc/

[4] "Designing Social Interfaces" by Christian Crumlish and Erin Malone. O'Reilly Media, 2009.

[5] "Building Microservices" by Sam Newman. O'Reilly Media, 2015.

[6] "High-Performance MySQL" by Baron Schwartz, Peter Zaitsev, Vadim Tkachenko. O'Reilly Media, 2012.

[7] "User Experience Design: Creating Designs Users Really Love" by Gavin Doughtie and Peter Stahl. Smashing Magazine, 2016.

[8] "Social Media Marketing All-in-One For Dummies" by Jan Zimmerman and Deborah Ng. For Dummies, 2017.

[9] "The Art of Community: Building the New Age of Participation" by Jono Bacon. O'Reilly Media, 2012.

[10] "Online Privacy: A Reference Handbook" by Catherine D. Slade. ABC-CLIO, 2015.

[11] "Social Media Metrics: How to Measure and Optimize Your Marketing Investment" by Jim Sterne. Wiley, 2017.

[12] "Machine Learning: A Probabilistic Perspective" by Kevin P. Murphy. MIT Press, 2012.

[13] "Privacy and Security for Cloud Computing" by Siani Pearson and George Yee. Artech House, 2013.

[14] "Scalability Rules: 50 Principles for Scaling Web Sites" by Martin L. Abbott and Michael T. Fisher. Pearson Education, 2011.

[15] "Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design" by Michael J. Hernandez. Addison-Wesley, 2013.

[16] "The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses" by Eric Ries. Crown Publishing Group, 2011.

[17] "The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities" by Mark Dowd, John McDonald, and Justin Schuh. Addison-Wesley, 2006.

[18] "Social Media and Public Relations: Eight New Practices for the PR Professional" by Deirdre K. Breakenridge. Pearson Education, 2012.

[19] "Microservices in Action" by Morgan Bruce and Paulo A. Pereira. Manning Publications, 2018.

[20] "MySQL High Availability: Tools for Building Robust Data Centers" by Charles Bell, Mats Kandhal, and Lars Thalmann. O'Reilly Media, 2010.

APPENDICES

APPENDIX - I

FRONT END CODE

```
import React, { useEffect, useState } from 'react';
import axios from 'axios';
import {  useNavigate } from 'react-router-dom';
import '../Components/Login.css';
// import './Products.css';
const LoginForm = () => {
const [user, setName] = useState('');
const [mail, setMail] = useState('');
const [number, setNumber] = useState('');
const [pass, setPass] = useState('');
const [popupMessage, setPopupMessage] = useState('');
const [role, setRole] = useState('Customer');
const navigate = useNavigate();
const [emailExistsError, setEmailExistsError] = useState('');
const [mobileNumberExistsError, setMobileNumberExistsError] = useState('');
const [Incorrect, setIncorrect] = useState('');
const handleChangeName = (e) => {
const value = e.target.value;
setName(value);
};
const handleChangeNumber = (e) => {
const value = e.target.value;
setNumber(value);
};
const handleChangeMail = (e) => {
const value = e.target.value;
setMail(value);
```

```
};
  const handleChangePass = (e) => {
    const value = e.target.value;
    setPass(value);
  };
  const handleEmailBlur = async () => {
    if (mail) {
      try {
        const response = await axios.get(`http://127.0.0.1:8181/api/v1/auth/email-
exists?email=${mail}`);
        if (response.status === 200) {
          setEmailExistsError('');
        }
      } catch (error) {
        console.error('Error checking email:', error);
        setEmailExistsError('*Email already exists');
      }
    } else {
      setEmailExistsError('');
    }
  };
  const handleMobileNumberBlur = async () => {
    if (number) {
      try {
        const response = await axios.get(
          `http://127.0.0.1:8181/api/v1/auth/mobile-number-exists?mobileNumber=${number}`
        );
        if (response.status === 200) {
          const { exists } = response.data;
          if (exists) {
            setMobileNumberExistsError('*Mobile number already exists');
          } else {
```

```
          setMobileNumberExistsError('');
        }
      } else
        setMobileNumberExistsError('');
      }
    } catch (error) {
      console.error('Error checking mobile number:', error);
      setMobileNumberExistsError('');
    }
  } else {
    setMobileNumberExistsError('');
  }
};
useEffect(() => {
  const sign_in_btn = document.querySelector('#sign-in-btn');
  const sign_up_btn = document.querySelector('#sign-up-btn');
  const container = document.querySelector('.container');
  sign_up_btn.addEventListener('click', () => {
    container.classList.add('sign-up-mode');
  });
  sign_in_btn.addEventListener('click', () => {
    container.classList.remove('sign-up-mode');
  });
}, []);
const handleRoleChange = (selectedRole) => {
  setRole(selectedRole);
};
const handleSignUp = async (e) => {
  e.preventDefault();
```

```
  try {
    const response = await axios.post('http://127.0.0.1:8181/api/v1/auth/register', {
      name :user,
      email:mail,
      mobilenumber:number,
      password:pass,
    });
    if (response.status === 200) {
      setPopupMessage('Registered Successfully');
      navigate('/');
      setName('');
      setNumber('');
      setPass('');
      setMail('');
    }
  } catch (error) {
    console.error('Error: ', error);
  }
};
const handleSignIn = async (e) => {
  e.preventDefault();
  try {
    const response = await axios.post('http://127.0.0.1:8181/api/v1/auth/authenticate', {
      email: mail,
      password: pass,
    });
    let token = response.data.token;
    let user = response.data.userResponse;
    localStorage.setItem('token', token);
    localStorage.setItem('user', JSON.stringify(user));
```

```
    navigate('/home');
   } catch (error) {
    console.error('Error: ', error);
    setIncorrect('*Invalid credentials');
   }
  };
  const handlePopupClose = () => {
   setPopupMessage('');
  };
  return (
   <div className="container">
    <div className="forms-container">
     <div className="signin-signup">
      <form action="#" className="sign-in-form" onSubmit={handleSignIn}>
       <h2 className="title">Sign in</h2
       <div className="message" style={{fontSize:"25px"}}>
       {Incorrect && <p>{Incorrect}</p>}
       </div>
       <div className="input-field">
        <i className="fas fa-user"></i>
        <input required type="email" placeholder="Email" value={mail}
onChange={handleChangeMail} />
       </div>
       <div className="input-field">
        <i className="fas fa-lock"></i>
        <input required type="password" placeholder="Password" value={pass}
onChange={handleChangePass} />
       </div>
       <input type="submit" value="Login" className="btn solid" />
      </form>
      <form action="#" className="sign-up-form" onSubmit={handleSignUp}>
       <h2 className="title">Sign up</h2>
```

```
          <div className="input-field">
            <i className="fas fa-user"></i>
            <input required type="text" placeholder="Name" value={user}
onChange={handleChangeName} />
          </div>
          <div className="input-field">
            <i className="fas fa-envelope"></i>
            <input required type="email" placeholder="Email" value={mail}
onChange={handleChangeMail} onBlur={handleEmailBlur}/>
            </div>
            <div className="message">
            {emailExistsError && <p>{emailExistsError}</p>}
            </div>
          <div className="input-field">
            <i className="fas fa-envelope"></i>
            <input required type="number" placeholder="Mobile Number" value={number}
onChange={handleChangeNumber} onBlur={handleMobileNumberBlur}/>
            </div>
            <div className="message">
            {mobileNumberExistsError && (
              <p>{mobileNumberExistsError}</p>
              )}
              </div>
          <div className="input-field">
            <i className="fas fa-lock"></i>
            <input required type="password" placeholder="Password" value={pass}
onChange={handleChangePass} />
          </div>
          <input type="submit" className="btn" value="Sign up" />
        </form>
      </div>
      {popupMessage === 'Registered Successfully' && (
       <div className="popupContainer">
```

```
      <div className="popup">
        <p>{popupMessage}</p>
        <button className="popupClose" onClick={handlePopupClose}>
          Close
        </button>
      </div>
    </div>
  )}
</div>
{popupMessage === 'Password is Incorrect' && (
  <div className="popupContainer">
    <div className="popup">
      <p>{popupMessage}</p>
      <button className="popupClose" onClick={handlePopupClose}>
        Close
      </button>
    </div>
  </div>
)}
<div className="panels-container">
  <div className="panel left-panel">
    <div className="content">
      <h3>New here?</h3>
      <p>Enter your personal details and start the journey with us</p>
      <button className="btn transparent" id="sign-up-btn">
        Sign up
      </button>
    </div>
    <img src="img/log.svg" className="image" alt="" />
  </div>
  <div className="panel right-panel">
```

```
        <div className="content">
          <h3>One of us?</h3>
          <p>To keep connected with us please login with your personal info</p>
          <button className="btn transparent" id="sign-in-btn">
            Sign in
          </button>
        </div>
        <img src="img/register.svg" className="image" alt="" />
      </div>
    </div>
  </div>
 );
};
export default LoginForm
```

BACK-END CODE

```
package com.project.supermarket.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import com.project.supermarket.entity.Billing;
import com.project.supermarket.service.BillingService;
@CrossOrigin
@RestController
public class BillingController {
    @Autowired
    private BillingService ser;
    @PostMapping("/postPay")
    private  String SaveDetailsRequest(@RequestBody List<Billing> billingList) {
```

```
        ser.saveDetails(billingList);
        return "Saved details in billing list.";
    }
}
package com.project.supermarket.controller;
import java.io.IOException;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import com.project.supermarket.entity.Product;
import com.project.supermarket.service.ProductService;
@CrossOrigin
@Controller
public class ProductController {
    private final ProductService productService;
    private static final Logger logger = Logger.getLogger(ProductController.class.getName());
    @Autowired
    public ProductController(ProductService productService) {
        this.productService = productService;
    }
    @PostMapping("/products")
    public ResponseEntity<String> addProduct(
            @RequestParam("file") MultipartFile file,
            @RequestParam("name") String name,
            @RequestParam("price") String price,
            @RequestParam("stock") int stock
```

```
    ) {
        try {
            Product product = new Product();
            product.setName(name);
            product.setPrice(price);
            product.setStock(stock);
            String response = productService.addProduct(product, file);
            return ResponseEntity.status(HttpStatus.OK).body(response);
        } catch (IOException e) {
            logger.log(Level.SEVERE, "Error occurred during product addition", e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
occurred during product addition");
        } catch (Exception e) {
            logger.log(Level.SEVERE, "Unexpected error occurred during product addition", e);
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Unexpected
error occurred during product addition");
        }
    }
    @GetMapping("/product")
    public ResponseEntity<List<Product>> getAllProducts() {
        try {
            List<Product> menProducts = productService.getAllProducts();
            return ResponseEntity.ok(menProducts);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).build();
        }
    }
    }
package com.project.supermarket.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Lob;
```

```
import jakarta.persistence.Table;
@Entity
@Table(name = "product")
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String price;
    private int stock;
    @Lob
    @Column(length = 2097152)
    private byte[] image;
    public byte[] getImage() {
        return image;
    }
    public void setImage(byte[] image) {
        this.image = image;
    }
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getPrice() {
        return price;
    }
    public void setPrice(String price) {
```

```
        this.price = price;
    }
    public int getStock() {
        return stock;
    }
    public void setStock(int stock) {
        this.stock = stock;
    }
    public Product() {
        super();
    }
    public Product(Long id, String name, String category, String price, int stock, byte[] image) {
        super();
        this.id = id;
        this.name = name;
        this.price = price;
        this.stock = stock;
        this.image = image;
    }
}
package com.project.supermarket.entity;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
@Entity
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Billing {
```

```
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private Long productId;
    private String productName;
    private int productQuantity;
    private int productPrice;
    private int total;
}
```

APPENDIX – II

SCREENSHOTS

HOME PAGE

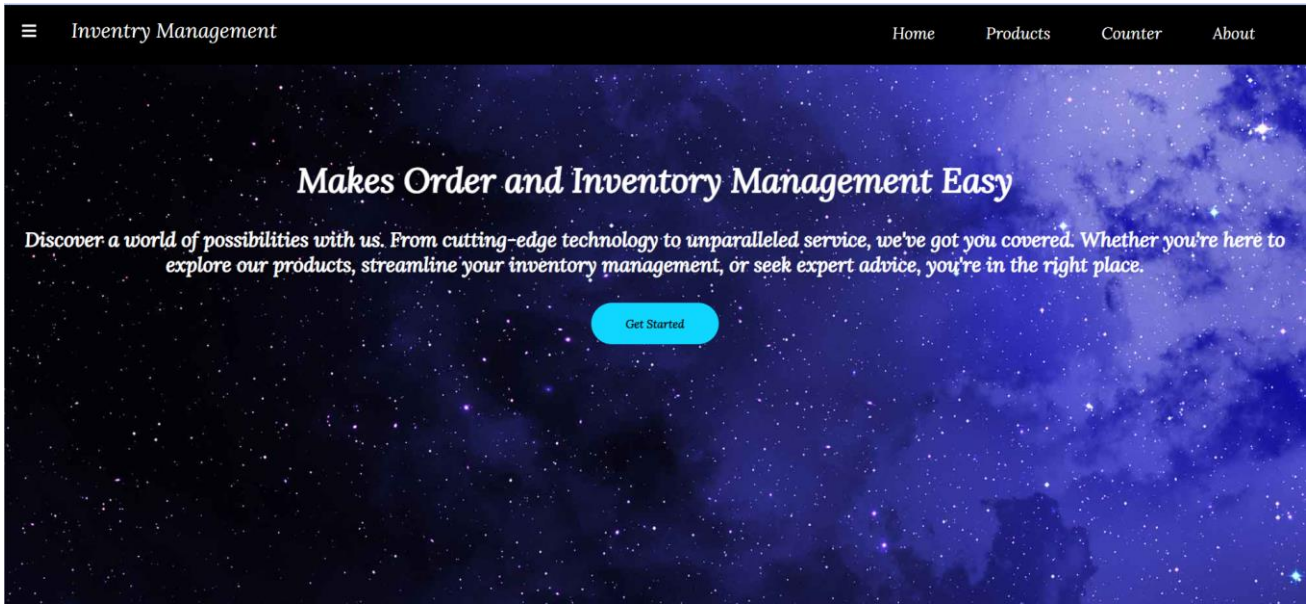

Fig 8: HOME PAGE

PRODUCT PAGE

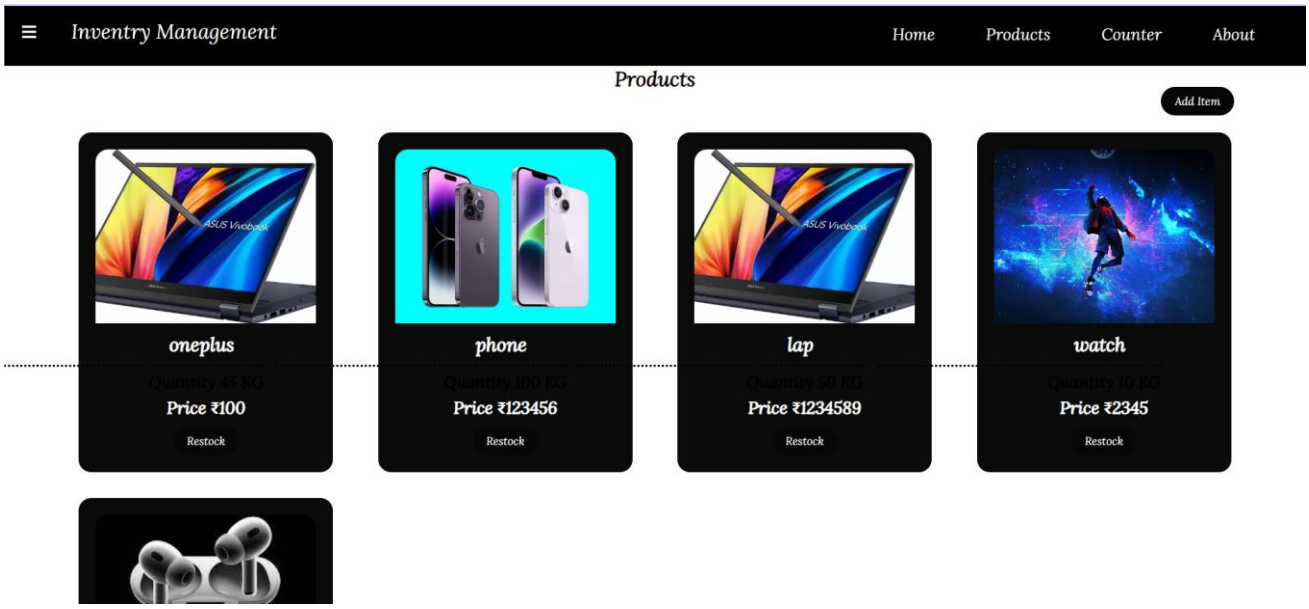

Fig 9: PRODUCT PAGE

ADD ITEM PAGE

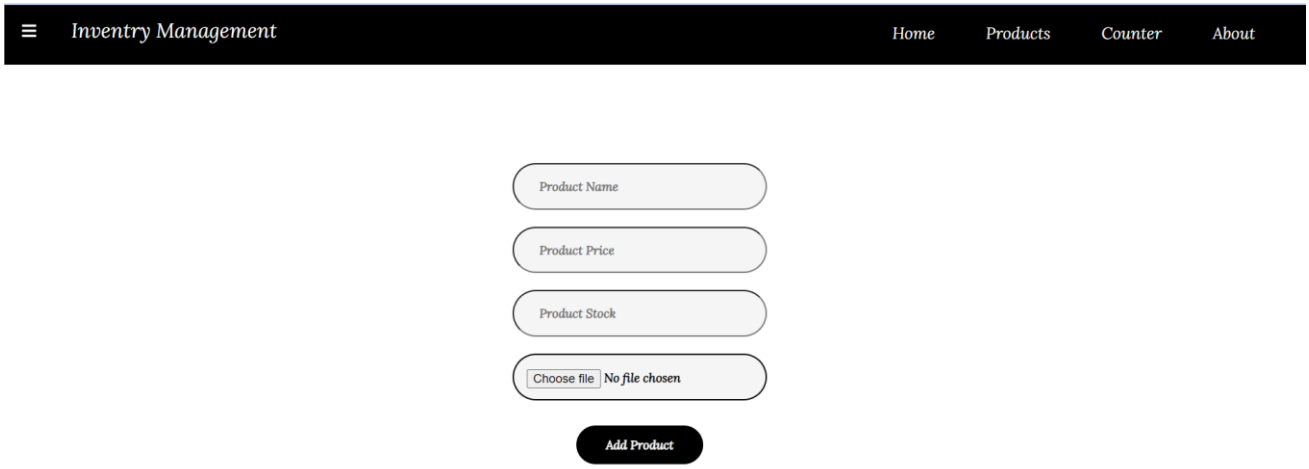

Fig 10: ADD ITEM PAGE

BILLING PAGE

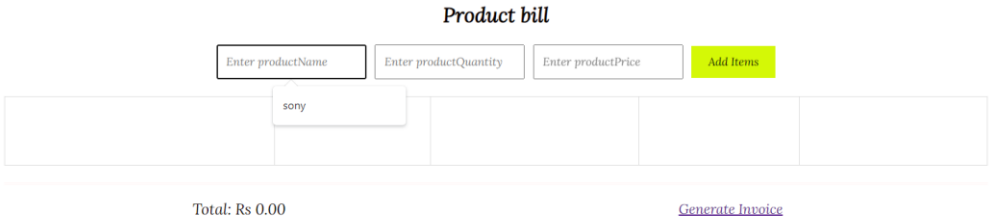

Fig 11: BILLING PAGE

RECEIPT PAGE

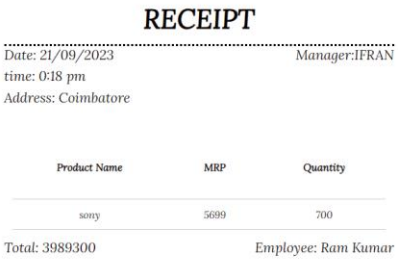

Fig 12: RECEIPT PAGE

FEEDBACK PAGE